Alexandria university
Faculty of Engineering
Electrical Power and Machines
1st Term

SMART HOME PROJECT

Microprocessor

Student Name: Mohamed Ahmed Mahmoud Menazaa

Student Name: Nouran Ahmed El-Sayed Ahmed Darwish

Student Name: Mayar Magdy Ragab Ibrahim

Student Name: Dina Ahmed Hussien Fawzy Youssef

Student Name: Muhammed Mahmoud Ibrahim Elsayed

❖ Abstract

Internet of things plays an important role in human life as well as in the educational field because they are able to provide information and complete the given tasks while we are busy doing some other work.

Smart-home technology encompasses a wide range of everyday household devices that can connect to one another and to the Web.

This project aims to present the Smart Home concept. Using an ESP8266 micro controller, with IP connectivity for accessing and controlling devices and appliances remotely using Android based Smart phone app.

We control the 7-segment counter by using NodeMCU and LDR sensor to measure the intensity of light and display it to the application or web page.

We used Flutter framework to develop our application and connect it with IP address of ESP8266.

ESP8266 is used as a Wi-Fi technology. The proposed system consists of a hardware interface and software interface which we will discuss in our document.

❖Project Description

- In this project we will discuss the interfacing of 7 segment display with NodeMCU ESP8266-12E Wi-Fi module. Numbers are going to be displayed on 7 segment display. Seven segment display control is given to a web page. Web page contains buttons. There are 10 buttons, and each button is assigned a number from 0 to 9. Pressing any button will display its corresponding number on the 7-segment display.
- NodeMCU is working as a server. It is serving a web page. To access a server, we need a client. For our project client can be any

device which can connect to a Wi-Fi network and has a browser in it.

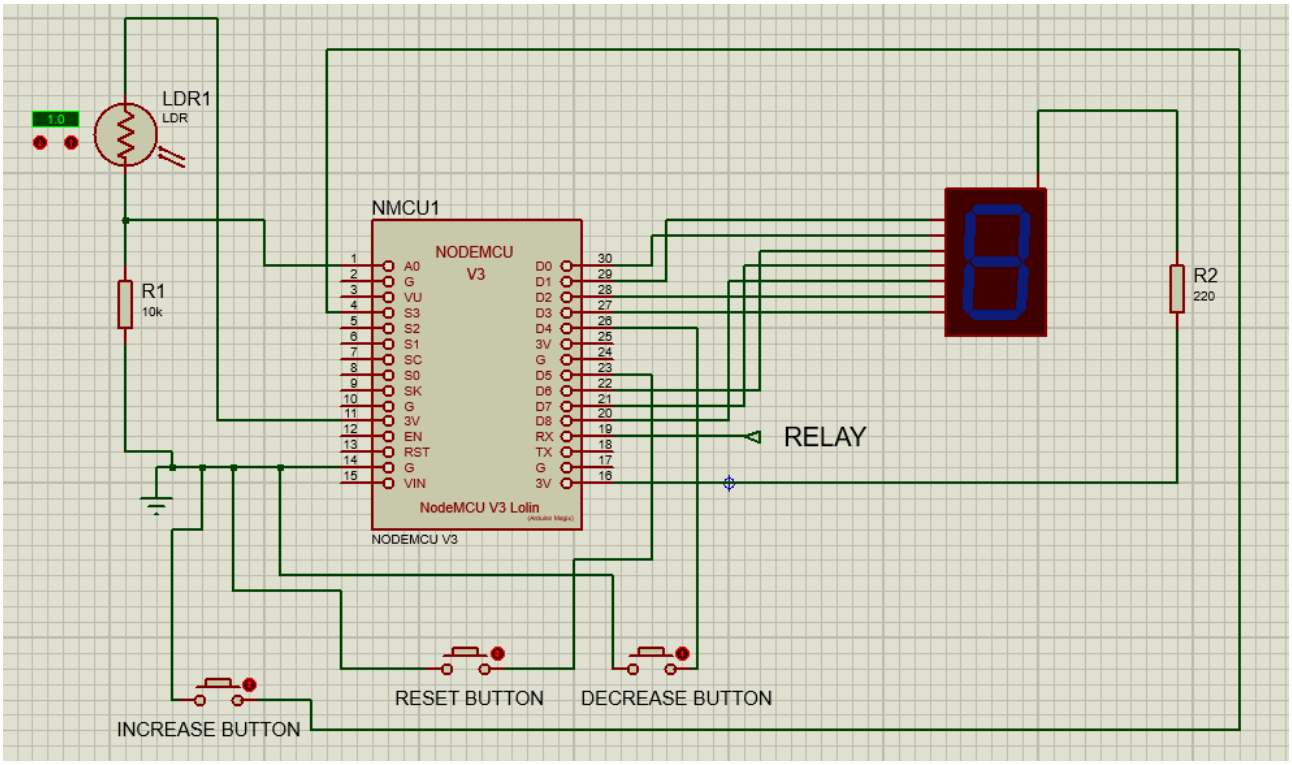
- Desktop computer. laptop, mobile or a notebook can be a client in our case. We have to enter the web server (NodeMCU) address in our browser to access the web page (7-segment control page).
- In this report we will explain hardware implementation, hardware description, software description Flutter and MIT app inventor.

