QATra Irrigation System

Progress Report

Texas A&M University at Qatar
ECEN 404: Electrical Design Lab II

Group Members:
Maryam Al-Emadi
Roqayya AlYousef
Fatima Al-Janahi
Noof Al-Sayed
Mentor: Dr. Hazem Nounou
Course instructor: Dr. Ali Ghrayeb
Date: 30 March 2020

Texas A&M at Qatar University, March 2020
“An aggie doesn’t lie, cheat or steal, or tolerate those who do.”

TABLE OF CONTENTS

Abstract

Qatar has limited natural water resources, as it has no rivers or lakes. In addition, rain is very rare in Qatar with an annual average of 80 mm of rain water [1]. Groundwater in Qatar supplies the country with 62 million m^3 of water per year, but it does not cover the water needs of the country. Therefore, desalination is being used for domestic usage and treating used water is mainly being used for agriculture. Seawater desalination process is costly and uses natural gas as fuel (a non-renewable energy source), which increases the emission of carbon dioxide in Qatar and accelerates global warming [2]. Cultivation can reduce the impact of global warming, and managing the amount of water being used for cultivation can preserve our natural resources. Hence, QATra irrigation system was designed. QATra is an advanced irrigation system that is able to reduce water consumption by checking the soil moisture and controlling the amount of water dispensed. A mobile application, where the user can view the daily water content of the soil and get notified when the plants are watered, is used in our system. The user is informed in case of a fault in the system and can control the irrigation system using the application. The project aims to address environmental issues, cut labor costs, encourage smart innovations and boost cultivation.

In this paper, we will first discuss the visual prototype of the mobile application and the program code of the different microcontrollers. Then, we will discuss the functional prototyping, testing, troubleshooting and experimental results of various parts of the system. Last, we will discuss our progress relative to the timeline we proposed at the beginning of the project and provide verification, future recommendations, improvements and optimizations for the system.

Visual Prototyping and Program Code

Our system consists of various components that are linked together. To develop a better understanding of our system and illustrate how each component should work, program flows and diagrams were created for our three main components.

- **Mobile Application**

The IOS mobile application is one of the main components in our project. The purpose of the mobile application is to give the user the ability to control the irrigation system, monitor the moisture levels, and get informed in case of potential fault in the system. While building the mobile application, some features were taken into consideration such as the ease and friendliness of the mobile application to assist our audience adaptation to the new technology. **Figure 1** shows the suggested diagram of our mobile application.

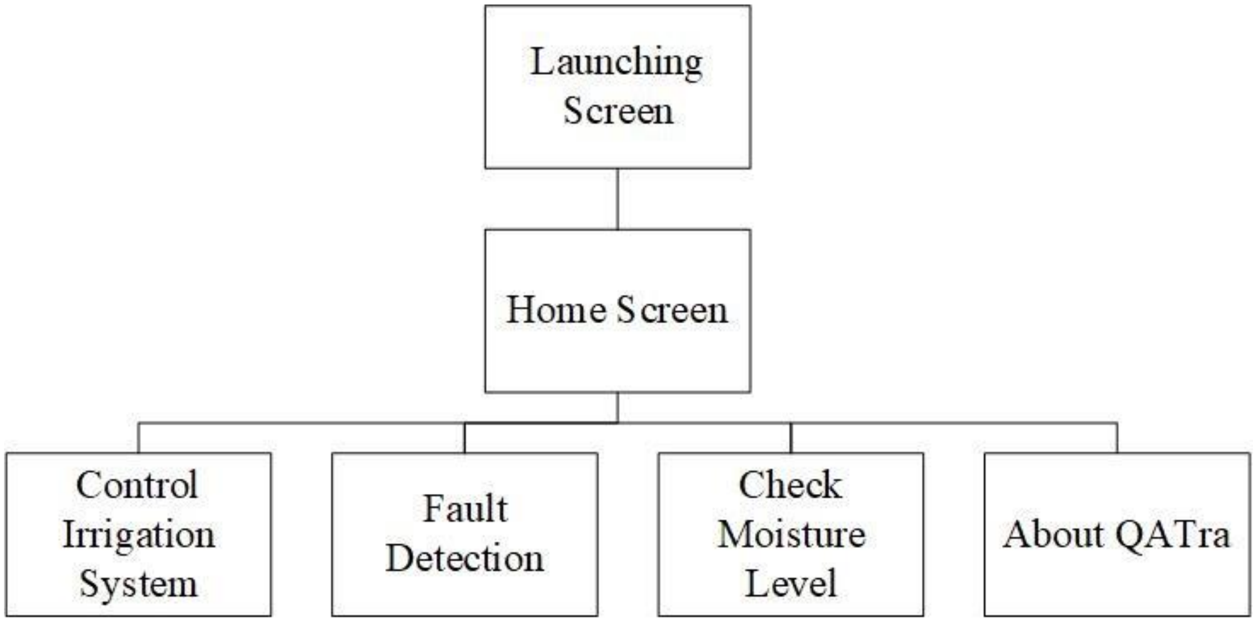


Figure 1: Diagram of the IOS mobile application.

- **End Node**

The main function of the end node microcontroller is to take measurements of the soil moisture level and transmit it to the main microcontroller. **Figure 2** shows the program code flowchart for the end node. The end node consists of a microcontroller that is connected to the soil moisture sensor and to a transceiver. As shown in the figure, the grove soil moisture sensor takes the readings of the water content in the soil and then it converts the data from integers to characters. This step is important because the NRF24L01 transceiver only operates by transmitting characters from one chip to another. Once the conversion is done, the sensor reading gets transmitted to the main microcontroller using the NRF24L01 transceiver.

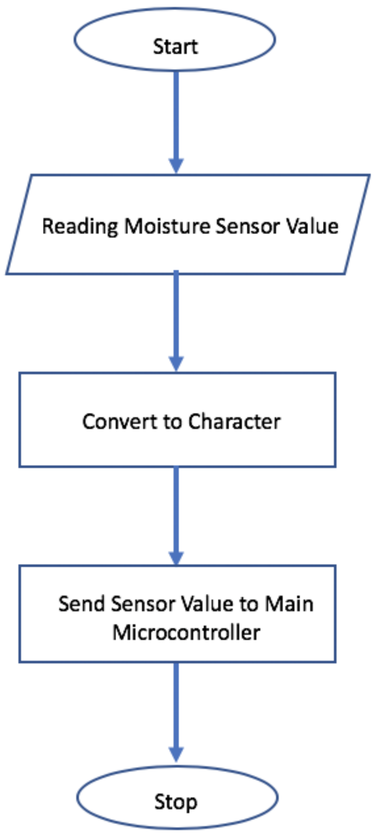

Start
Reading Moisture Sensor Value
Convert to Character
Send Sensor Value to Main Microcontroller
Stop

Figure 2: Flowchart of program code of end node.

