IoT SMART GARDENING

1.1. IoT DESIGN METHODOLOGY

Step 1 : Purpose and Requirement Specification

- **Purpose** – The purpose of IoT gardening is to maintain the well-being of a garden using the power of the Internet of Things (IoT). With the help of present tools and software, planter is integrated with sensors that monitors the real-time status of the plants.

- **Behaviour** – The IoT smart gardening has both auto and manual modes.

- **System Management Requirements -** A Raspberry Pi is used to relay useful information of the garden, such as luminosity, humidity and the moisture content in the soil from various sensors into a cloud database. Once the information is in the cloud, it can be accessed from anywhere using a Smartphone app.

- **Data Analysis Requirements** – The system performs local Analysis of data, based on that it will decide whether to turn on street lights or to turn on water pumps.

Step 2 : Process Specification

In this step the use cases of the IoT system are formally described based on and derived from the purpose and requirement specification.

Figure 2.1 shows the process diagram for street lights and water pumps. The process diagram shows two modes of the system – auto and manual. When auto mode is chosen the system monitors the light level. If the light level is low, the system changes the light state to “on”. Where as if the light level is high, the system change the state of light to “off”. But in case of manual mode user will change the state of light.

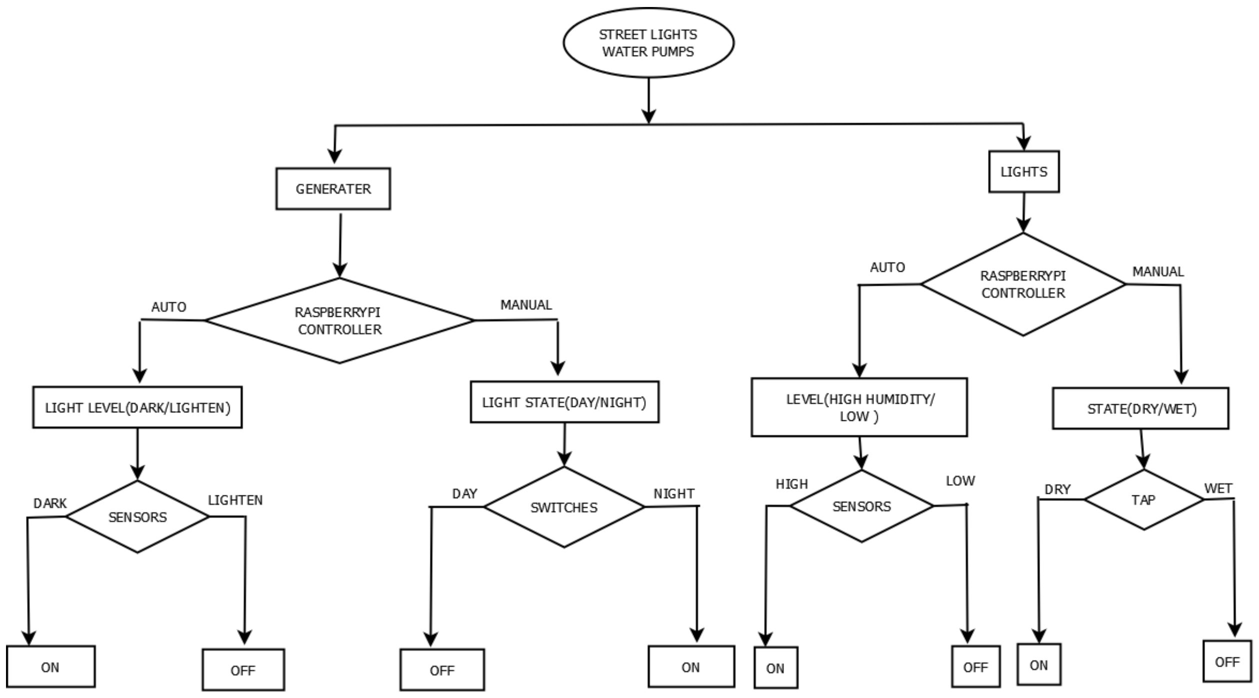


Figure 2.1: Process Specification for Street Lights and Water

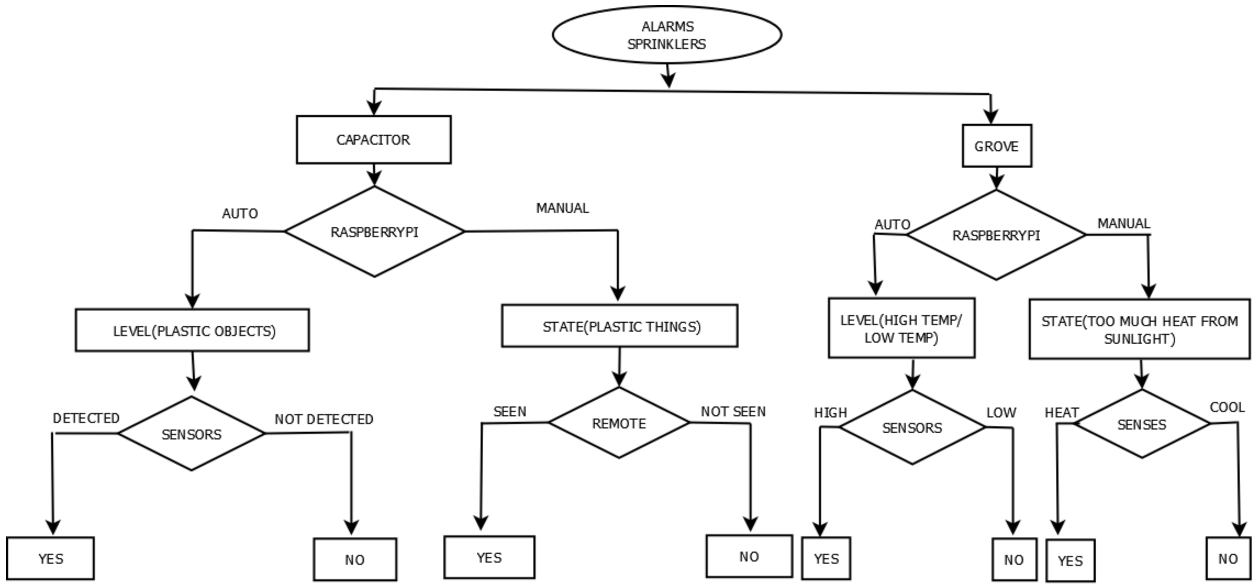


Figure 2.2: Process Specification for Alarms and Sprinklers

Step 3 : Domain Model Specification

- **Physical Entity** – The Physical Entity is a discrete and identifiable entity in the physical environment (e.g., plants, soil, light, etc.). The IoT system provides information about the physical entity by using sensors and performs actuation upon the Physical Entity like switching on a light.

- **Virtual Entity** - Virtual entity is the representation of physical entity in the digital world. For each physical entity, there is a virtual entity in domain model. In the IoT smart gardening there is one virtual entity for Appliance and one more virtual entity for lights.

- **Device** – Device provide a medium for communication between physical entity and virtual entity. Devices are used to gather information about physical entity and perform actuations upon physical entities. Device used in smart gardening is minicomputer.

- **Resources** – Resource are soft component which can be either “on-device” or “network resources”. On-device Resource used in smart gardening is operating system that runs on the single board mini computer.

- **Service** – The services provided by smart gardening system are as follows,
    - ✓ Drip irrigation system
    - ✓ App controlled water system
    - ✓ Automatic watering schedules
    - ✓ Database of the garden's health status
    - ✓ Real-time feedback of the garden's various sensors

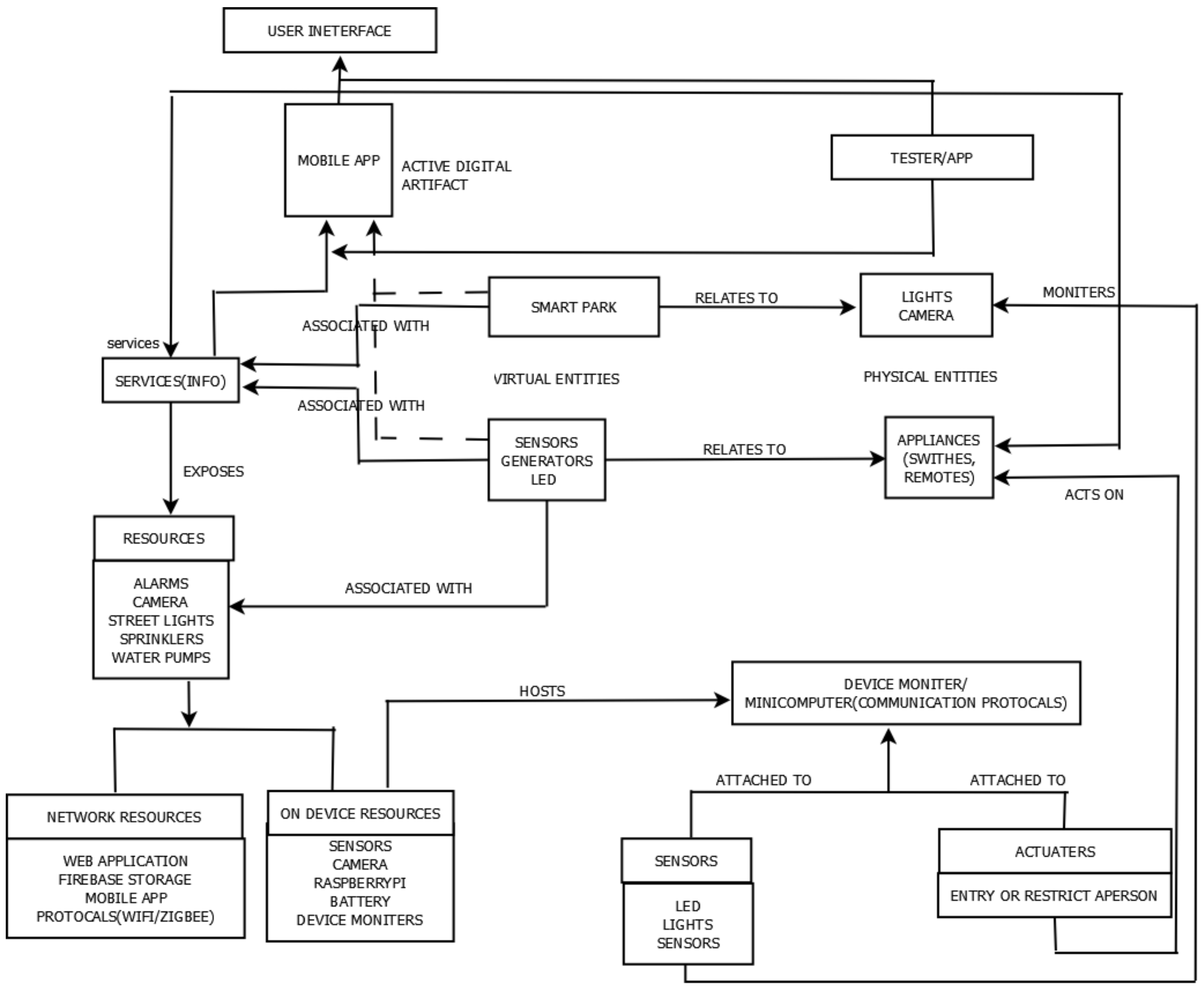


Figure 3: Domain model of the IoT Smart Gardening

Step 4 : Information model Specification

Information model defines the structure of all the information in the IoT system. In smart gardening application there are two Virtual Entities – a virtual entity for Appliance with the attributes of Streetlights, plastic object detectors, temperature sensors and a virtual entity for water pumps.