❖ Hardware Description

- NodeMCU is an open-source development board and firmware based in the widely used ESP8266 -12E Wi-Fi module. It allows you to program the ESP8266 Wi-Fi module with the simple and powerful LUA programming language or Arduino IDE.
- These features make the NodeMCU extremely powerful tool for Wi-Fi networking.
- NodeMCU has 17 GPIO pins which can be assigned to various functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light, and Button programmatically.
- Each digital enabled GPIO can be configured to internal pull-up or pull-down or set to high impedance.
- We will connect NodeMCU with the 7 segment which is a form of electronic display device for displaying decimal numerals.
- We will connect NodeMCU with LDR which stands for Light Dependent Resistor, which is a passive electronic component, basically a resistor which has a resistance that varies depending on the light intensity.
- We will connect 3 push buttons (Increase button – Decrease button – Reset button).
- Increase button: to increase the value of the 7-segment.
- Decrease button: to decrease the value of the 7-segment.

- Reset button: to change the value of the 7-segment into zero.
- Two resistors are connected (10k ohm – 220 ohm).

❖Hardware Implementation



- **Used Pins:**
  - Data Pins: (GPIO5, GPIO16, GPIO12, GPIO13, GPIO15, GPIO4, GPIO0)
  - Input Pins for buttons (Increase, Decrease, Reset): (GPIO10, GPIO2, GPIO14)
  - Relay Pin: GPIO3
  - LDR pin: ADC0
  -

❖Software Description

- **Header file:**
  For handling the hardware and serving the sensors data over the network, we used the following module (machine, time, network, socket, ujson).

```
# Import modules
from machine import Pin,ADC # Pins for GPIO pins and ADC for reading analog data from the LDR
from time import sleep_ms, time   # just for the delays in the code
from network import WLAN , AP_IF # Import WirelessLAN and AccessPoint Internet Family
from socket import socket, AF_INET, SOCK_STREAM  # import socket class
import ujson
```

- **Lookup tables:**
  For pin configuration to display different numbers, we use precomputed lookup tables

```
############### Initial data and variables ##################
### 7 segment lookup tables ###
#Array with data pins values to show any digit on 7 segment
draw_digit=[[0,0,0,0,0,0,1]#0
    ,[1,0,0,1,1,1,1]#1
    ,[0,0,1,0,0,1,0]#2
    ,[0,0,0,0,1,1,0]#3
    ,[1,0,0,1,1,0,0]#4
    ,[0,1,0,0,1,0,0]#5
    ,[0,1,0,0,0,0,0]#6
    ,[0,0,0,1,1,1,1]#7
    ,[0,0,0,0,0,0,0]#8
    ,[0,0,0,0,1,0,0]]#9
#Array with data pins values to show 0 with slow motion
zero_slow=[[0,1,1,1,1,1,1]
    ,[0,0,1,1,1,1,1]
    ,[0,0,0,1,1,1,1]
    ,[0,0,0,0,1,1,1]
    ,[0,0,0,0,0,1,1]
    ,[0,0,0,0,0,0,1]
    ,[0,0,0,0,0,0,1]]
#Array with data pins values to show 9 with slow motion
nine_slow=[[1,1,1,1,1,1,0]
    ,[1,1,1,1,1,0,0]
    ,[0,1,1,1,1,0,0]
    ,[0,0,1,1,1,0,0]
    ,[0,0,0,1,1,0,0]
    ,[0,0,0,0,1,0,0]
    ,[0,0,0,0,1,0,0]]
```