- **Main Microcontroller**

The main microcontroller is the main link between the mobile application, the water pump/sprinklers and the end node. **Figure 3** shows the program code flowchart for the main microcontroller with only one end node for simplicity. As shown in **Figure 3**, the user can input through the mobile application the type of plant for each end zone. The main microcontroller then receives the soil moisture level data from the end node through the transceiver. Next, it checks if there is a potential fault in the system and if there is, it notifies the user through the mobile application that there is a potential fault in the system. If there is no fault, it checks if the system is controlled automatically or manually by the user. If it is controlled manually, then it'll be controlled by the user through the mobile application. If it is controlled automatically, then the system will run by itself. This is done by first converting back the moisture level data from character to integer. Then the code will check if the soil moisture level is below a certain threshold and if this condition is met, the main microcontroller will instruct to turn the water pump on. This process will continue through a loop until the condition is no longer valid.

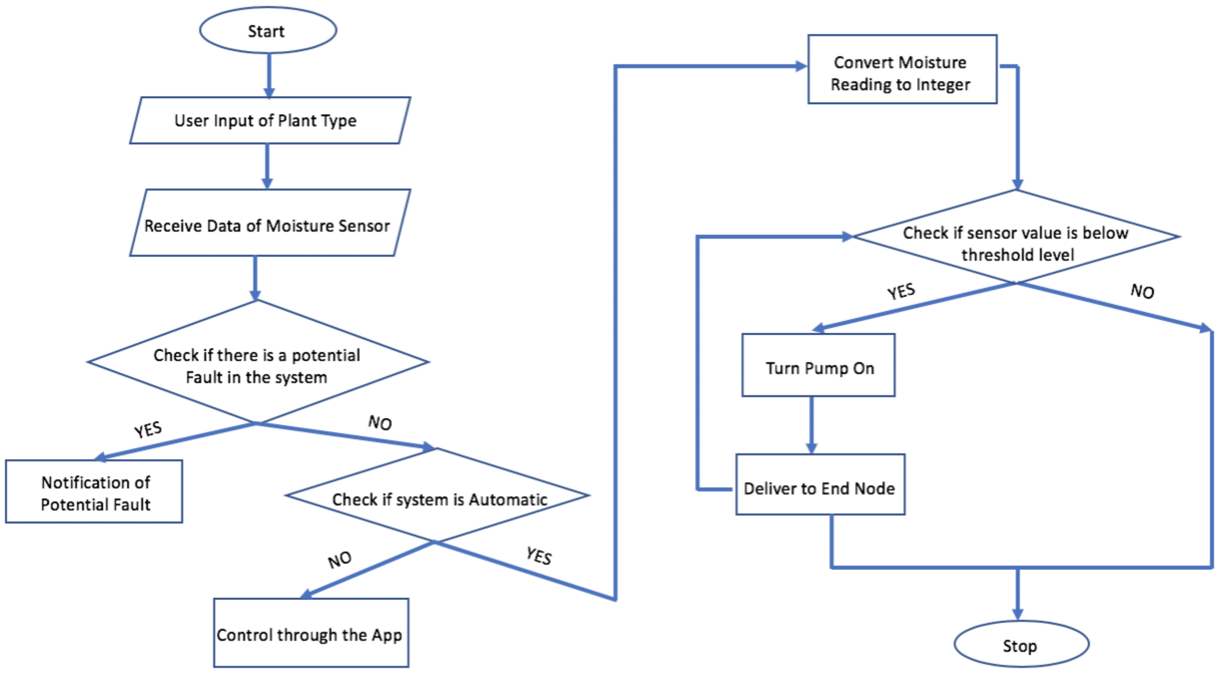

Start
User Input of Plant Type
Receive Data of Moisture Sensor
Check if there is a potential Fault in the system
YES
NO
Notification of Potential Fault
Check if system is Automatic
NO
YES
Control through the App
Convert Moisture Reading to Integer
Check if sensor value is below threshold level
YES
NO
Turn Pump On
Deliver to End Node
Stop

Figure 3: Flowchart of program code of main microcontroller.

Functional Prototyping, Testing, Troubleshooting, and Experimental Results

- **Testing and Experimental Results**

After we gained understanding on how each component should work; the system was implemented, the codes were written, and the mobile application was built. Meanwhile, each part of the system was tested to ensure a smooth operation of the whole system.

- End Node

As mentioned earlier, the end node is responsible for taking soil moisture measurements and sending them to the main microcontroller. Thus, each end node, shown in **Figure 4**, is connected to a moisture sensor (to take soil moisture level) and a NRF24l01 transceiver (to send data to the main microcontroller).

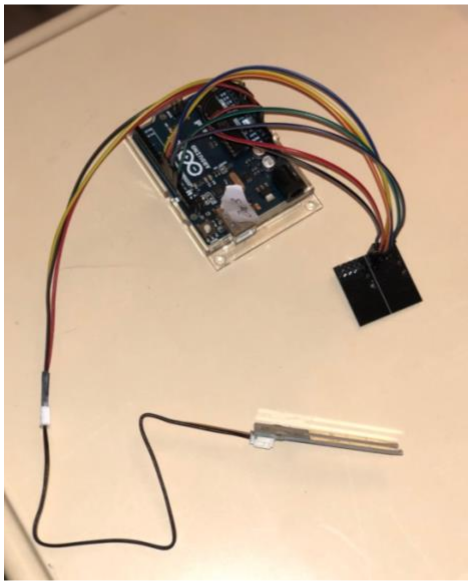
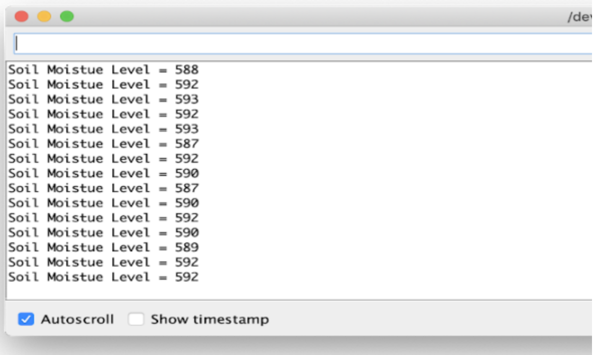


Figure 4: End node microcontroller connections.