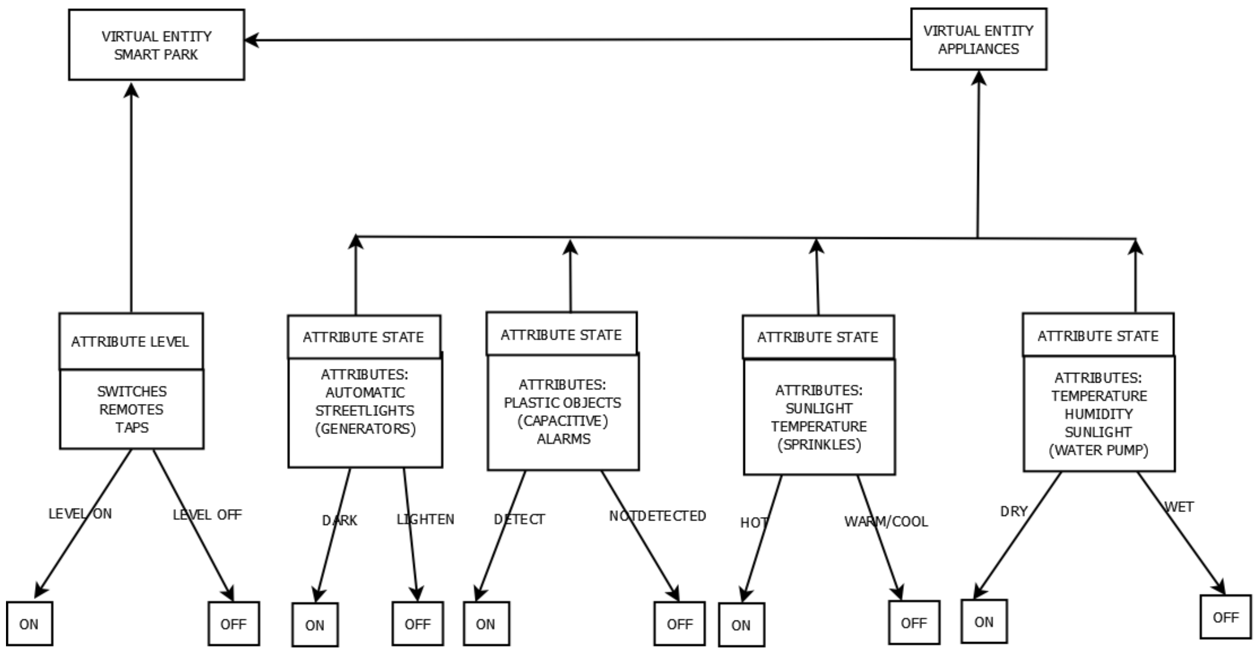

VIRTUAL ENTITY SMART PARK
VIRTUAL ENTITY APPLIANCES
ATTRIBUTE LEVEL
SWITCHES REMOTES TAPS
ATTRIBUTE STATE
ATTRIBUTES: AUTOMATIC STREETLIGHTS (GENERATORS)
ATTRIBUTE STATE
ATTRIBUTES: PLASTIC OBJECTS (CAPACITIVE) ALARMS
ATTRIBUTE STATE
ATTRIBUTES: SUNLIGHT TEMPERATURE (SPRINKLES)
ATTRIBUTE STATE
ATTRIBUTES: TEMPERATURE HUMIDITY SUNLIGHT (WATER PUMP)
LEVEL ON
LEVEL OFF
DARK
LIGHTEN
DETECT
NOTDETECTED
HOT
WARM/COOL
DRY
WET
ON
OFF
ON
OFF
ON
OFF
ON
OFF
ON
OFF

Figure 4: Information model for IoT Smart Gardening

Step 5 : Service Specifications

The fifth step in the IoT design methodology is to define the service specification. Service specification define the services in the IoT system types, service inputs/output, service endpoints, service preconditions and service effects.

Figure 5.1 shows an example of deriving the services from the process specification and information model for IoT Smart Gardening system. From the process specification and information model the states and attributes can be identified for which each state and attribute defines a service. The mode service sets mode to auto or manual or retrieves the current mode. The state service sets the light appliance state to on/off or retrieves the current light state. The controller service monitors the light level in auto mode and switches the light on/off and updates the status in status database.

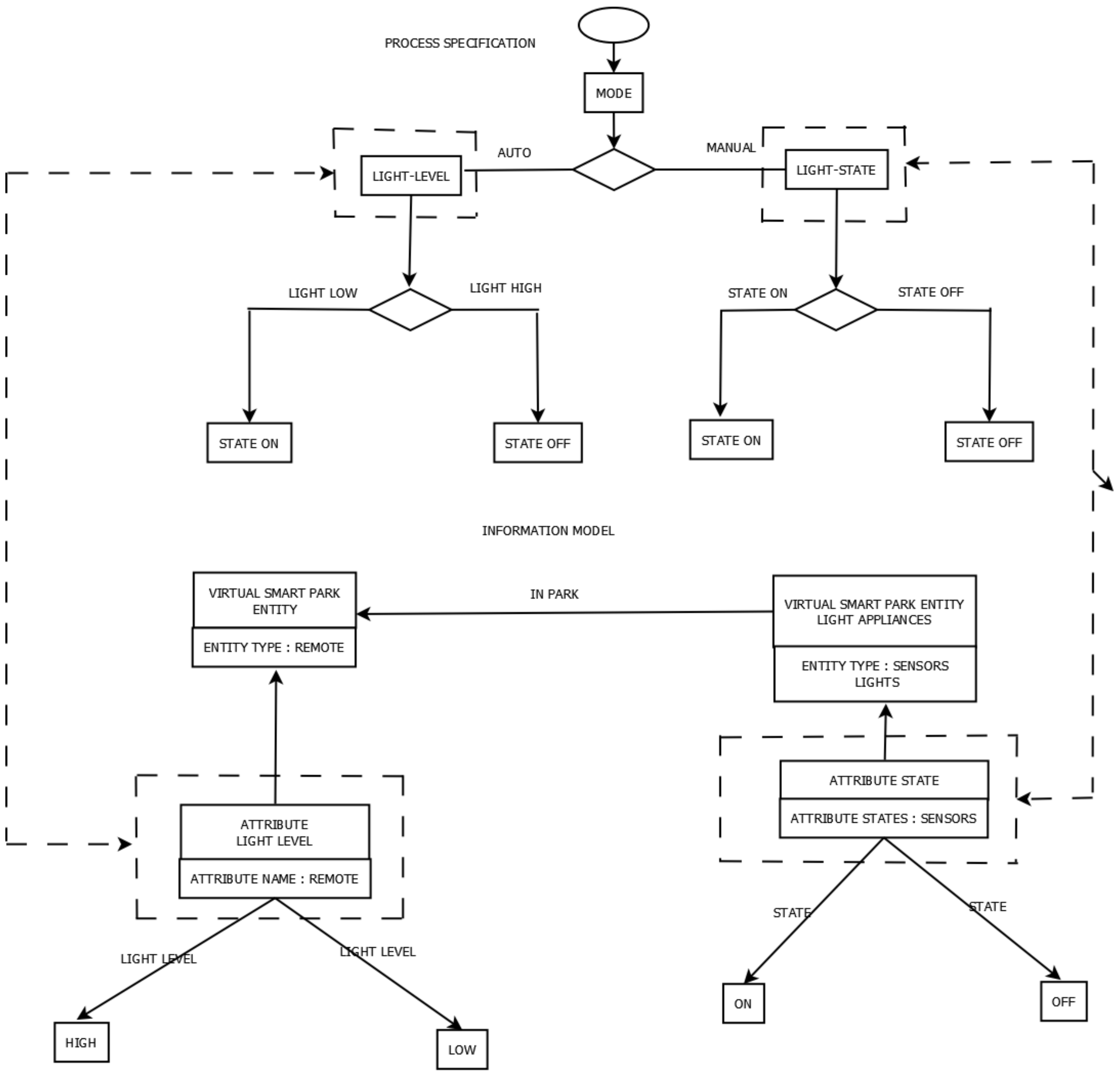


Figure 5.1: Deriving Services from process specification and information model for IoT smart Gardening

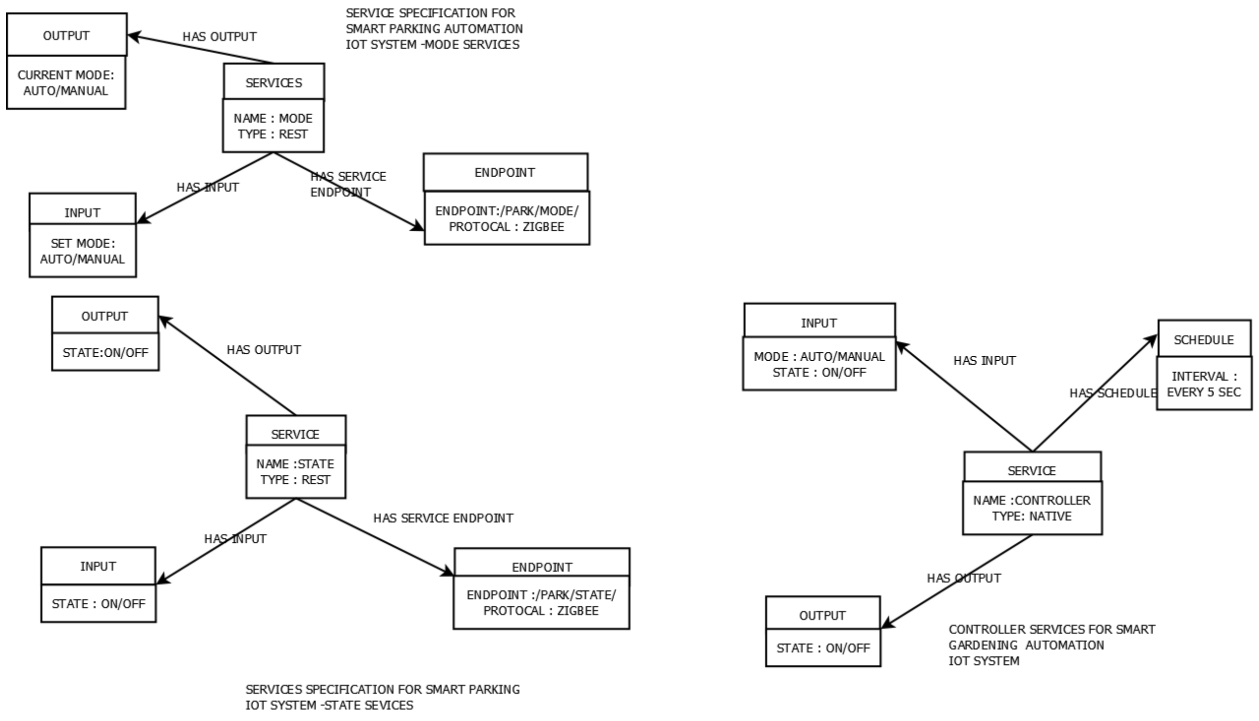


Figure 5.2: Service Specification for IoT Smart Gardening

Virtual entity is the representation of physical entity in the digital world. For each physical entity, there is a virtual entity in domain model. In the IoT smart gardening there is one virtual entity for Appliance and one more virtual entity for lights

The mode service sets mode to auto or manual or retrieves the current mode. The state service sets the light appliance state to on/off or retrieves the current light state. The controller service monitors the light level in auto mode and switches the light on/off and updates the status in status database.

Step 6 : IoT Level Specification

The sixth step in the IoT design methodology is to define the IoT level for the system. The services provided by the system of numerous elements like sensors, protocols, actuators, cloud services, and layers the services in the sence such a number is chosen to include these various types of components into a complex network.

Services by the node device which have unique identities and states can perform remote sensing, actuating and monitoring capabilities of on and off from remotes place.Communication established between things and cloud based server over the Internet by various IOT protocols and the alert can be disabled by the user from the mobile application.