- **Initializing variables and pins:**
  As our algorithm demand set of variables for the computation and control purposes, we initialize the following variables to be used and their purpose described in the comments, and the function of GPIO pins should be specified before usage so we define pins as input or output according to their functions and we make use of the internal pull up resistors on the input pins.

```
### initial variable data ###
Counter = 0  #Initialize Counter
LDR = ADC(0) # config the analog input
stop_resume = True  # for stop and resume functionality of the whole app
stop_resume_state = ""



#SET INPUT & OUTPUT PINS
data_pins=[5,16,12,13,15,4,0] #A__B__C__D__E__F__G#
input_pins=[10,2,14] #increase__decrease__reset#
bulb = Pin(3,Pin.OUT)
out_pins = []
in_pins = []
for k in data_pins:
    out_pins.append(Pin(k,Pin.OUT))
for k in input_pins:
    in_pins.append(Pin(k,Pin.IN,Pin.PULL_UP))
```

- **Seven segment control functions:**

  As the lookup tables are in ascending order based on the number of the pin configuration, we access the right number configuration list, and we iterate through it to control each pin in individually and we set functions to manipulate the counter value which we control the seven-segment display based on it.

```
### to output the counter data to the seven segment display
def seven_segment(Counter):#Draw Counter on 7 Segment
    if Counter>=0 and Counter<=9:
        for k in range(7):#Draw Counter Normally
            out_pins[k].value(draw_digit[Counter][k])
        sleep_ms(500)
    elif Counter<0:
        for k in range(7):#Draw Nine Slow Motion
            for l in range(7):
                out_pins[l].value(nine_slow[k][l])
            sleep_ms(500)
    else :
        for k in range(7):#Draw Zero Slow Motion
            for l in range(7):
                out_pins[l].value(zero_slow[k][l])
            sleep_ms(500)
```

```
#### To change the counter value
def increase():#Increase Counter Function__is CALLED Any where we need increase counter
    global Counter
    Counter+=1
    seven_segment(Counter)#Send Counter to show on 7 Segment
    if(Counter>9):
        Counter=0

def decrease():#Decrease Counter Function__is CALLED Any where we need decrease counter
    global Counter
    Counter-=1
    seven_segment(Counter)#Send Counter to show on 7 Segment
    if(Counter<0):
        Counter=9

def reset():#Reset Counter Function__is CALLED Any where we need reset counter
    global Counter
    Counter=0
    seven_segment(Counter)#Send Counter to show on 7 Segment
```

- **Interrupt and digital input:**

1) digital input:

because of the mechanical constrains of the digital input device (push buttons) which is represented in the form of the debouncing issue, we used software sampling technique to overcome this issue.

```
### for taking the input from the pushbuttons


def debounce(pin):#Debouncing Function To Avoid Non Perfect Contact
    previous_value = None#Initial A Temp Variable
    for k in range(10):#Take 10 Samples of Signal
        current_value = pin.value()#Take Sample
        if previous_value != None and previous_value != current_value:
            return None #If Rippled Return None
        previous_value = current_value
    return previous_value#AFTER 10 Samples return New State
```