First, a program was written to ensure a proper moisture level reading from the end node moisture sensor. The output of the program is shown in Figure 5.



```
Soil Moistue Level = 588
Soil Moistue Level = 592
Soil Moistue Level = 593
Soil Moistue Level = 592
Soil Moistue Level = 593
Soil Moistue Level = 587
Soil Moistue Level = 592
Soil Moistue Level = 590
Soil Moistue Level = 587
Soil Moistue Level = 590
Soil Moistue Level = 592
Soil Moistue Level = 590
Soil Moistue Level = 589
Soil Moistue Level = 592
Soil Moistue Level = 592
```

Figure 5: Testing of soil moisture sensor.

- Main Microcontroller

As previously stated, the main microcontroller is the link between the mobile application, irrigation system, and end nodes. Thus, the main microcontroller (Figure 6) is connected to the mobile application using firebase server (to send and receive data), to the relay, pumps, and sprinklers (to irrigate the plants based on user's input), and NRF24L01 transceiver (to receive soil moisture sensor reading from the end node microcontrollers).

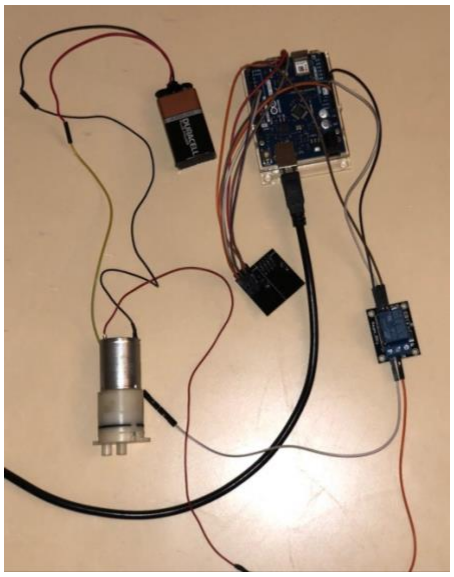
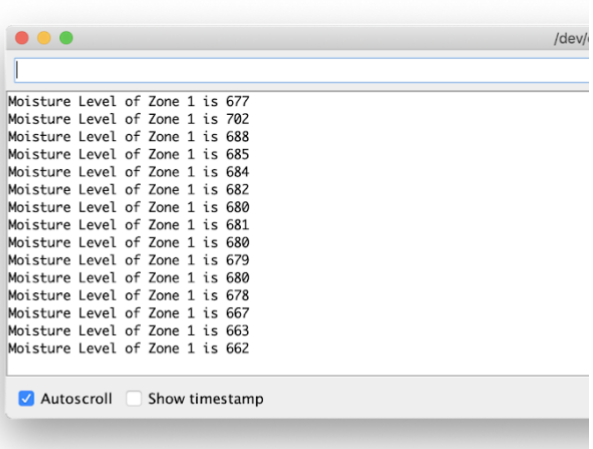


Figure 6: Main microcontroller connections.

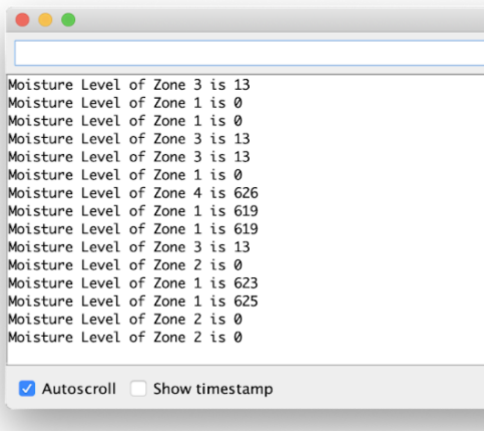
To test the data transfer between the end node microcontroller and the main microcontroller, an arduino code was developed. The main aim of this arduino code was to ensure that the main microcontroller is successfully receiving the soil moisture readings from the end node microcontroller. The result of this code is shown in Figure 7.



```
Moisture Level of Zone 1 is 677
Moisture Level of Zone 1 is 702
Moisture Level of Zone 1 is 688
Moisture Level of Zone 1 is 685
Moisture Level of Zone 1 is 684
Moisture Level of Zone 1 is 682
Moisture Level of Zone 1 is 680
Moisture Level of Zone 1 is 681
Moisture Level of Zone 1 is 680
Moisture Level of Zone 1 is 679
Moisture Level of Zone 1 is 680
Moisture Level of Zone 1 is 678
Moisture Level of Zone 1 is 667
Moisture Level of Zone 1 is 663
Moisture Level of Zone 1 is 662
```

Figure 7: Main microcontroller receiving soil moisture level from zone.

After confirming that the wireless connection between the main microcontroller and zone 1 end node, the code was applied for the three other end nodes. The communication between the main microcontroller and the four end nodes was successful, as the main microcontroller was able to receive the soil moisture level for the four end nodes (Figure 8).



```
Moisture Level of Zone 3 is 13
Moisture Level of Zone 1 is 0
Moisture Level of Zone 1 is 0
Moisture Level of Zone 3 is 13
Moisture Level of Zone 3 is 13
Moisture Level of Zone 1 is 0
Moisture Level of Zone 4 is 626
Moisture Level of Zone 1 is 619
Moisture Level of Zone 1 is 619
Moisture Level of Zone 3 is 13
Moisture Level of Zone 2 is 0
Moisture Level of Zone 1 is 623
Moisture Level of Zone 1 is 625
Moisture Level of Zone 2 is 0
Moisture Level of Zone 2 is 0
```

Figure 8**:** Main microcontroller receiving soil moisture levels from four zones.

Then, the arduino code was developed to activate the pump and irrigate the plants based on the soil moisture levels reading and the user input from the mobile application. Since the user is able to choose the irrigation mode (automatic or manual) and to choose the plant type (to set minimum and maximum moisture level threshold).

- Mobile Application

The mobile application was developed using xCode, an integrated development environment for IOS applications. The purpose of the mobile application is to give the user the ability to control and monitor the irrigation system. The friendliness and ease of the mobile application was considered while developing the mobile application for our targeted audience. The following **Figure 9** shows the launching screen of our application. The launching screen shows our logo for a few seconds and then automatically moves to the main screen. The main screen of the mobile application shows several options (**Figure 10**). The options are: control the irrigation system, fault detection, check the soil moisture levels in different zones, and learn about QATra.

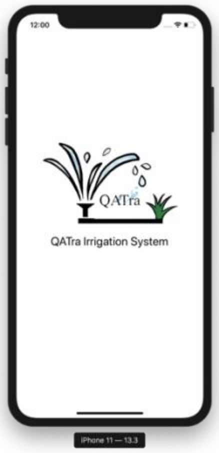


Figure 9: Launching screen.