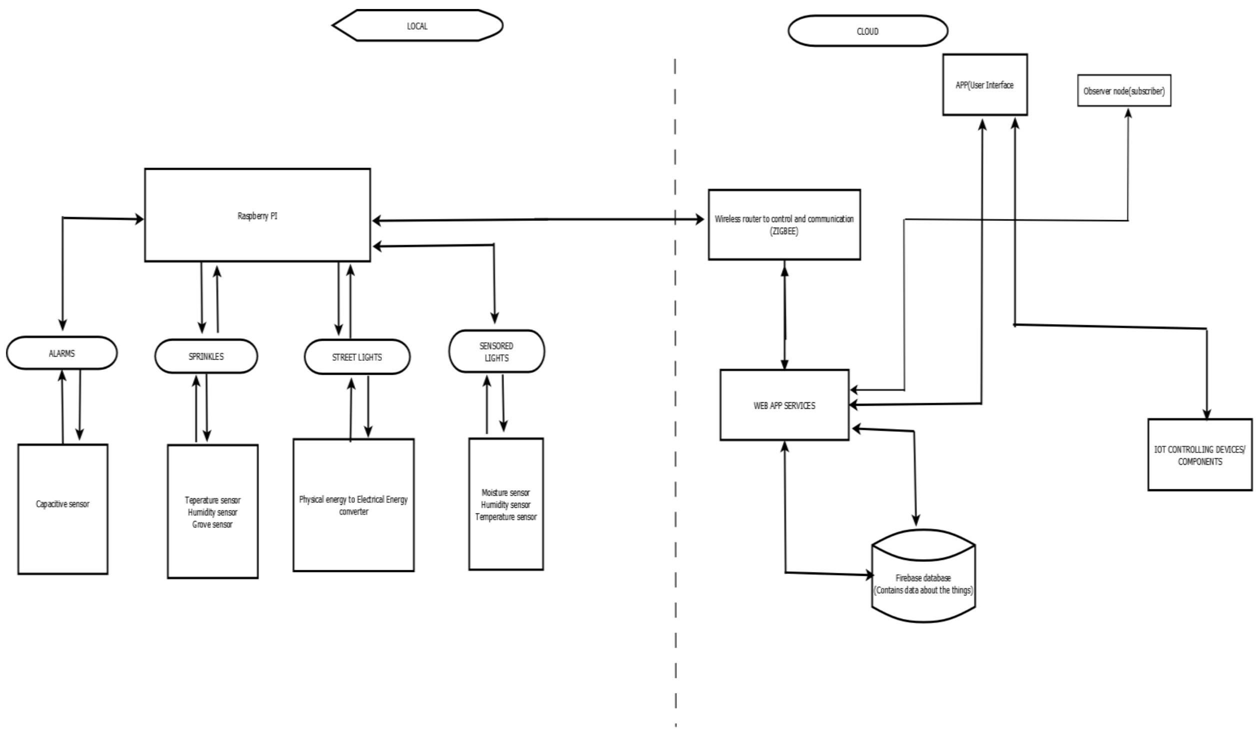

LOCAL
CLOUD
APP(User Interface
Observer node(subscriber)
Raspberry PI
Wireless router to control and communication (ZIGBEE)
ALARMS
SPRINKLES
STREET LIGHTS
SENSORED LIGHTS
WEB APP SERVICES
IOT CONTROLLING DEVICES/ COMPONENTS
Capacitive sensor
Teperature sensor Humidity sensor Grove sensor
Physical energy to Electrical Energy converter
Moisture sensor Humidity sensor Temperature sensor
Firebase database (Contains data about the things)

Figure 6: Deployment design of IoT Smart Gardening

Step 7 : Functional View Specification

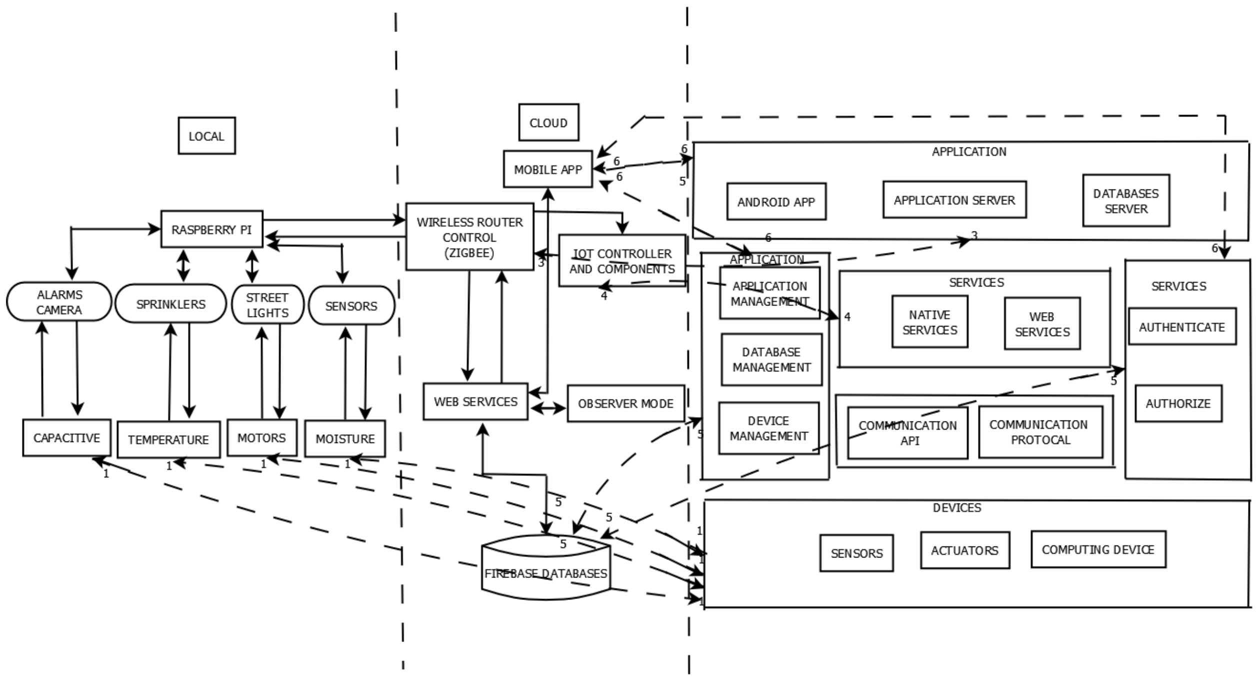


Figure 7: Mapping deployment level to functional groups for IoT Smart Gardening

arious functional groups. Figure 7 shows the mapping of deployment level to functional groups for smart gardening.

IoT device maps to the device functional groups like sensors, actuators and computing devices and the management functional groups. Resources map to the device functional group and communication functional group. Web services maps to services functional group. Database maps to Management functional group where as Application maps to application functional group and security functional group.

Step 8 : Operational View Specification

The eighth step is to define the operational view specifications. In this step various options pertaining to the IoT system deployment and operation are defined such as service hosting options, storage options, device options and application hosting options.

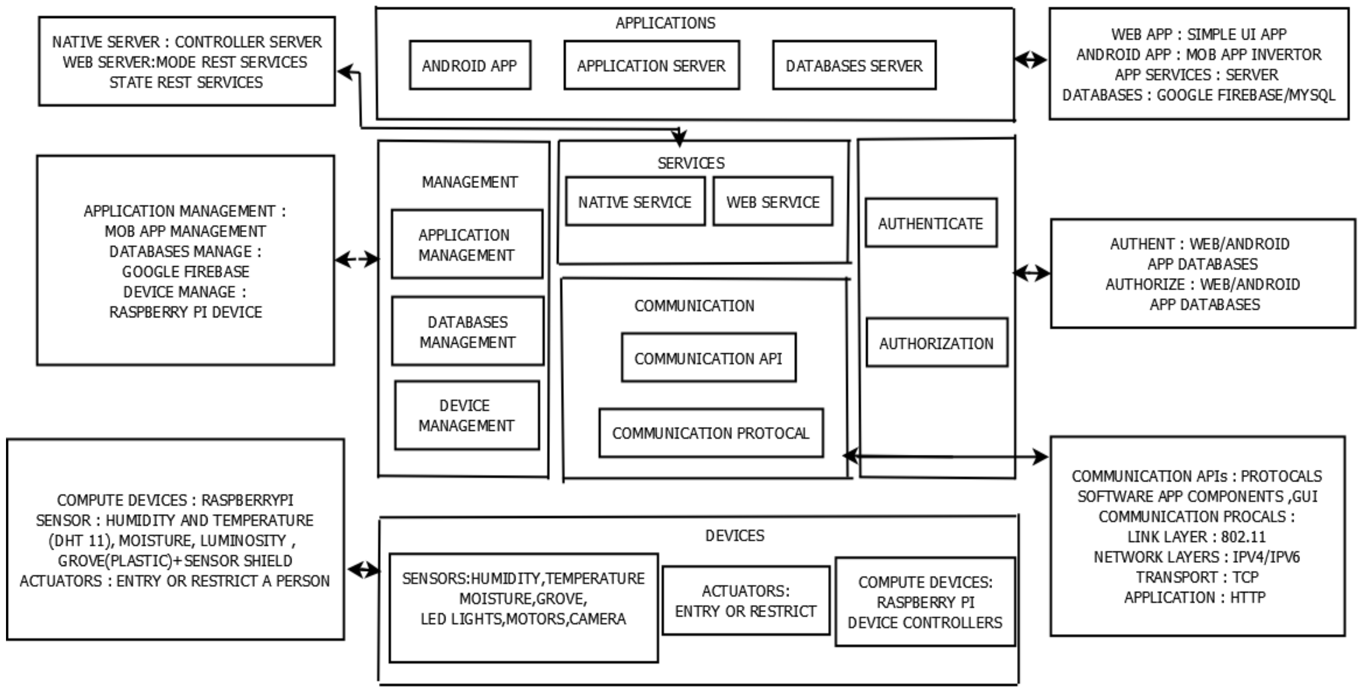


Figure 8: Mapping functional groups to operational view for IoT Smart Gardening

1.2. APPLICATIONS OF IoT SMART GARDENING

- Real-time feedback of the garden's various sensors
- Database of the garden's health status
- Drip irrigation system
- App controlled water system
- Automatic watering schedules
- Automatic street lights

Step 9 : Devices and Component Integration

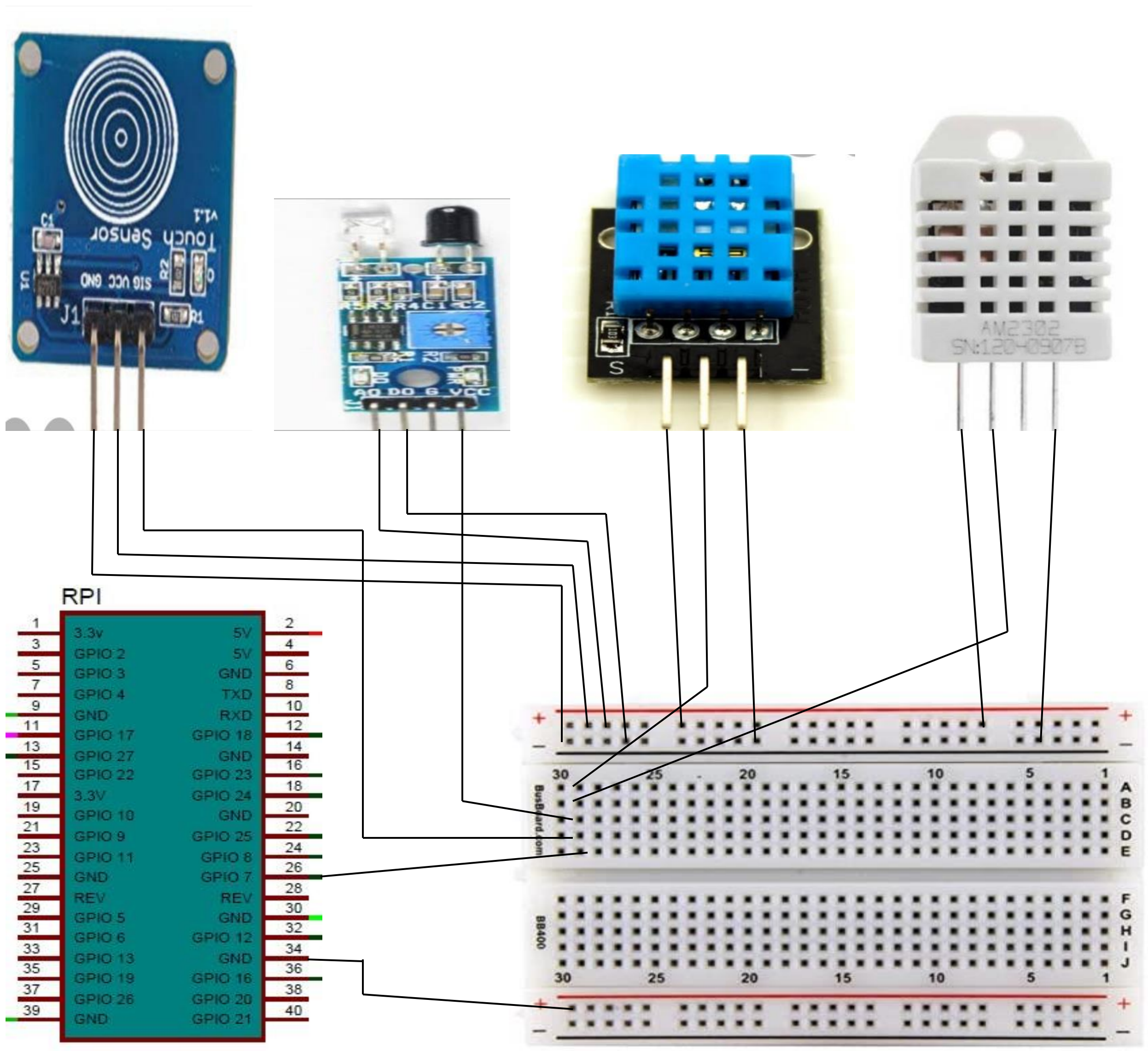


Figure 9: Pin Diagram

RASPBERRY PI

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse.