2) interrupt and interrupt service routine:

as all the GPIO pins of ESP8266 have the interrupt functionality we use interrupt with pins GPIO10, 2, 14. and we use interrupt handler function as follow:

```
def increase_interrupt(pin):#Interrupt Routine for Increment
    d = debounce(pin)#Check Bouncing
    if not d:
        increase()#After 10 Samples Excute Increment

def decrease_interrupt(pin):#Interrupt Routine for Decrement
    d = debounce(pin)#Check Bouncing
    if not d:
        decrease()#After 10 Samples Excute Decrement

def reset_interrupt(pin):#Interrupt Routine for Reset
    d = debounce(pin)#Check Bouncing
    if not d:
        reset()#After 10 Samples Excute Reset



# SET Interrupts Pins
handlers=[increase_interrupt, decrease_interrupt, reset_interrupt]
for k in range(3):
    in_pins[k].irq(trigger=Pin.IRQ_FALLING, handler=handlers[k])
```

- **Json data formatter:**

  We use ujson module to format our data in proper standard format of json to serve through our API.

```
#####  send json data
def send_json(Counter, state, stop_resume_state):

    jsonData = {"Counter":Counter,"LDR": state, "state": stop_resume_state}
    encoded = ujson.dumps(jsonData)

    return encoded
```

- **NETWORKING AND CONTROL:**

1) Networking:

We set the ESP8266 as an access point you create its own Wi-Fi network and nearby Wi-Fi devices can connect to it (like smartphone or your computer).

We'll show you how to set the ESP8266 as an access point in your web server projects.

At first, we imported WLAN and AP_IF from network module that makes ESP8266 works as an access point, then we created an object named Wi-Fi from WLAN that works at access point internet family, and we configure the access point name and password then we turn on access point.

```
WIFI = WLAN(AP_IF) # Create a WLAN object as AccessPoint
WIFI.config(essid='Seven Segment Controller',password='7777*7777',authmode=4) #Configure Access Point Name , Encryption and Password
WIFI.active(True)  #Turn Access Point on
while not WIFI.isconnected():
        pass   # Don't Skip until Connection Success  Note That ESP IP is 192.168.4.1 in Default
```

Once the Wi-Fi is set up the way to access the network is by using sockets. A socket represents an endpoint on a network device, and when two sockets are connected together communication can proceed. Internet protocols are built on top of sockets, such as email (SMTP), the web (HTTP), telnet, ssh, among many others. Each of these protocols is assigned a specific port, which is just an integer. Given an IP address and a port number you can connect to a remote device and start talking with it.

First, we import AF_INET, SOCK_STREAM from socket module which they refer to IPv4 and TCP protocol, then we created object that refers to socket which depend on IPv4 and TCP protocol, we gave this object the default address of ESP8266 and HTTP port (80).

```
s = socket(AF_INET,SOCK_STREAM)   #AddressFamily:IP v4 | TCP Protocol
s.bind(('',80))  #Assign socket to ESP Address on Port 80 (HTTP PORT)
s.listen(10)  #Start accepting TCP connections with maximum 10 connections
```

After we do those commands, we do a while loop with try in it to avoid connection issues, stopping code and make it resets.

Then we accept connections from object connection, sender_address.

We set connection timeout to 3 seconds as if the connection exceeds 3 seconds it will close the connection and start looking for a new connection, we limit the received connection to 1024 bytes, then we remove set timeout.

```
while(1):
    try:
        ####### connection and request part #########
        #Start Accepting New connection and make new accept object to use and take client address and port
        connection,sender_address=s.accept()
        connection.settimeout(3)#Set Connection timeout to 3 Seconds
        request=connection.recv(1024) # receive data with maximum 1024 Bytes
        connection.settimeout(None) # Unlimited Timeout
        request = str(request)#Cast Byte Object to String
```

Using the find method from the string class to filter the HTTP request to identify the type of request.

```
####### filtering the request and request part #########
increase_request =request.find('GET /?increase')#Search for increase parameter
decrease_request = request.find('GET /?decrease')#Search for decrease parameter
reset_request = request.find('GET /?reset')#Search for reset parameter
on_request = request.find('GET /?on')#Search for decrease parameter
off_request = request.find('GET /?off')#Search for reset parameter
stop_resume_request = request.find('GET /?sr')#Search for reset parameter
```