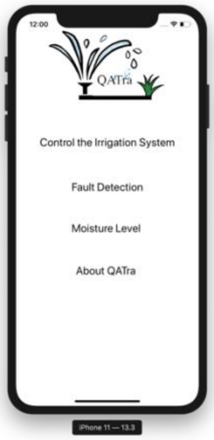


Figure 10: Main screen.

Control irrigation system page (**Figure 11**) will give the user the choice to choose the irrigation mode and the plant type for each of the four zones. While the fault detection page (**Figure 12**), informs the user of the possible faults in the system. The moisture level page (**Figure 13**) is capable to view the soil moisture level for the four zones. Finally, the last page (**Figure 14**) shows a small paragraph about QATra Irrigation System.

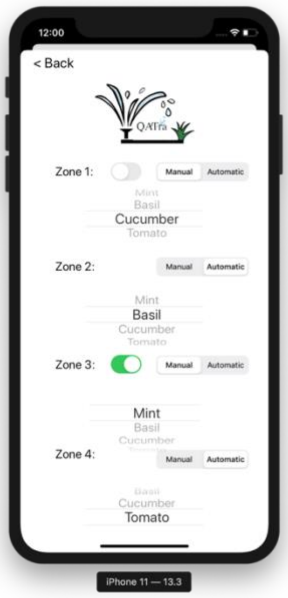


Figure 11: Control the irrigation system page.

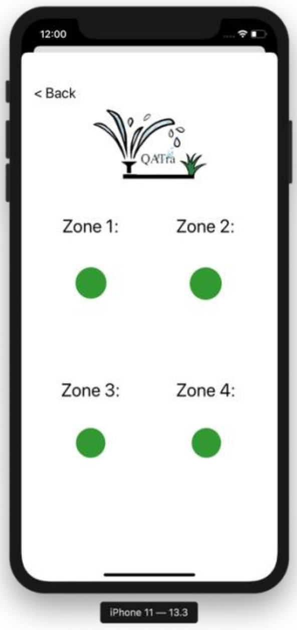


Figure 12: Fault detection page.

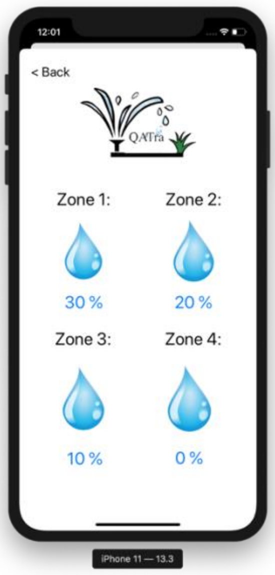
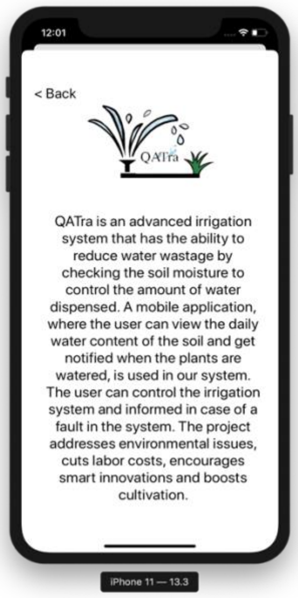


Figure 13: Soil moisture level page.



QATra is an advanced irrigation system that has the ability to reduce water wastage by checking the soil moisture to control the amount of water dispensed. A mobile application, where the user can view the daily water content of the soil and get notified when the plants are watered, is used in our system. The user can control the irrigation system and informed in case of a fault in the system. The project addresses environmental issues, cuts labor costs, encourages smart innovations and boosts cultivation.

Figure 14: About QATra page.

The mobile application is ready to be deployed, as it was successfully tested on a real iPhone. Also, it is connected to the firebase server, which will allow the communication between the mobile application and the main microcontroller.

- **Functional Prototype**

Figure 15 shows the functional prototype of the smart irrigation system. The system consists of many parts that includes microcontrollers, water pumps, relay switches, sprinklers, moisture sensors, transceivers, batteries and a mobile application. As shown in **Figure 16**, the system has two main inputs which are the soil moisture sensor readings and the type of plant the user is inputting to each end zone. The main function of the system is that it takes the soil moisture readings for each zone and checks if the soil water content is below the minimum threshold or above a maximum threshold. Whenever it is at or under a minimum threshold it automatically pumps water to irrigate the plants in order to prevent it from falling below this minimum threshold and it stops the irrigation if it reaches the maximum threshold. What makes the system more sophisticated is that the user, through the mobile application, can turn the system on or off and can monitor the soil moisture level for each zone. Moreover, if the system detects a potential fault, it informs the user by sending a notification to the mobile application.

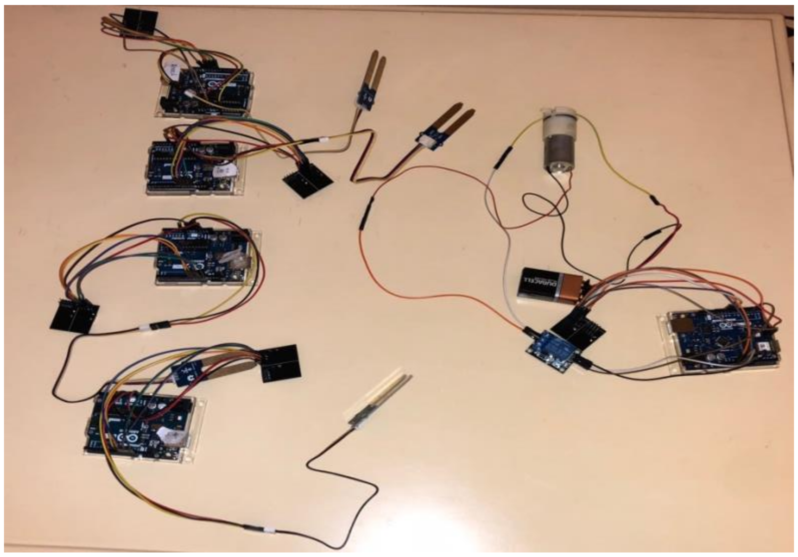


Figure 15: Functional prototype of QATra irrigation system.

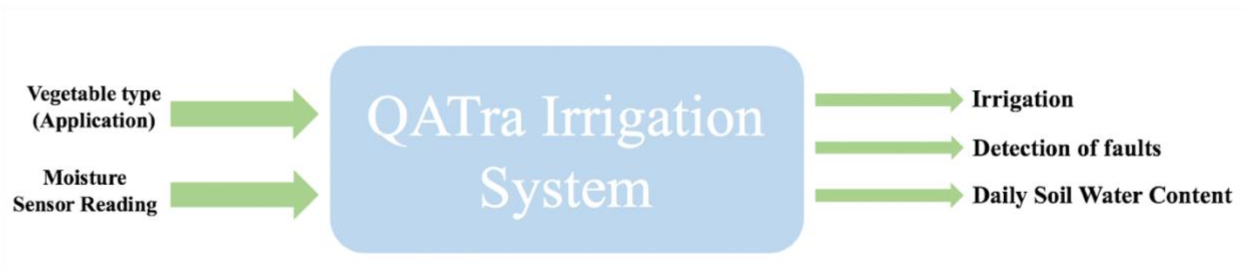


Figure 16: System high level representation.