It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.

An SD card inserted into the slot on the board acts as the hard drive for the Raspberry Pi. It is powered by USB and the video output can be hooked up to a traditional RCA TV set, a more modern monitor, or even a TV using the HDMI port.

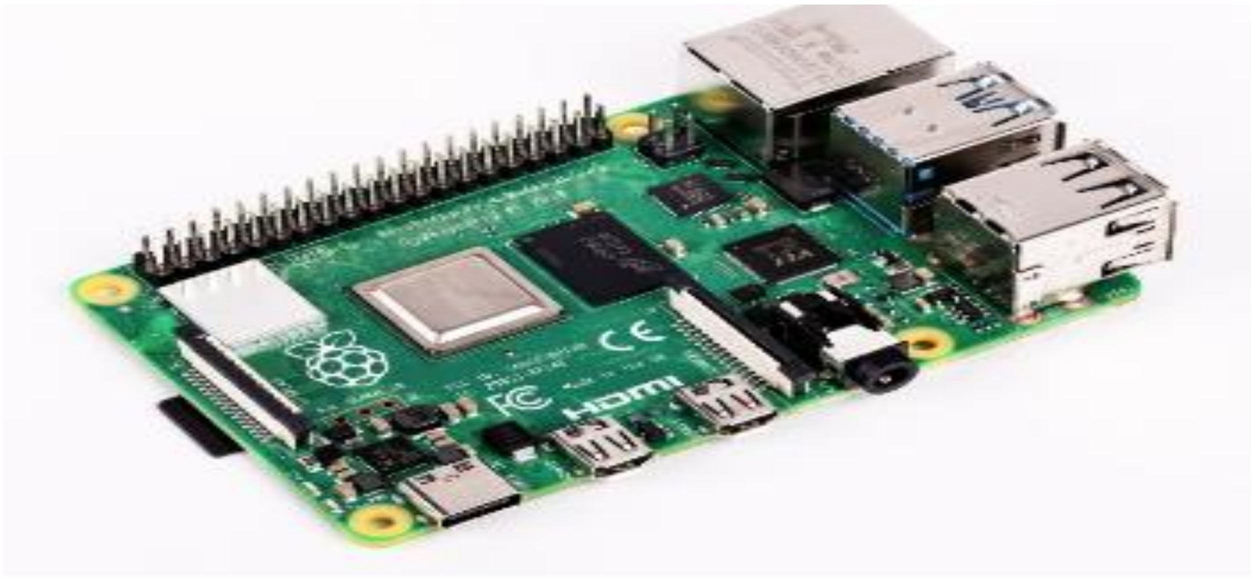


Figure 10: Rasberry Pi

The devices which converts the electrical signals into digital signals are known as sensors. The different types of sensors incorporated in this system are listed below.

 Humidity sensor – used to measure the humidity content of the soil.

 Temperature sensor – used to measure the temperature of the soil.

 IR sensor – used to measure the darkness in the park.

 capacitive sensor – used to identify the grievance in the park.

DHT11 (Teperature Sensor)

The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed).

Its fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old

DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them.

Change in the capacitance value occurs with the change in humidity levels. The IC measure, process this changed resistance values and change them into digital form.

Low cost,3 to 5V power and I/O,2.5mA max current use during conversion (while requesting data),Good for 20-80% humidity readings with 5% accuracy,Good for 0-50°C temperature readings ±2°C accuracy,No more than 1 Hz sampling rate (once every second),Body size 15.5mm x 12mm x 5.5mm.

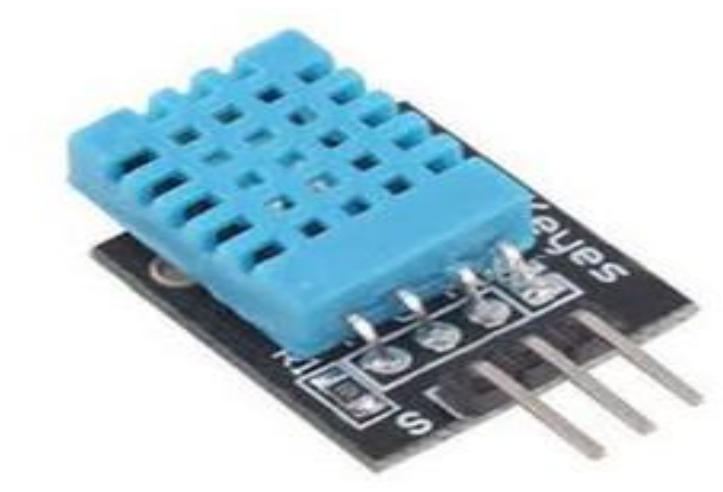


Figure 11: DHT11(Temperature sensor)

DHT22(Humidity Sensor)

Compared to the DHT22, this sensor is less precise, less accurate and works in a smaller range of temperature/humidity, but its smaller and less expensive

The DHT22 module contains a capacitive humidity sensor, DS18B20 temperature sensor and an unnamed 8-bit microcontroller. According to its datasheet, it is capable of detecting 0 to 100% relative humidity and temperatures from -40 to 125 degree celsius.

The resolution for both humidity and temperature is 0.1 (RH and degree celsius) while the accuracy is +/ 2 for humidity and +/ 0.3 for temperature.Compared to the DHT11, the DHT22 has a wider temperature range and better accuracy. Physically, the housing of this sensor is larger than the DHT11

The DHT22 is also called AM2302. Adafruit says that AM2302 is just the wired version of DHT22. Seeed also calls their DHT22 grove board as AM2302 temperature and humidity sensor pro with emphasis on "pro" since it is superio to the DHT11.

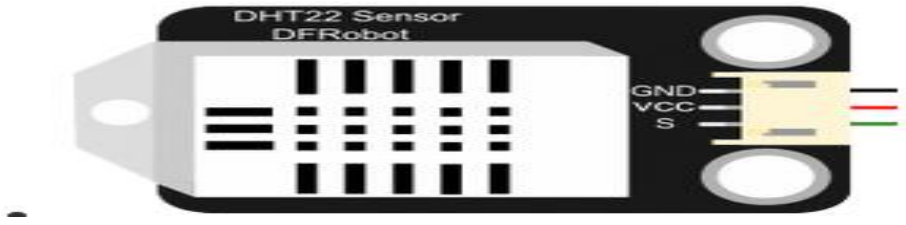


Figure 12: DHT22(HumiditySensor)

I discovered that AM2302 is really the DHT22 with an included pull-up resistor on its DOUT pin. Anyway, if you own the same module as I do, note that there is already a pull-up resistor included in the module. The same is true for Seeed's DHT22 board.

To begin asking the DHT22 for humidity and temperature data, the microcontroller must perform the start signal, which is a logic 0 (low) for at least 500 microseconds followed by a logic 1 (high).

The DHT22 must then respond by pulling the bus low for 80 microseconds then pulling the bus high for another 80 microseconds before pulling it low again

Figure 13: IR Sensors

IR Sensor

A Light Sensor generates an output signal indicating the intensity of light by measuring the radiant energy that exists in a very narrow range of frequencies basically called “light” and which ranges in frequency from “Infra-red” to “Visible” up to “Ultraviolet” light spectrum. The light sensor is a passive devices that convert this “light energy” whether visible or in the infra-red parts of the spectrum into an electrical signal output. Light sensors are more commonly known as “Photoelectric Devices” or “Photo Sensors” because the convert light energy (photons) into electricity (electrons).

Darkness Detector circuits like this can be used in applications where we can automatically turn on lights when it becomes dark.

In addition to the LDR, we have also used the good old 555 Timer IC in Astable Mode to generate the required square wave. There are some passive components like capacitor and resistors.

A simple dark detector or darkness detector is designed in this project. The project is implemented using very simple components like 555 and LDR (few passive components as well).

The working of the project is explained here. First, we will start with the 555 Timer. It is configured in Astable mode but the RESET pin is controlled by the LDR and Resistor network. When there is ample light around the LDR, its resistance becomes very low.

In our lab setup, it came down to around 2 KΩ. In this condition, the voltage divider formed by the 1 MΩ resistor and the LDR will produce almost 0V at its output.

As this is given to the RESET pin of the 555 timer IC, the 555 Timer IC is Reset. As a result, you won’t get any output at the output pin. When we block the LDR with an obstacle or hand, the light falling on it will decrease.

The resistance of the LDR will increase and in our case (lab setup with studio lighting) the resistance increased to around 120 KΩ. This will pull up the reset pin and the A stable Mode will be activated.

Figure 14: Capacitive Sensor

CAPACITIVE SENSOR

A proximity sensor is a sensor able to detect the presence of nearby objects without any physical contact. ... The inductive sensors detect the metal target whereas, the photoelectric and capacitive sensors detect the plastic and organic targets

Generally speaking, a sensor is a device that is able to detect changes in an environment. By itself, a sensor is useless, but when we use it in an electronic system, it plays a key role. A sensor is able to measure a physical phenomenon (like temperature, pressure, and so on) and transform it into an electric signa

An IoT system consists of sensors/devices which “talk” to the cloud through some kind of connectivity. Once the data gets to the cloud, software processes it and then might decide to perform an action, such as sending an alert or automatically adjusting the sensors/devices without the need for the user.

WATER PUMP CONTROLLER

This controller is suitable for any type of motor - Single or Three Phase.Switches ON the pump when the water in the overhead tank goes below the pre-decided minimum level.

Switches OFF the pump when the water level in the overhead tank reaches the maximum level therefore prevents overflow. Shall again switch ON the pump when there is sufficient water in the underground tank.Therefore no need to switch ON or switch OFF the pump manually.

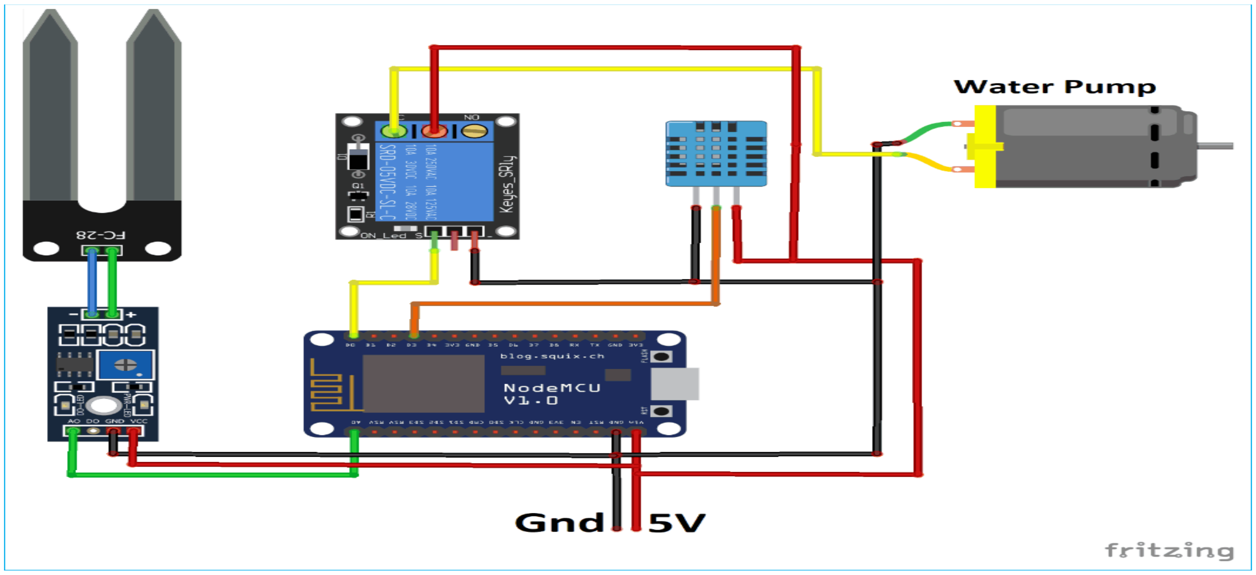


Figure 15: Water pump is connected to sensors which pours water on plants when humidity is high.