- **Control:**

  Based on the type of the request, we control the attached devices as follow:

  1) The counter value on the seven-segment display.

```
##    stop / resume functionality
if(stop_resume_request != -1):
          stop_resume = not stop_resume

##        system state
if   stop_resume:
     stop_resume_state = "Running"
else:
     stop_resume_state = "Stopping"
```

And based on the value of the keypad:

```
# keypad code
for i in range(10):
    if (request.find('GET /?num'+str(i)) != -1):
        Counter = i
        seven_segment(Counter)
```

2) Measuring the light bulb intensity with the LDR sensor every one second:

```
## updating the current time variable
current_time = time()

if (current_time - previous_time > 1):
    previous_time = current_time
    # The LDR Value
    ldr_value = LDR.read()
    if ldr_value > 5:
        state = "weak"
    if ldr_value > 600:
        state = "moderate"
    if ldr_value > 900:
        state = "intensive"
```

3) Switching the light bulb (on/off):

```
#   Relay (bulb) control
if(on_request != -1):
    bulb.value(1)
elif(off_request != -1):
    bulb.value(0)
```

- **Rendering:**
  We serve the data of the states and sensors as json format based on the server-side rendering architecture thus we close the connection after sending the response, and in case of any error we handle it by resetting the connection.

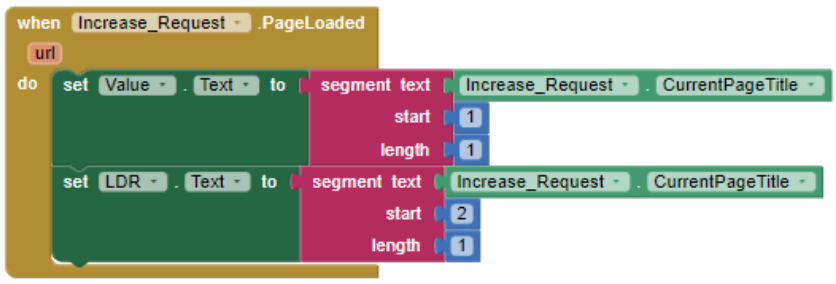
```
    #send web page after updating counter
    connection.sendall("HTTP/1.0 200 OK\r\nServer: NodeMCU\r\nContent-Type: text/json\r\n\r\n"+send_json(Counter, state, stop_resume_state))
    connection.close()#close connection


except :
    connection.close()#In case error close connection
```