- **Troubleshooting**

During our process of building our system, we faced some problems. However, these problems were solved.

- Transceiver NRF24L01

After connecting NRF24L01 transceivers to two Arduinos (microcontrollers), we tried to send strings as a starting point to check whether the connection was secured. Strings were received at the receiver node Arduino. However, when we connected the soil moisture level sensor to the transmitter node Arduino and tried sending integers, the receiver node Arduino was receiving either zeros or question marks. We overcame that problem by changing the type of the parameters that are transmitted and received by the NRF24L01 transceiver to integers and unsigned characters.

- Raspberry pi (Gateway)

In our proposed design, the gateway was the link between the main microcontroller (central node) and the server. During the programming and testing stage of the Raspberry Pi with Arduino, NRF24L01 transceivers were used to send and receive data. However, 255 was being received all of the time no matter what we sent. We decided to eliminate the Raspberry pi from our system and make the central node Arduino as the link between the system and the server.

- The wifi connection of Central node Arduino after eliminating the Raspberry pi (Gateway)

The Arduino is a microcontroller, unlike the Raspberry pi which is a computer that can enable Wi-fi easily. After researching, we found a Wi-fi micro-chip that can be used to connect the Arduino to the internet. This chip is ESP8266. We talked to Mr. Wesam regarding this and he ordered the Wi-fi micro-chip for us. When we received the ESP8266, we started programming it but kept getting errors. Mr. Wesam provided another type of Arduino called Arduino Uno Wifi Rev 2. The Wi-fi connection worked because the new Arduino has a Wifi library that can be used. We scanned the networks nearby then connected to the Wifi successfully by providing the network name and password in the code.

<u>Our Progress Relative to the Timeline</u>

Until March 9th, our project was on track. The code of the main microcontroller was finalized, and the fault detection algorithm was established. However, due to COVID-19 circumstances, the university closed its doors. Thus, one of our members (Fatima) took the equipment to her house. Our group agreed on splitting the code between us, writing the code, and sending the code to Fatima to test it. Also, the current situation makes it hard for the team to fully merge the different parts of the system together (i.e connecting the sprinkles, 4 zones and the mobile application together) and so we are still negotiating a way to get it done before the demo day. We are slightly behind our timeline. According to the timeline, by this time we should be finished with the system and started with testing the system as a whole. Also, the change that happened with regards to the demo day made us change our plan to what will be presented and how it will be presented in the showcase. The fact that the project needs to be videotaped and submitted a day before the 21st of April, made us modify our time plan to accommodate those changes. We are slightly behind. Nevertheless, we are able to catch up and deliver the prototype on April 21st.

Conclusion

The next steps that need to be taken for assembling the project and accomplishing its functions are: implementing the fault detection algorithm in the main microcontroller code, securing the connection between the different parts of the system, and resuming the testing to ensure reliable operation of the system. Testing the system will be till the demo day. On the demo day, we plan to present a video that shows 4 zones/end nodes (4 different plants) working with the system. We will remove the moisture sensor from the plant of one of the end nodes, so that the system will detect a fault and show it on the app. We will show how one end node can be turned off and controlled manually. For the last two end nodes, one of the plants' moisture level will be above the maximum threshold (to show how the system works automatically) and the other plants' moisture level will be under the minimum threshold to show how the sprinkler will be turned on.

For verification, it is important to note that separate parts of the system were verified instead of the system as a whole. First, the mobile application was tested on the virtual device, which showed us the layout of the application, whether the transitions between the pages were working, and the application as a whole was running. Later, the mobile application was installed on an iPhone, and we verified that it can be deployed. Second, the moisture sensor manual said that the moisture sensor level is less than 300 for dry soil and between 300 and 700 for moist soil. Therefore, we tested the moisture sensor with different types of soil and measured the soil moisture level to verify whether the moisture sensors were working properly or not.

For improvements, we believe that the system can be cheaper. Now, the system costs 520 dollars. This price includes the irrigation system (pumps, relays, sprinklers and pipes) as well as the controller part. Therefore, it is more expensive than other smart irrigation systems, since they include the controllers only. If the system was to be fabricated all at once, instead of buying the different components in our system from different companies (i.e. if we manufactured those components) and the system was mass produced, then the system will be cheaper. We realized that the price can be a financial constraint, and therefore we believe an improvement to it is necessary. Moreover, we believe that the fault detection method can be optimized. Since currently it can only notify the user in case of a potential fault, without specifying where the exact fault is. The reason for that is because fault detection is a whole field on its own, and we decided that we will partially implement it in our system. For future, we or other groups that wish to improve our project can enhance the fault detection algorithm.

For future recommendations, either if we continue developing this project or other people continue with it, we believe there are additional features that can be added. Those features are weather intelligence and multi-language. Our project is specifically designed for Qatar. Since rain is very rare in Qatar, we believed that a weather sensor is not necessary. However, if the app was to be developed for different countries, a weather sensor can enhance the performance of the system. Also, the app can be linked to a weather forecast, in order to plan irrigation schedules accordingly.

Moreover, the language used in our mobile application is English, since it is the most widely spoken language around the world. Other languages can be implemented like Arabic, Urdu, etc. to make the system more accessible and universal.