STREETLIGHTS

A Light Sensor generates an output signal indicating the intensity of light by measuring the radiant energy that exists in a very narrow range of frequencies basically called “light” and which ranges in frequency from “Infra-red” to “Visible” up to “Ultraviolet” light spectrum. The light sensor is a passive devices that convert this “light energy” whether visible or in the infra-red parts of the spectrum into an electrical signal output. Light sensors are more commonly known as “Photoelectric Devices” or “Photo Sensors” because the convert light energy (photons) into electricity (electrons)

Figure 16: Streetlights will turn ON during night with the help sensor connectivity

ALARM

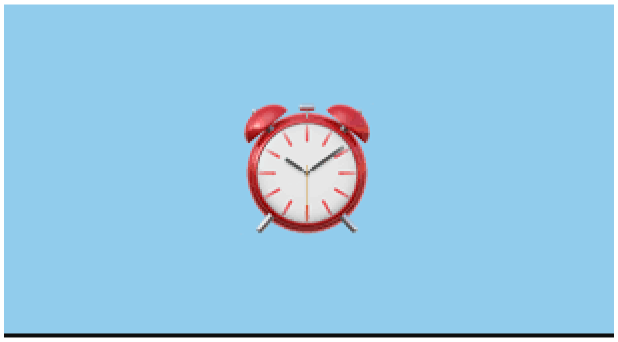


Figure 17: Alarms will turn ON when there is garbage on Park

Step 10 : Application development

The apps can provide an overview of all measurements of the sensors. Users can receive an overview over their smart devices and status, and can deviate at any time from present configurations, taking control themselves as and when required. The sensors are an indicator of soil humidity, temperature and light intensity, and it transfers all information to the web application. The actuators ensure optimum water management, temperature, light intensity, and soil nutrients in the garden. This system can diagnose the garden’s condition and provide recommendations to users in terms of interface devices and controllers. The recommendation is made based on local data collected from the specific situation of the user and internet and cloud database of plants and gardening information. For monitoring the gardening activity, a PostgreSql server can be used with the web server and web client, the data is stored in the database. The users can control this prototype by using a web application where they can see the plant via monitoring menu which is a real-time communication between application and sensors .

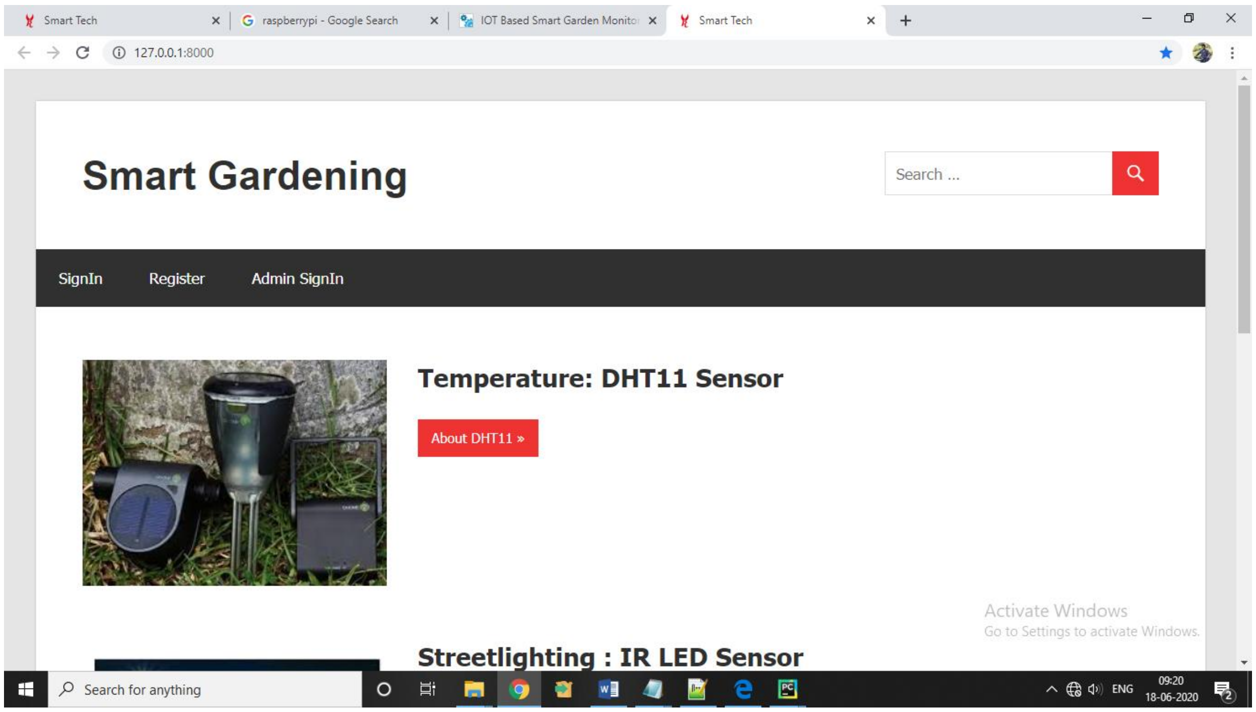


Figure 18: Smart Gardening Iot .

Admin Module of IOT Smart Gardening where he can update information of sensors and he can view it :

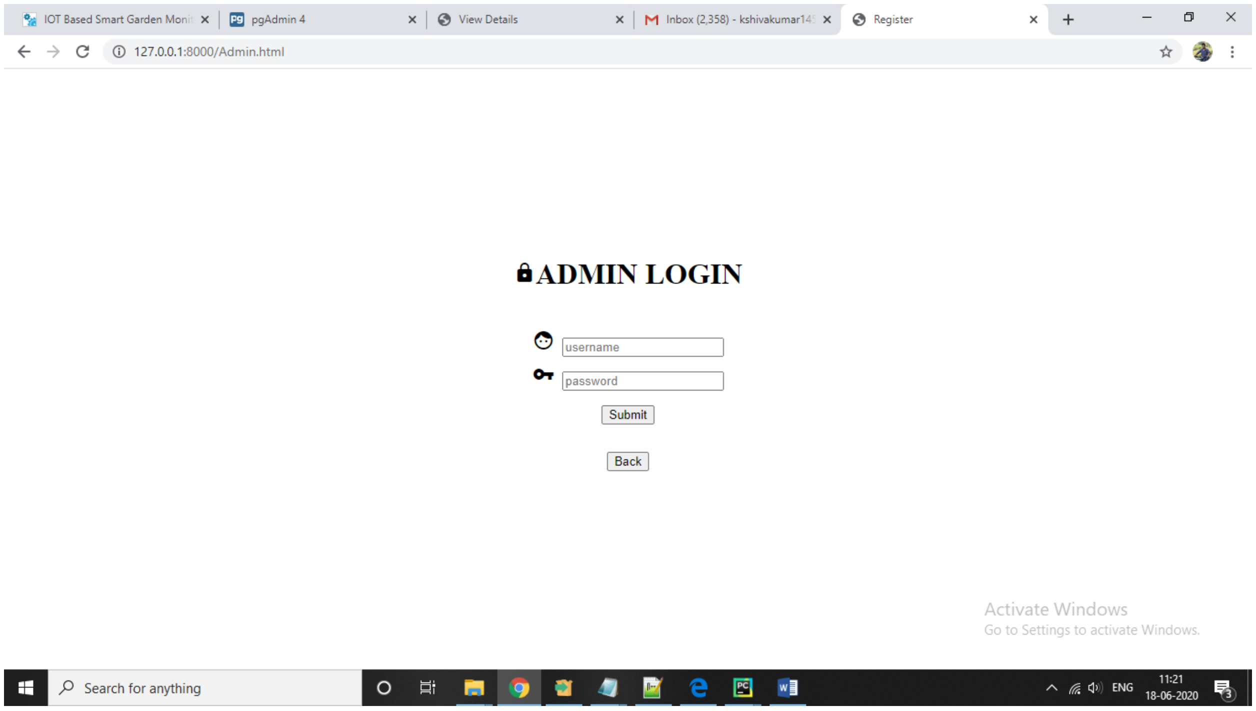


Figure 19: Admin Login Page

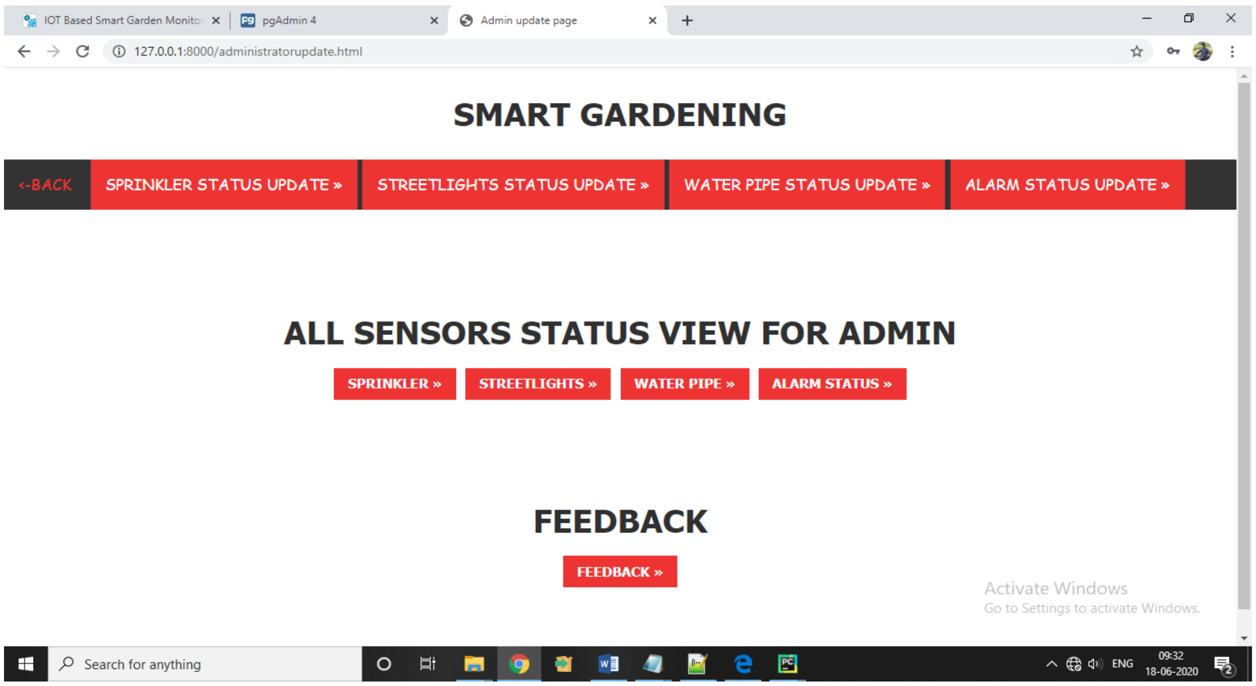

IOT Based Smart Garden Monito
pgAdmin 4
Admin update page
127.0.0.1:8000/administratorupdate.html
SMART GARDENING
<-BACK
SPRINKLER STATUS UPDATE »
STREETLIGHTS STATUS UPDATE »
WATER PIPE STATUS UPDATE »
ALARM STATUS UPDATE »
ALL SENSORS STATUS VIEW FOR ADMIN
SPRINKLER »
STREETLIGHTS »
WATER PIPE »
ALARM STATUS »
FEEDBACK
FEEDBACK »
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG
09:32
18-06-2020

Figure 20: Admin can access a Smart Gardening Iot website and he can update information of sensors and can view it.