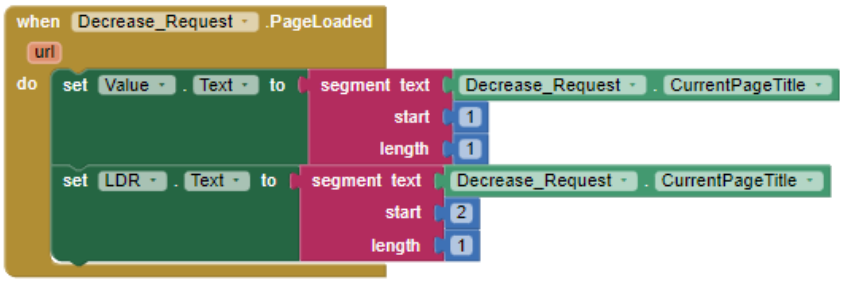
❖MIT App Inventor

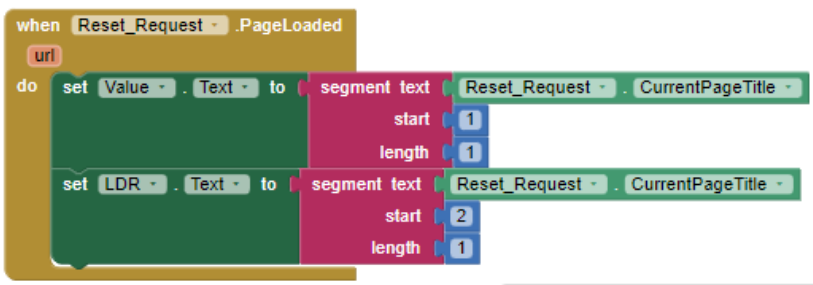
- MIT App Inventor is an online platform designed to teach computational thinking concepts through development of mobile applications. Students create applications by dragging and dropping components into a design view and using a visual blocks language to program application behavior.
- The MIT App Inventor user interface includes two main editors: the design editor and the blocks editor. The design editor, or designer is a drag and drop interface to lay out the elements of the application's user interface (UI). The blocks editor is an environment in which app inventors can visually lay out the logic of their apps using color-coded blocks that snap together like puzzle pieces to describe the program.
- To aid in development and testing, App Inventor provides a mobile app called the App Inventor Companion (or just "the Companion") that developers can use to test and adjust the behavior of their apps in real time. In this way, anyone can quickly build a mobile app and immediately begin to iterate and test.
- Components are made up of three major elements: properties, methods, and events.

- Properties control the state of the component and are readable and/or writable by the app developer.
- Methods operate on multiple inputs and possibly return a result. Events respond to changes in the device or app state based on external factors.
- In MIT App Inventor, users code application behavior using a block-based programming language. There are two types of blocks in App Inventor: built-in blocks and component blocks.
- The built-in blocks library provides the basic atoms and operations generally available in other programming languages, such as Booleans, strings, numbers, lists, mathematical operators, comparison operators, and control flow operators.
- Developers use component blocks (properties, methods, and events) to respond to system and user events, interact with device hardware, and adjust the visual and behavioral aspects of components.