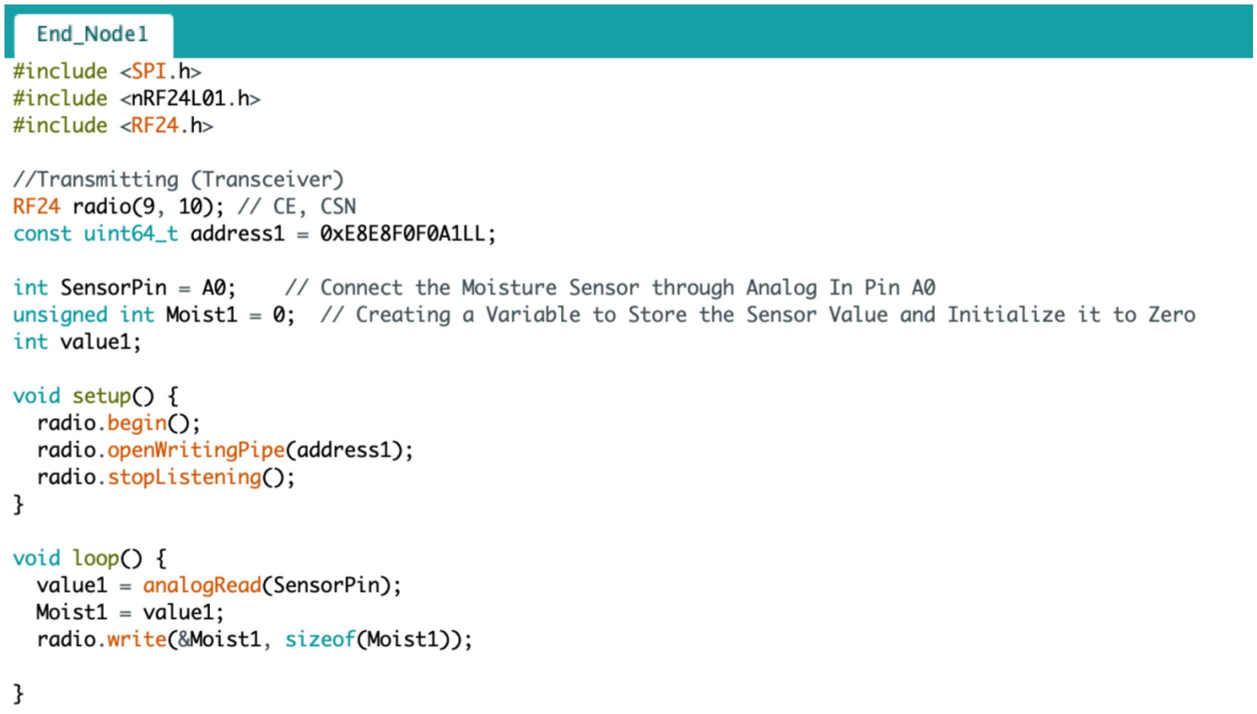
Qatar National Vision 2030 aims to achieve sustainable development to ensure a better life for future generations. QATra hopes to contribute to this vision.

References:

[1] H. M. Baalousha and O. K. M. Ouda, “Domestic water demand challenges in Qatar,” Arabian Journal of Geosciences, vol. 10, no. 24, Dec. 2017.
[2] M. Darwish and R. Mohtar, “Qatar water challenges,” Desalination and Water Treatment, vol. 51, no. 1-3, Jun. 2012.

Appendix:

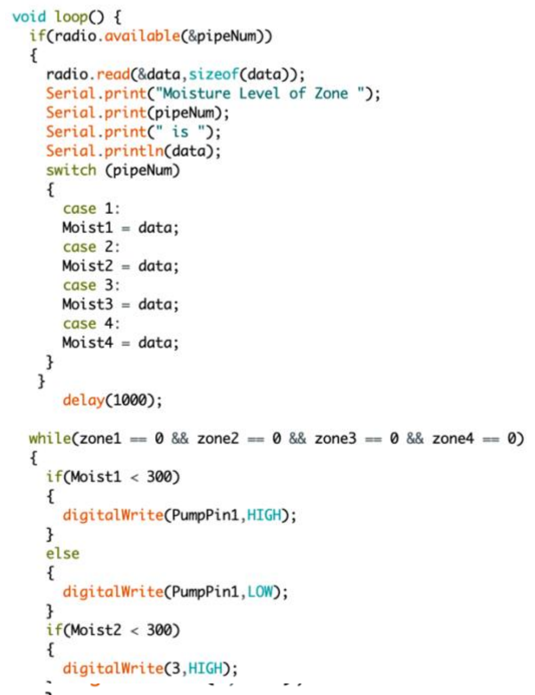
- **End Node Code**



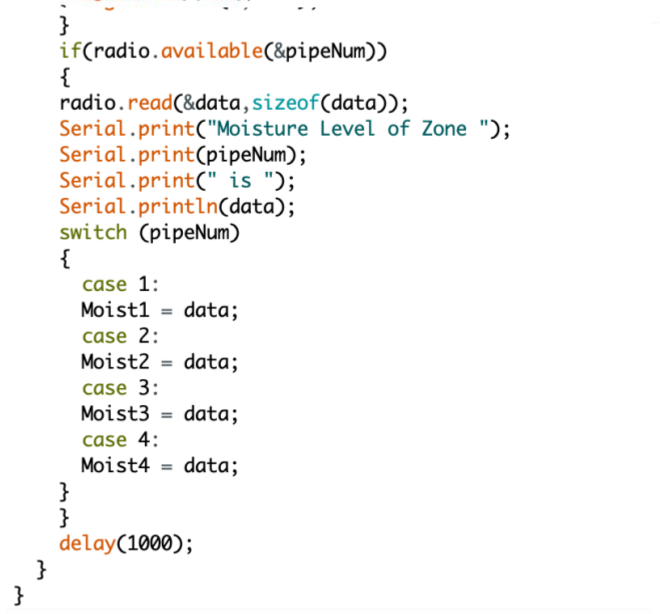
End_Node1

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

//Transmitting (Transceiver)
RF24 radio(9, 10); // CE, CSN
const uint64_t address1 = 0xE8E8F0F0A1LL;

int SensorPin = A0;     // Connect the Moisture Sensor through Analog In Pin A0
unsigned int Moist1 = 0;  // Creating a Variable to Store the Sensor Value and Initialize it to Zero
int value1;

void setup() {
  radio.begin();
  radio.openWritingPipe(address1);
  radio.stopListening();
}

void loop() {
  value1 = analogRead(SensorPin);
  Moist1 = value1;
  radio.write(&Moist1, sizeof(Moist1));

}
```

- **Main Microcontroller Code**



```
void loop() {
  if(radio.available(&pipeNum))
  {
    radio.read(&data,sizeof(data));
    Serial.print("Moisture Level of Zone ");
    Serial.print(pipeNum);
    Serial.print(" is ");
    Serial.println(data);
    switch (pipeNum)
    {
      case 1:
      Moist1 = data;
      case 2:
      Moist2 = data;
      case 3:
      Moist3 = data;
      case 4:
      Moist4 = data;
    }
   }
      delay(1000);

  while(zone1 == 0 && zone2 == 0 && zone3 == 0 && zone4 == 0)
  {
    if(Moist1 < 300)
    {
      digitalWrite(PumpPin1,HIGH);
    }
    else
    {
      digitalWrite(PumpPin1,LOW);
    }
    if(Moist2 < 300)
    {
      digitalWrite(3,HIGH);
```