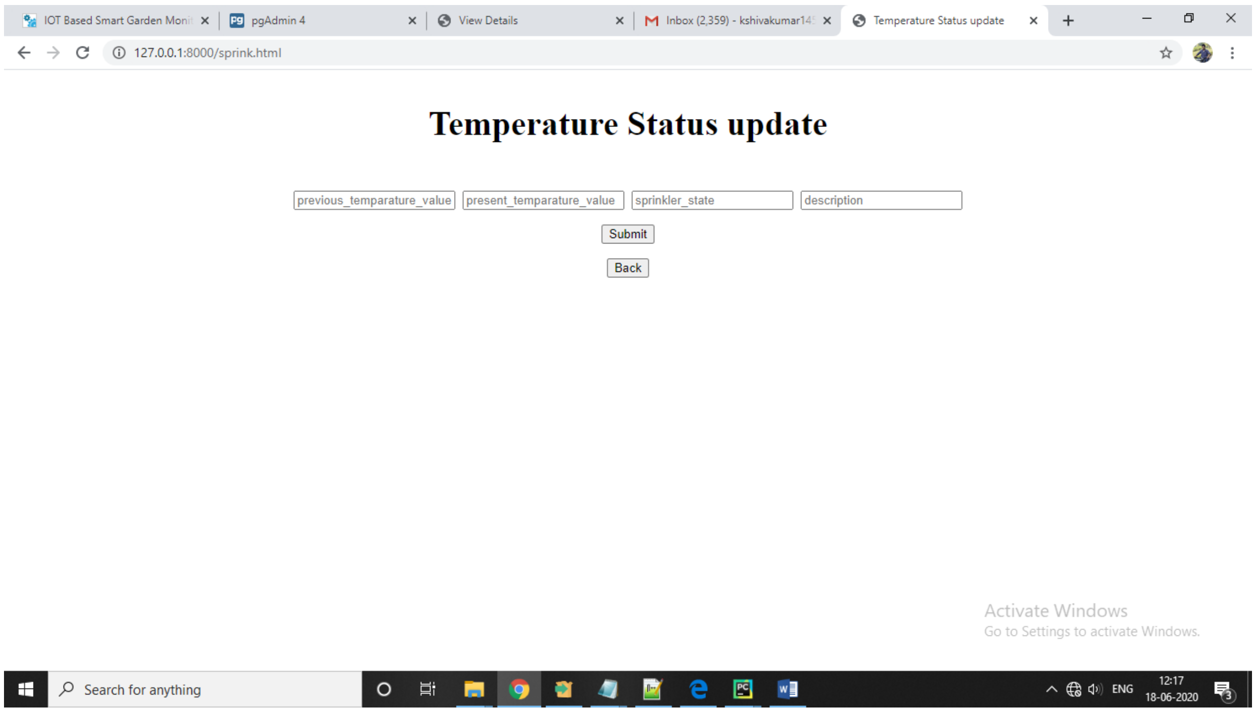

IOT Based Smart Garden Moni
pgAdmin 4
View Details
Inbox (2,359) - kshivakumar14
Temperature Status update
127.0.0.1:8000/sprink.html
Temperature Status update
previous_temparature_value
present_temparature_value
sprinkler_state
description
Submit
Back
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG
12:17
18-06-2020

Figure 21: Admin can update temperature status here

User Module of IOT Smart Gardening where user can view the sensors status and update the complaints:

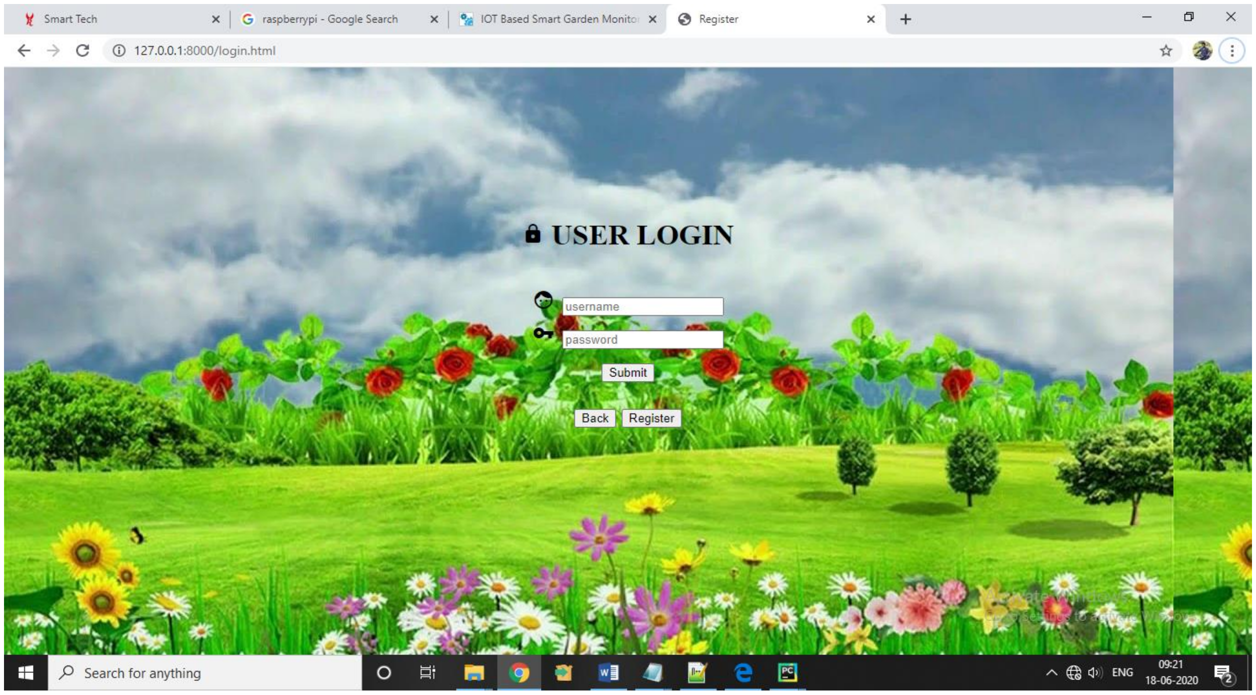


Figure 22: Login page of Smart Gardening Iot website

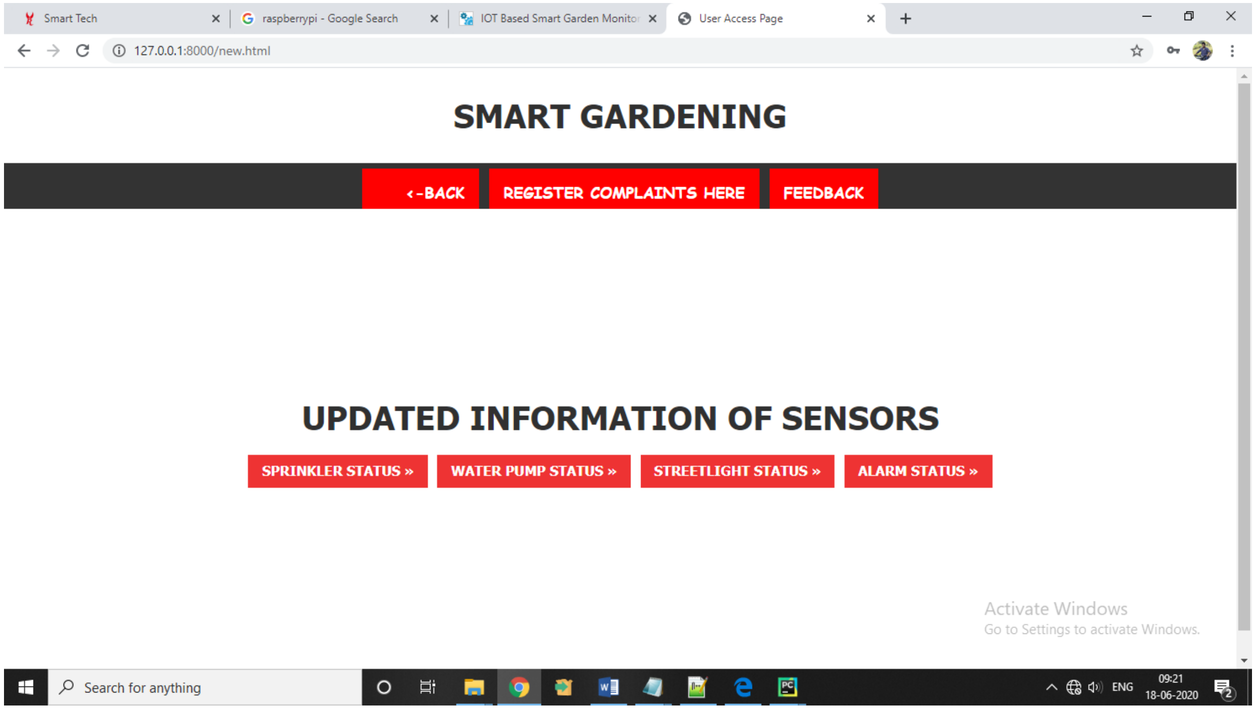


Figure 23: User view of Smart Gardening Iot website and user can view all updated information of sensors.

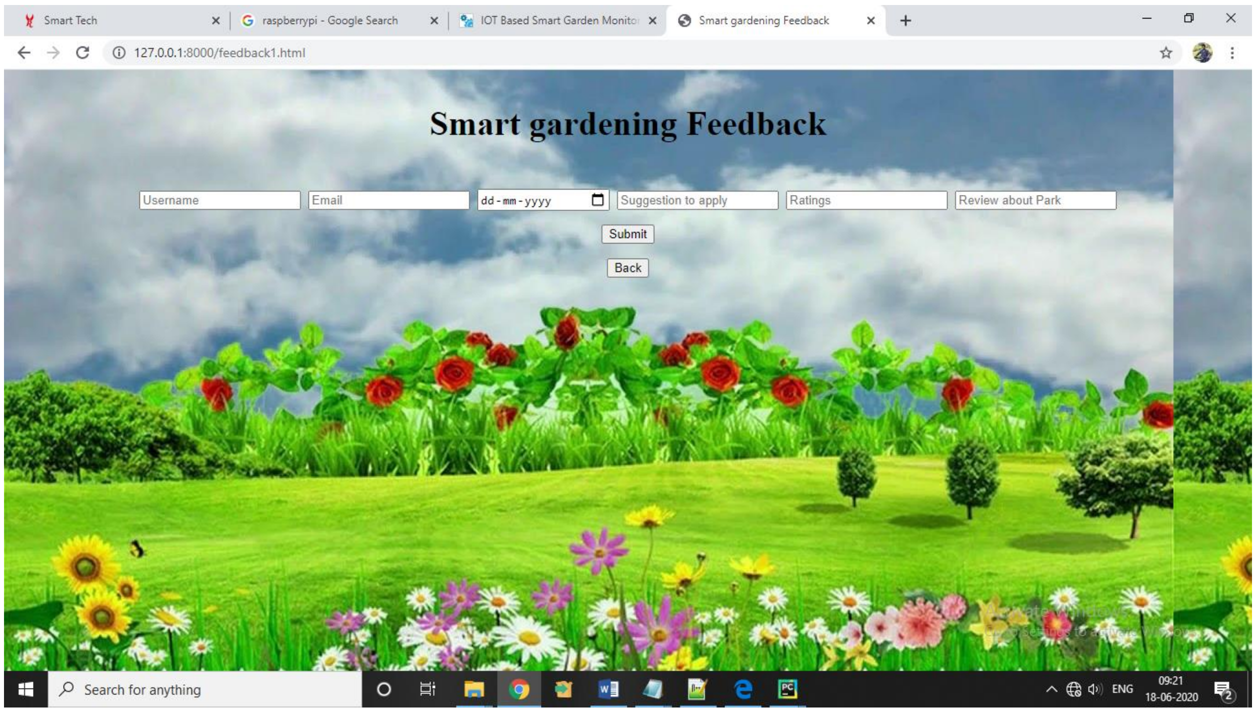
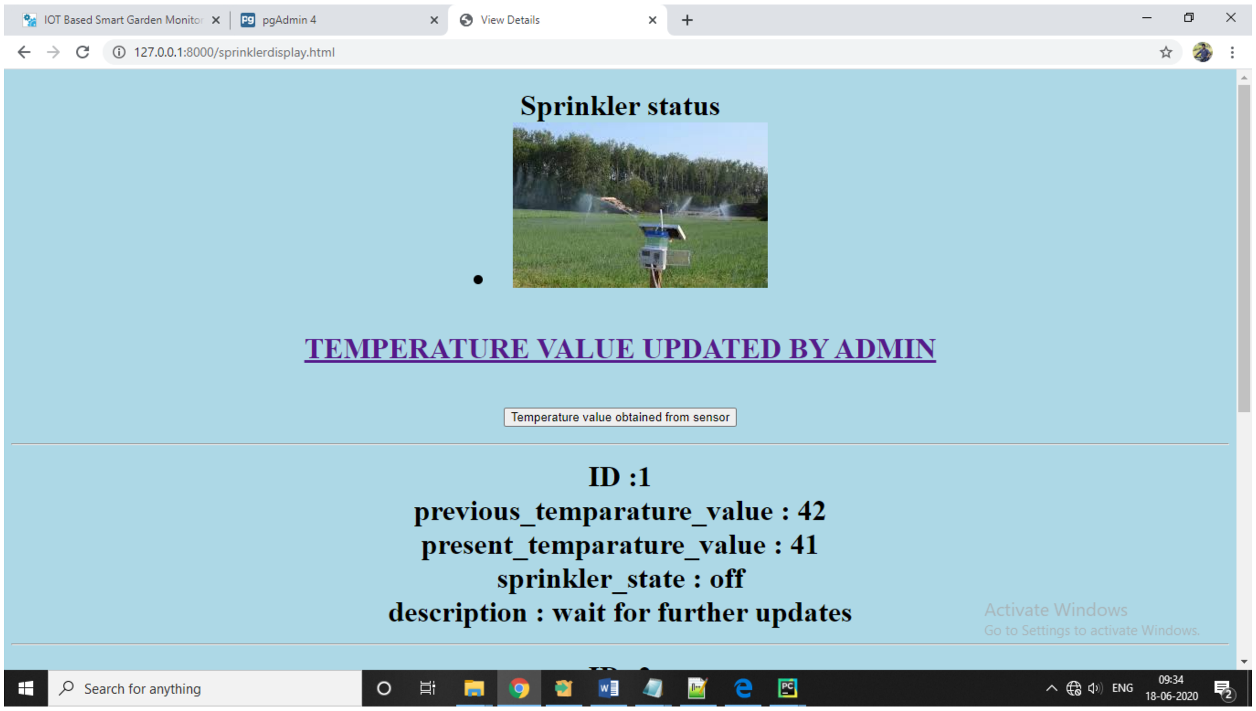