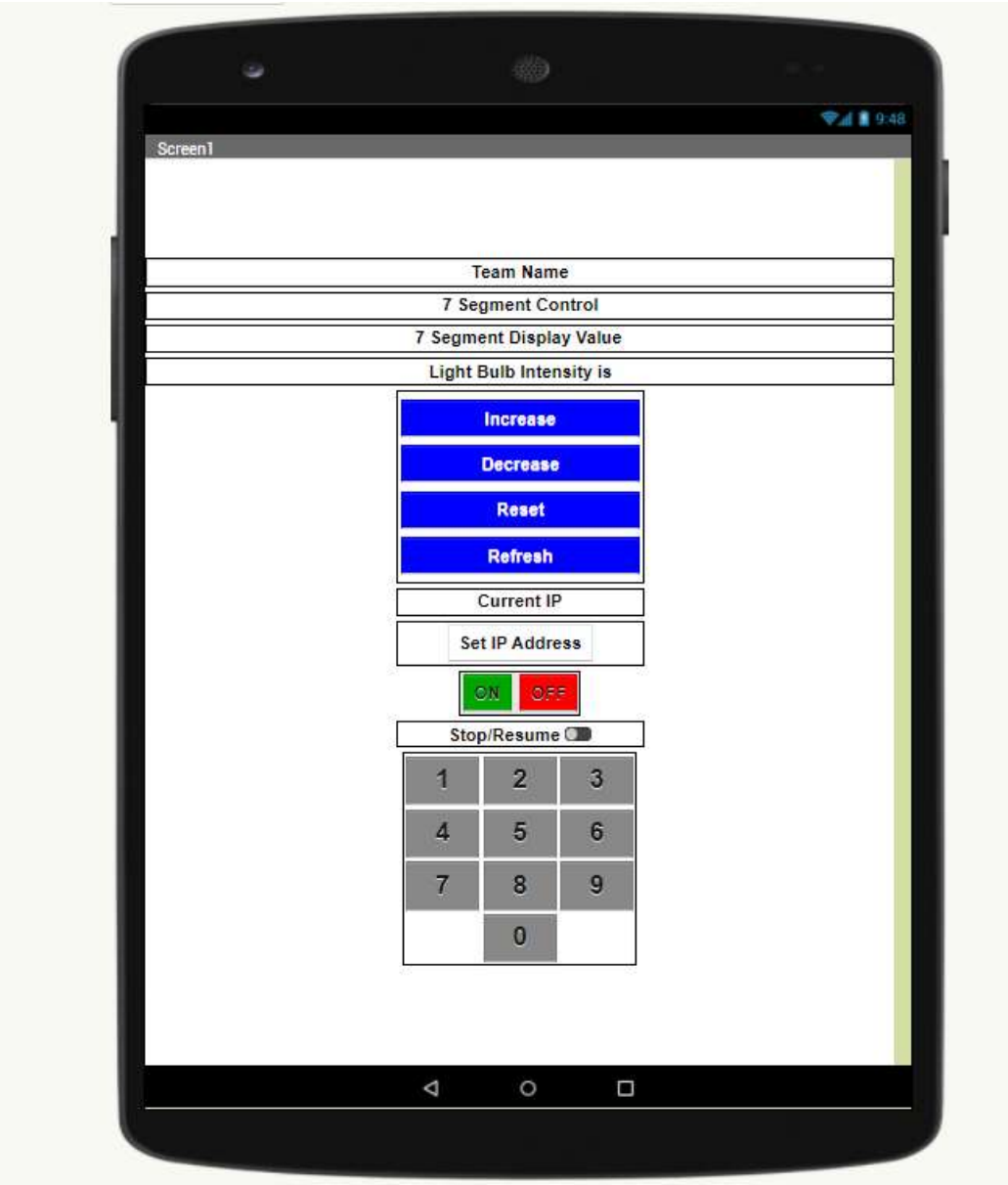
Sample of Blocks







User Interface



❖ **We created the MIT App Inventor, but we faced a problem while using the app:**

The 7- segment display value is displayed in a wrong way

That's why we used Flutter instead.

❖ Flutter App

- **Overview:**

  We used flutter framework and dart language to develop our mobile application.

  We choose flutter because of its cross-platform capability which mean we only have one-code base for our targeted platforms, flutter also comes with built-in batteries philosophy and has great community and libraries, helped us to develop our app faster and make it more reliable with one-code base to maintain.

User interface:

- **<u>The flutter app breakdown:</u>**

  Our app based on three main components all of them rely on stateful widget functionality of the flutter framework.

1) **raised button widget:**

```
RaisedButton.icon(
    onPressed: () async {
      var res = await http.get('http://192.168.4.1/?on');
      setState(() {
        if (res.statusCode == 200) {
          var jsonResponse = jsonDecode(res.body);
          Counter = jsonResponse['Counter'];
          LDR = jsonResponse['LDR'];
          state = jsonResponse['state'];
        }
      });
    },
    icon: Icon(
      Icons.lightbulb,
      color: Colors.yellow,
      size: 30,
    ),  // Icon
    label: Text(
      "ON",
      style: TextStyle(fontSize: 22),
    )),  // Text, RaisedButton.icon
```

2) **row and text widgets:**

```
Row(
  children: <Widget>[
    Text(
      'The 7 Segment Value is $Counter',
      style: TextStyle(fontSize: 28, color: Colors.lightBlue),
    )  // Text
  ],  // <Widget>[]
),  // Row
```

3) **numeric keyboard widget:**

```
NumericKeyboard(
  onKeyboardTap: _onKeyboardTap,
  textColor: Colors.blueAccent,
  rightButtonFn: () {
    setState(() {
      text = text.substring(0, text.length - 1);
    });
  },
  rightIcon: Icon(
    Icons.backspace,
    color: Colors.red,
  ),  // Icon
  leftButtonFn: () async {
    print('left button clicked');
    var res = await http.get('http://192.168.4.1/?num$text');
    setState(() {
      if (res.statusCode == 200) {
        var jsonResponse = jsonDecode(res.body);
        Counter = jsonResponse['Counter'];
        LDR = jsonResponse['LDR'];
        state = jsonResponse['state'];
      }
    });
  },
  leftIcon: Icon(
    Icons.check,
    color: Colors.green,
  ),  // Icon
),  // NumericKeyboard
```