```
    }
    if(radio.available(&pipeNum))
    {
    radio.read(&data,sizeof(data));
    Serial.print("Moisture Level of Zone ");
    Serial.print(pipeNum);
    Serial.print(" is ");
    Serial.println(data);
    switch (pipeNum)
    {
      case 1:
      Moist1 = data;
      case 2:
      Moist2 = data;
      case 3:
      Moist3 = data;
      case 4:
      Moist4 = data;
    }
    }
    delay(1000);
  }
}
```

- Mobile Application Code

- Control Irrigation System Code

```
- //
- //  ViewController.swift
- //  QATra Irrigation System
- //
- //  Created by Fatima Al-Janahi on 2/6/20.
- //  Copyright © 2020 Fatima Al-Janahi. All rights reserved.
- //
-
- import UIKit
-
- class ViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource {
-     //**************************************************************
-     //*************Zone 1*****************************************
-
-     @IBOutlet weak var pickerView1: UIPickerView!
-     @IBOutlet weak var control1: UISegmentedControl!
-     @IBOutlet weak var switch1: UISwitch!
-
-     override func viewDidLoad() {
-         super.viewDidLoad()
-         pickerView1.dataSource = self
-         pickerView1.delegate = self
-         pickerView2.dataSource = self
-         pickerView2.delegate = self
-         pickerView3.dataSource = self
-         pickerView3.delegate = self
-         pickerView4.dataSource = self
-         pickerView4.delegate = self
-
-         let chosen1 = UserDefaults.standard.integer(forKey: "chosen1")
-         let chosen2 = UserDefaults.standard.integer(forKey: "chosen2")
-         let chosen3 = UserDefaults.standard.integer(forKey: "chosen3")
-         let chosen4 = UserDefaults.standard.integer(forKey: "chosen4")
-         pickerView1.selectRow(chosen1, inComponent: 0, animated: false)
-         pickerView2.selectRow(chosen2, inComponent: 0, animated: false)
-         pickerView3.selectRow(chosen3, inComponent: 0, animated: false)
-         pickerView4.selectRow(chosen4, inComponent: 0, animated: false)
-
```

```
-
-         let zone1 = UserDefaults.standard.integer(forKey: "zone1")
-         let zone1On = UserDefaults.standard.integer(forKey: "zone1On")
-         if zone1 == 0
-         {
-             control1.selectedSegmentIndex = 1
-             switch1.isHidden = true
-         }
-         else
-         {
-             control1.selectedSegmentIndex = 0
-             switch1.isHidden = false
-         }
-
-         if zone1On == 0
-         {
-             switch1.isOn = false
-         }
-         else
-         {
-             switch1.isOn = true
-         }
-
-         let zone2 = UserDefaults.standard.integer(forKey: "zone2")
-         let zone2On = UserDefaults.standard.integer(forKey: "zone2On")
-         if zone2 == 0
-         {
-             control2.selectedSegmentIndex = 1
-             switch2.isHidden = true
-         }
-         else
-         {
-             control2.selectedSegmentIndex = 0
-             switch2.isHidden = false
-         }
-
-         if zone2On == 0
-         {
-             switch2.isOn = false
-         }
-         else
-           {
-               switch2.isOn = true
-           }
-
-           let zone3 = UserDefaults.standard.integer(forKey: "zone3")
-           let zone3On = UserDefaults.standard.integer(forKey: "zone3On")
-           if zone3 == 0
-           {
-               control3.selectedSegmentIndex = 1
-               switch3.isHidden = true
-           }
-           else
-           {
-               control3.selectedSegmentIndex = 0
-               switch3.isHidden = false
-           }
-
-           if zone3On == 0
-           {
-               switch3.isOn = false
-           }
-           else
-           {
-               switch3.isOn = true
-           }
-
-           let zone4 = UserDefaults.standard.integer(forKey: "zone4")
-           let zone4On = UserDefaults.standard.integer(forKey: "zone4On")
-           if zone4 == 0
-           {
-               control4.selectedSegmentIndex = 1
-               switch4.isHidden = true
-           }
-           else
-           {
-               control4.selectedSegmentIndex = 0
-               switch4.isHidden = false
-           }
-
-           if zone4On == 0
-           {
```

```
-           switch4.isOn = false
-       }
-       else
-       {
-           switch4.isOn = true
-       }
-   }
-
-
-   @IBAction func index1(_ sender: UISegmentedControl) {
-       switch control1.selectedSegmentIndex
-       {
-       case 0:
-           UserDefaults.standard.set(1, forKey: "zone1")
-           switch1.isHidden = false
-           print("zone 1 manual")
-       case 1:
-           UserDefaults.standard.set(0, forKey: "zone1")
-           switch1.isHidden = true
-           print("zone 1 automatic")
-       default:
-           UserDefaults.standard.set(1, forKey: "zone1")
-           switch1.isHidden = false
-           print("zone 1 manual")
-       }
-
-   }
-
-   @IBAction func switch1control(_ sender: UISwitch) {
-       if switch1.isOn
-       {
-           UserDefaults.standard.set(1, forKey: "zone1On")
-           print("zone 1 manual on")
-       }
-       else
-       {
-           UserDefaults.standard.set(0, forKey: "zone1On")
-           print("zone 1 manual off")
-       }
-   }
-
```