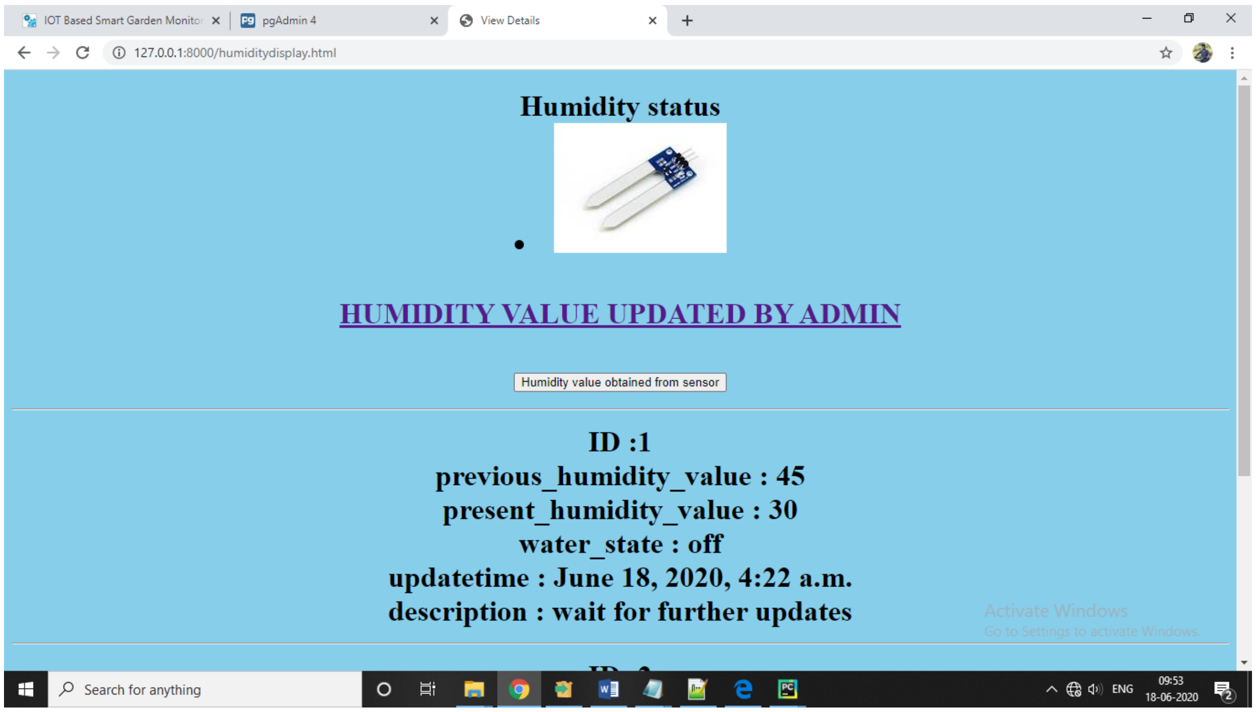
Figure 24: User can provide a feedback and view all feedback of users



IOT Based Smart Garden Monitor
pgAdmin 4
View Details
127.0.0.1:8000/sprinklerdisplay.html
Sprinkler status
TEMPERATURE VALUE UPDATED BY ADMIN
Temperature value obtained from sensor
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG
09:34
18-06-2020

ID :1
previous_temparature_value : 42
present_temparature_value : 41
sprinkler_state : off
description : wait for further updates

Figure 25: user view of sensor status and sprinkler status


IOT Based Smart Garden Monito
pgAdmin 4
View Details
127.0.0.1:8000/humiditydisplay.html
Search for anything
Activate Windows
Go to Settings to activate Windows.
ENG
09:53
18-06-2020

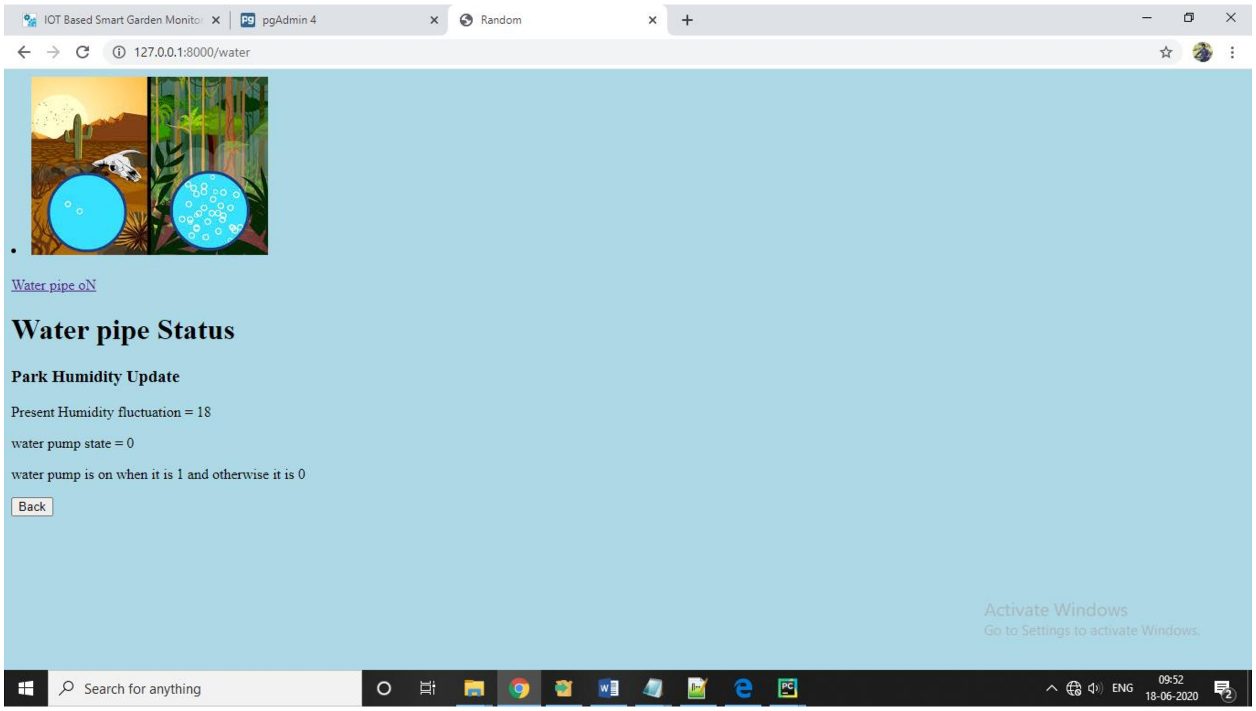
Humidity status

HUMIDITY VALUE UPDATED BY ADMIN

Humidity value obtained from sensor

ID :1
previous_humidity_value : 45
present_humidity_value : 30
water_state : off
updatetime : June 18, 2020, 4:22 a.m.
description : wait for further updates

Figure 26 :Humidity status and water pump status view for user



Water pipe oN

Water pipe Status

Park Humidity Update

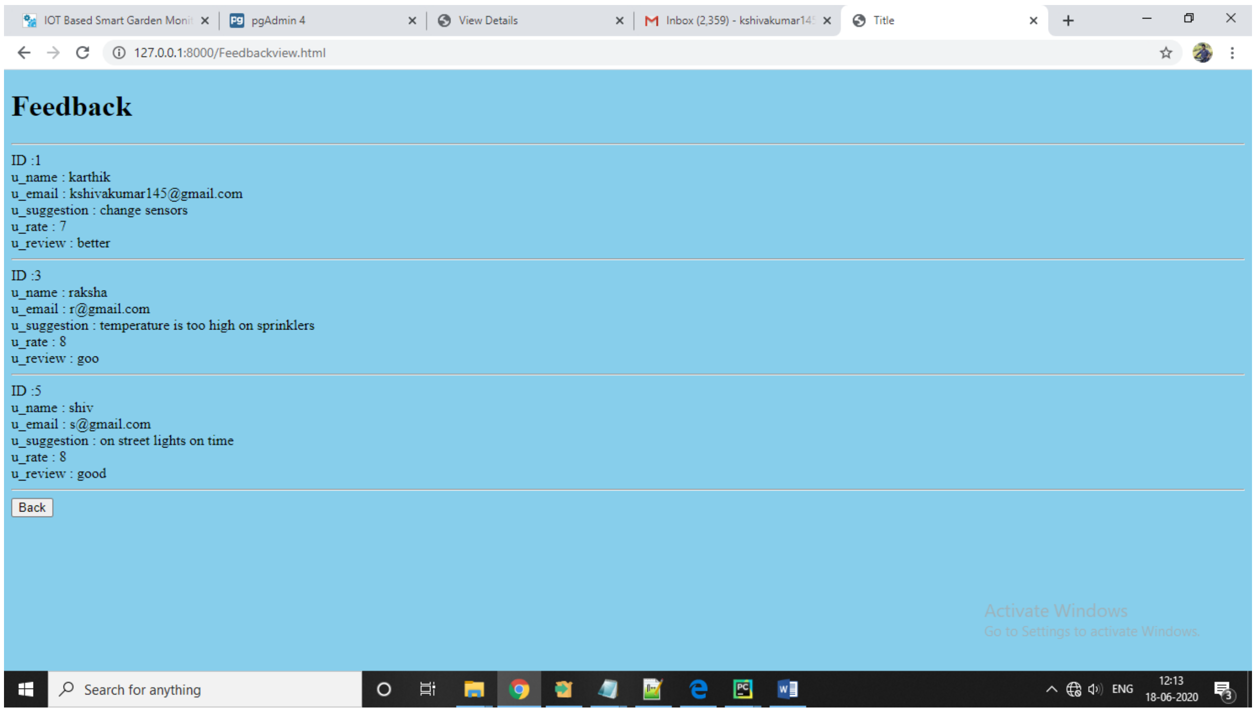
Present Humidity fluctuation = 18

water pump state = 0

water pump is on when it is 1 and otherwise it is 0

Back

Figure 27: Direct sensor information to user by random python code


IOT Based Smart Garden Moni
pgAdmin 4
View Details
Inbox (2,359) - kshivakumar14
Title
127.0.0.1:8000/Feedbackview.html
Search for anything
ENG
12:13
18-06-2020
Activate Windows
Go to Settings to activate Windows.