```
-
-       let veg = ["Mint",
-       "Basil",
-       "Cucumber",
-       "Tomato"]
-
-       //******************************************
-       //*********ZONE 2*******************
-
-       @IBOutlet weak var pickerView2: UIPickerView!
-       @IBOutlet weak var control2: UISegmentedControl!
-       @IBOutlet weak var switch2: UISwitch!
-
-       @IBAction func index2(_ sender: UISegmentedControl) {
-           switch control2.selectedSegmentIndex
-           {
-           case 0:
-               UserDefaults.standard.set(1, forKey: "zone2")
-               switch2.isHidden = false
-               print("zone 2 manual")
-           case 1:
-               UserDefaults.standard.set(0, forKey: "zone2")
-               switch2.isHidden = true
-               print("zone 2 automatic")
-           default:
-               UserDefaults.standard.set(1, forKey: "zone2")
-               switch2.isHidden = false
-               print("zone 2 manual")
-           }
-       }
-
-       @IBAction func switch2control(_ sender: UISwitch) {
-
-           if switch2.isOn
-           {
-               UserDefaults.standard.set(1, forKey: "zone2On")
-               print("zone 2 manual on")
-           }
-           else
-           {
-               UserDefaults.standard.set(0, forKey: "zone2On")
-            print("zone 2 manual off")
-        }
-    }
-
-    //***********************************
-    //*********ZONE 3*******************
-    @IBOutlet weak var pickerView3: UIPickerView!
-    @IBOutlet weak var control3: UISegmentedControl!
-    @IBOutlet weak var switch3: UISwitch!
-
-    @IBAction func index3(_ sender: UISegmentedControl) {
-        switch control3.selectedSegmentIndex
-        {
-        case 0:
-            UserDefaults.standard.set(1, forKey: "zone3")
-            switch3.isHidden = false
-            print("zone 3 manual")
-        case 1:
-            UserDefaults.standard.set(0, forKey: "zone3")
-            switch3.isHidden = true
-            print("zone 3 automatic")
-        default:
-            UserDefaults.standard.set(1, forKey: "zone3")
-            switch3.isHidden = false
-            print("zone 3 manual")
-        }
-    }
-
-
-    @IBAction func switch3control(_ sender: UISwitch) {
-        if switch3.isOn
-        {
-            UserDefaults.standard.set(1, forKey: "zone3On")
-            print("zone 3 manual on")
-        }
-        else
-        {
-            UserDefaults.standard.set(0, forKey: "zone3On")
-            print("zone 3 manual off")
-        }
-    }
-       //**********************************
-       //***************ZONE 4*************
-
-       @IBOutlet weak var pickerView4: UIPickerView!
-       @IBOutlet weak var control4: UISegmentedControl!
-       @IBOutlet weak var switch4: UISwitch!
-
-       @IBAction func index4(_ sender: UISegmentedControl) {
-           switch control4.selectedSegmentIndex
-           {
-           case 0:
-               UserDefaults.standard.set(1, forKey: "zone4")
-               switch4.isHidden = false
-               print("zone 4 manual")
-           case 1:
-               UserDefaults.standard.set(0, forKey: "zone4")
-               switch4.isHidden = true
-               print("zone 4 automatic")
-           default:
-               UserDefaults.standard.set(1, forKey: "zone4")
-               switch4.isHidden = false
-               print("zone 4 manual")
-           }
-
-       }
-
-       @IBAction func switch4control(_ sender: UISwitch) {
-
-           if switch4.isOn
-           {
-               UserDefaults.standard.set(1, forKey: "zone4On")
-               print("zone 4 manual on")
-           }
-           else
-           {
-               UserDefaults.standard.set(0, forKey: "zone4On")
-               print("zone 4 manual off")
-           }
-       }
-
-       func numberOfComponents(in pickerView: UIPickerView) -> Int {
```

```
-           return 1
-       }
-
-       func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
-           return veg.count
-       }
-
-       func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String? {
-           return veg[row]
-       }
-
-       func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
-           let Veg1 = veg[pickerView1.selectedRow(inComponent: 0)]
-           let Veg2 = veg[pickerView2.selectedRow(inComponent: 0)]
-           let Veg3 = veg[pickerView3.selectedRow(inComponent: 0)]
-           let Veg4 = veg[pickerView4.selectedRow(inComponent: 0)]
-           UserDefaults.standard.set(pickerView1.selectedRow(inComponent: 0), forKey: "chosen1")
-           UserDefaults.standard.set(pickerView2.selectedRow(inComponent: 0), forKey: "chosen2")
-           UserDefaults.standard.set(pickerView3.selectedRow(inComponent: 0), forKey: "chosen3")
-           UserDefaults.standard.set(pickerView4.selectedRow(inComponent: 0), forKey: "chosen4")
-           print(Veg1)
-           print(Veg2)
-           print(Veg3)
-           print(Veg4)
-       }
-
-   }
```

- Fault Detection Code:

```
-   //
-   //  FaultDetectionViewController.swift
-   //  QATra Irrigation System
-   //
-   //  Created by Fatima Al-Janahi on 2/20/20.
-   //  Copyright © 2020 Fatima Al-Janahi. All rights reserved.
-   //
-
-   import UIKit
-
-   class FaultDetectionViewController: UIViewController {
-
-       @IBOutlet weak var green1: UIImageView!
-       @IBOutlet weak var green2: UIImageView!
-       @IBOutlet weak var green3: UIImageView!
-       @IBOutlet weak var green4: UIImageView!
-       @IBOutlet weak var red1: UIImageView!
-       @IBOutlet weak var red2: UIImageView!
-       @IBOutlet weak var red3: UIImageView!
-       @IBOutlet weak var red4: UIImageView!
-
-       override func viewDidLoad() {
-           super.viewDidLoad()
-           UserDefaults.standard.integer(forKey: "fault1")
-           UserDefaults.standard.integer(forKey: "fault2")
-           UserDefaults.standard.integer(forKey: "fault3")
-           UserDefaults.standard.integer(forKey: "fault4")
-
-           if fault1 == 0
-           {
-               green1.isHidden = false
-               red1.isHidden = true
-           }
-           else
-           {
-               green1.isHidden = true
-               red1.isHidden = false
-           }
-
-           if fault2 == 0
-           {
-               green2.isHidden = false
-               red2.isHidden = true
-           }
-           else
-           {
-               green2.isHidden = true
-               red2.isHidden = false
-           }
-
-           if fault3 == 0
-           {
-               green3.isHidden = false
-               red3.isHidden = true
-           }
-           else
-           {
-               green3.isHidden = true
-               red3.isHidden = false
-           }
-
-           if fault4 == 0
-           {
-               green4.isHidden = false
-               red4.isHidden = true
-           }
-           else
-           {
-               green4.isHidden = true
-               red4.isHidden = false
-           }
-       }
-
-       let fault1 = UserDefaults.standard.integer(forKey: "fault1")
-       let fault2 = UserDefaults.standard.integer(forKey: "fault2")
-       let fault3 = UserDefaults.standard.integer(forKey: "fault3")
-       let fault4 = UserDefaults.standard.integer(forKey: "fault4")
-
-
-   }
```

- Moisture Level Code

```
- //
- //  MoistViewController.swift
- //  QATra Irrigation System
- //
- //  Created by Fatima Al-Janahi on 2/20/20.
- //  Copyright © 2020 Fatima Al-Janahi. All rights reserved.
- //
- 
- import UIKit
- 
- class MoistViewController: UIViewController {
- 
-         override func viewDidLoad() {
-         super.viewDidLoad()
-         UserDefaults.standard.integer(forKey: "Moist1")
-         UserDefaults.standard.integer(forKey: "Moist2")
-         UserDefaults.standard.integer(forKey: "Moist3")
-         UserDefaults.standard.integer(forKey: "Moist4")
- 
-     }
- 
-     let Moist1 = UserDefaults.standard.integer(forKey: "Moist1")
-     let Moist2 = UserDefaults.standard.integer(forKey: "Moist2")
-     let Moist3 = UserDefaults.standard.integer(forKey: "Moist3")
-     let Moist4 = UserDefaults.standard.integer(forKey: "Moist4")
- }
```