Feedback

ID :1
u_name : karthik
u_email : kshivakumar145@gmail.com
u_suggestion : change sensors
u_rate : 7
u_review : better

ID :3
u_name : raksha
u_email : r@gmail.com
u_suggestion : temperature is too high on sprinklers
u_rate : 8
u_review : goo

ID :5
u_name : shiv
u_email : s@gmail.com
u_suggestion : on street lights on time
u_rate : 8
u_review : good

Back

Figure 28: User can view all feedback here.

DB Connectivity using Django Framework:

1.Settings.py

```
"""
Django settings for smartpark project.

Generated by 'django-admin startproject' using Django 3.0.5.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.0/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'qrfv9$c@u!(oe3r*@vxqflv3hp4c$mr!nxaltg*qtv3=w1)ztw'
```

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'park.apps.ParkConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'smartpark.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'smartpark.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'smartpark7',
        'USER': 'postgres',
        'PASSWORD': 'qwertyuiop',
        'HOST': 'localhost',

    }
}


# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIR = [
    os.path.join(BASE_DIR,'static')
]
STATIC_ROOT = os.path.join(BASE_DIR,'assets')


Media_URL = "/media/"
Media_root = os.path.join(BASE_DIR,'media')
```

2.View.py

```
from django.shortcuts import redirect, render
from django.http import HttpResponse
from django.contrib import messages
from .models import Sensorstatesupdate
from django.contrib.auth.models import User , auth
from .models import *
from django.shortcuts import render, redirect, HttpResponse, get_object_or_404, HttpResponseRedirect
from django.contrib.auth.forms import UserCreationForm
from .models import *
from .form import *
from django.core.mail import EmailMessage
from django.template.loader import get_template

from django.contrib.auth.models import User, Group
#from rest_framework import viewsets
#from .serializers import *
from .models import FeedbackForm

def home(request):
    return render(request,'home.html')
def ir(request):
    return render(request,'ir.html')
def dht11(request):
    return render(request,'dht11.html')
def dht22(request):
    return render(request,'dht22.html')
def cap(request):
    return render(request,'cap.html')
def adminpage(request):
    return render(request,'Admin.html')
def administrator(request):
    return render(request,'administratorupdate.html')
def administrators(request):
    print("Hello form is submitted")
    previous_temparature_value = request.POST.get('previous_temparature_value')
    present_temparature_value = request.POST.get('present_temparature_value')
    sprinkler_state = request.POST.get('sprinkler_state')

    previous_humidity_value = request.POST.get('previous_humidity_value')
    present_humidity_value = request.POST.get('present_humidity_value')
    water_pump_state = request.POST.get('water_pump_state')


    street_light_state = request.POST.get('street_light_state')
    alarm_state = request.POST.get('alarm_state')
#    next_update_time = request.POST.get('next_update_time')
    description = request.POST.get('description')



    adminupdate =
AdminUpdates(previous_temparature_value=previous_temparature_value,present_temparature_value=present_temp
arature_value,sprinkler_state=sprinkler_state,previous_humidity_value=previous_humidity_value,present_humidity
_value=present_humidity_value,water_pump_state=water_pump_state,street_light_state=street_light_state,alarm_st
ate=alarm_state,description=description)
    adminupdate.save()
    return render(request,'admin.html')


def registerpage(request):
    return render(request,'register.html')

def registerpage1(request):
    return render(request,'register1.html')

def register1(request):
    return render(request,'register1.html')



def loginpage(request):
    return render(request,'login.html')


def information(request):
    return render(request,'info.html')

def review(request):
    return render(request,'review.html')

def alarm(request):
    return render(request,'alarm.html')

def backtohome(request):
    return render(request,'home.html')

def mainpage(request):
    return render(request,'new.html')

def loginmethod(request):
     username   = request.POST['username']
     password   = request.POST['password']

     user = auth.authenticate(username = username , password=password)
#     return render(request,'new.html')

     if user is not None:
         auth.login(request,user)
         return redirect('new.html')
     else:
        messages.info(request,'invalid Credentials')
        print('invalid Credentials')
        return redirect('login.html')

#Register
def register(request):

        first_name = request.POST['first_name']
        last_name  = request.POST['last_name']
        username   = request.POST['username']
        password1  = request.POST['password1']
        password2  = request.POST['password2']
        email      = request.POST['email']

        if password1 == password2:
            if User.objects.filter(username=username).exists():
                messages.info(request,'Username already exists')
                print("User is not created... Username already exists")
            elif User.objects.filter(email=email).exists():
                 messages.info(request,'Email-ID already exists with a registered User')
                 print("User is not created... Email ID exists")
                 return redirect('registerpage')
            else:
                 user =
User.objects.create_user(username=username,password=password1,email=email,first_name=first_name,last_name=
last_name)
                 user.save();
                 print("User Created")
                 return render(request,'login.html')
        else:
            print("User is not created...")
            print("Password not matching...")
            messages.info(request,'Password not matching...')
            return redirect('registerpage')

#def plastic(request):
 #   return render(request,'plastic.html')


def p2l(request):
    return render(request,'p2l.html')

def humidity(request):
    return render(request,'humidity.html')

def temp(request):

#    dests = Sensorstateupdate.objects.all()

    return render(request,'temp.html')

def review(request):
    return render(request,'review.html')

def review1(request):
    return render(request,'review1.html')




def sensorupdate(request):
    tempstatus=Sensorstatesupdate.objects.all
    return render(request,'temp.html',{'all':tempstatus})
```

```
def review(request):

    rev = Register.objects.all()

    return render(request, "review.html", {'all4': rev})

def feedback(request):
    return render(request,'feedback.html')


def feedback1(request):
    return render(request,'feedback1.html')

def Feedbackreview(request):
    print("Hello form is submitted")
    u_name = request.POST.get('u_name')
    u_email = request.POST.get('u_email')
    u_date = request.POST.get('u_date')
    u_suggestion = request.POST.get('u_suggestion')
    u_rate = request.POST.get('u_rate')
    u_review = request.POST.get('u_review')
    feed_back =
FeedbackForm(u_name=u_name,u_email=u_email,u_date=u_date,u_suggestion=u_suggestion,u_rate=u_rate,u_revi
ew=u_review)
    feed_back.save()
    return render(request,'new.html')



def viewdetails(request):
    view = AdminUpdates.objects.all
    return render(request,'viewdetails.html',{'all':view})

def Feedbackview(request):
    Feed = FeedbackForm.objects.all
    return render(request,'Feedbackview.html',{'all1':Feed})


def adminlogin(request):
     username   = request.POST['username']
     password   = request.POST['password']

     user = auth.authenticate(username = username , password=password)
#     return render(request,'new.html')

     if user is not None:
         auth.login(request,user)
         return redirect('administratorupdate.html')
     else:
        messages.info(request,'invalid Credentials')
        print('invalid Credentials')
        return redirect('Admin.html')


def sprinkler(request):
    previous_temparature_value = request.POST.get('previous_temparature_value')
    present_temparature_value = request.POST.get('present_temparature_value')
    sprinkler_state = request.POST.get('sprinkler_state')
# #  next_update_time = request.POST.get('next_update_time')
    description = request.POST.get('description')
    sprin =
sprinklerstatus(previous_temparature_value=previous_temparature_value,present_temparature_value=present_temp
arature_value,sprinkler_state=sprinkler_state,description=description)
    sprin.save()

    if present_temparature_value >= previous_temparature_value:
        return render(request,'temp.html')
    else:
        return render(request,'Admin.html')


def sprink(request):
    return render(request,'sprink.html')

def sprinklerdisplay(request):
    spin = sprinklerstatus.objects.all
    return render(request,'sprinklerdisplay.html',{'all2':spin})



def lightings(request):

    sensor_update = request.POST.get('sensor_update')
    updatetime = request.POST.get('updatetime')
    description = request.POST.get('description')
    str = Streetlightings(sensor_update=sensor_update,updatetime=updatetime,description=description)
    str.save()
    dark='0'
    if sensor_update == '0':
        return render(request,'p2l.html')
    else:
        return render(request,'Admin.html')


def street(request):
    return render(request,'street.html')



def streetlightsdisplay(request):
    street=Streetlightings.objects.all
    return render(request,'streetlightsdisplay.html',{'all3':street})

def water(request):
    previous_humidity_value = request.POST.get('previous_humidity_value')
    present_humidity_value = request.POST.get('present_humidity_value')
    water_state = request.POST.get('water_state')
    updatetime = request.POST.get('updatetime')
    description = request.POST.get('description')
    wat =
Humidityupdation(previous_humidity_value=previous_humidity_value,present_humidity_value=present_humidity_
value,water_state=water_state,updatetime=updatetime,description=description)
    wat.save()

    if present_humidity_value >= previous_humidity_value:
        return render(request,'humidity.html')
    else:
        return render(request,'Admin.html')

def humidityupdatepage(request):
    return render(request,'humidityupdatepage.html')



def humiditydisplay(request):
    humid=Humidityupdation.objects.all
    return render(request,'humiditydisplay.html',{'all5':humid})


def garbage(request):
    garbage_update = request.POST.get('garbage_update')
    updatetime = request.POST.get('updatetime')
    description = request.POST.get('description')
    ala = alarmupdate(garbage_update=garbage_update,updatetime=updatetime,description=description)
    ala.save()
    found='1'
    if garbage_update == '1':
        return render(request,'alarm.html')
    else:
        return render(request,'Admin.html')



def alarmupdatepage(request):
    return render(request,'alarmupdatepage.html')



def alarmdisplay(request):
    alar=alarmupdate.objects.all
    return render(request,'alarmdisplay.html',{'all6':alar})

#WE MAY THINK WE ARE NARTURING OUR GARDEN,BUT OF COURSE ITS OURGARDEN THATIS REALLY
NURTURING US
#EVERYTHING THAT SLOW US DOWN AND FORCES PATIENCE EVERYTHING THAT SETS US BACK INTO
THE SLOW CIRCLES OF NATURE IS A HELP GARDNING IS AN INSTRUMENT OF GRACE



def tempadmindisplay(request):
    spin = sprinklerstatus.objects.all
    return render(request,'tempadmindisplay.html',{'all2':spin})
def humadmindisplay(request):
    humid=Humidityupdation.objects.all
    return render(request,'humadmindisplay.html',{'all5':humid})

def feedadmindisplay(request):
    Feed = FeedbackForm.objects.all
    return render(request,'feedadmindisplay.html',{'all1':Feed})


def alarmadmindisplay(request):
    alar=alarmupdate.objects.all
    return render(request,'alarmadmindisplay.html',{'all6':alar})

def streetadmindisplay(request):
    street=Streetlightings.objects.all
    return render(request,'streetadmindisplay.html',{'all3':street})
```

CONCLUSION :

In this project we presented the architecture and the implementation of a smart gardening system. The system consists of two types of sensors motes (DHT11,DHT22,Capacitive and IR sensor), special soil humidity sensors, with the use of relays python application that is used for data collection. Performance evaluation showed that our system manages to maintain soil humidity levels regardless of external factors (i.e. variations at temperature and sunlight). It also proved that the system is aware of the different watering needs each.

Various ways have been witnessed that Internet of Things is going to change our lives soon. Gardening is also surely to be a subject of this new technology. In this we have done the potentials how the IoT technology can be linked to smart gardening activities as a smart system. This system does not just provide easiness to gardening, but also facilitates a new culture where people desire a quality of life and well-being especially in an aging society. It seems that there are already a number of automated smart gardening systems, but there is still big room for improvement with the IoT technology