Final Report

Physical Computing

Spring 2018

Professors:

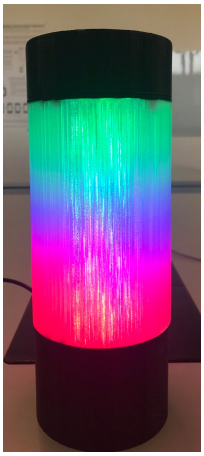
Marc Langheinrich

Ivan Elhart

Anton Fedosov

Simulator Weather Lamp

(SWL)



Sebastian Hidalgo, Amirehsan Davoodi, Tong Pan

“Bring the outside to the inside”

Introduction

For our physical computer project, we decided to build a Simulator Weather Lamp (SWL). The main function of the lamp is to acquire the forecasted weather condition from the internet and provide to the user an ambient light with the color resembling that of the weather. Additional features discuss more in detail in the following sections were added that enables the user to interact with the device and receive more weather related information.

Motivation

Our motivation behind this project was inspired by the amazing natural scenery and great weather conditions that we are able to experience in Lugano, Switzerland. Most of us, more often than not, our daily routine confines us to being in closed spaces for long periods of time. Let it be in schools, hospitals, or at work, these places with its surrounding walls keep us from admiring and enjoying the beauty of nature outside. With this in mind, we decided to approach the problem in a different way, so instead we asked: why not bring what we are missing outside these walls inside?
Whether it is the suiding view of rainfall or the relaxing view of a bright sunny day, we wanted to bring these sensations as close as possible to our daily life. From this, we created the motto of our project to be:

“Bring the outside to the inside”

Prototype Description

Basic Functionalities

As described earlier, SWL is an smart lamp that connects to the internet and retrieves the current weather conditions and responds by lighting up with the color resembling that of the weather. The location (city) from where the weather is retrieved can be chosen from five predetermined locations, giving the user a wider range of options. Picture 1. shows three different weather scenarios with the lamps’ corresponding response.

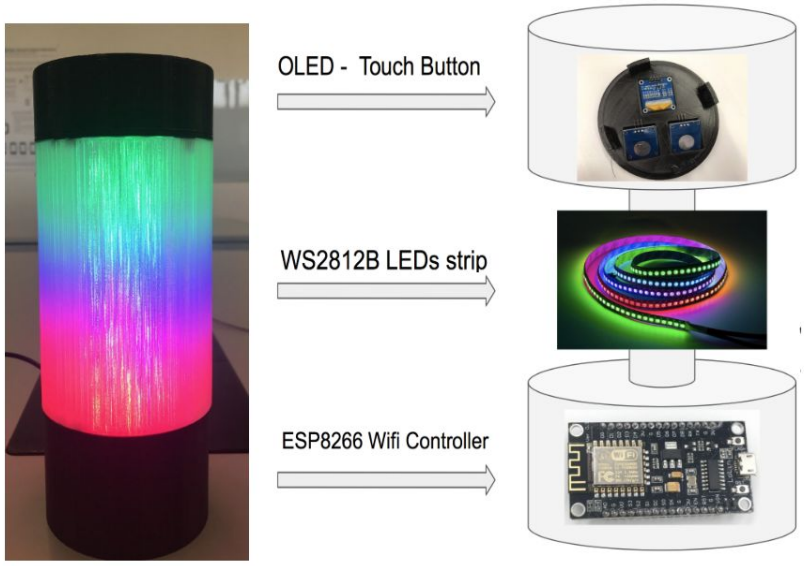
Picture 1. SWL weather response

Additionally, the lamp comes equipped with a small OLED display where information about the weather and states of the lamp are shown. It also comes equipped with the possibility to decrease or increase the light intensity based on the users desired. In the following sections, we will discuss the architecture of the lamp along with the hardware and software implementation performed in this project.

Components and Architecture

Physical Structure

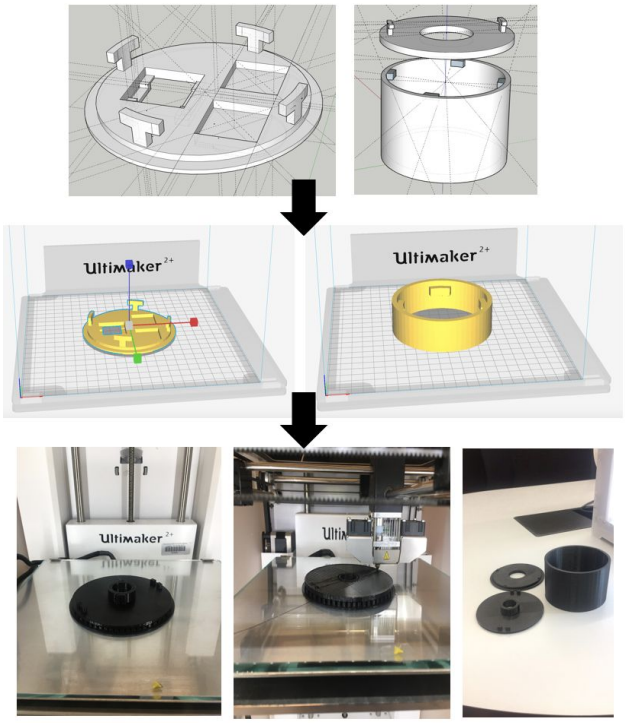
Picture 2 shows the physical structure (left) of the lamp alongside an *"X-ray"* version (right) showing the different hardware components and its physical location within the lamp as implemented in this project.



Picture 2. Lamp's Body

SWL's Frame (Body)

We decided to custom 3D print the entire body of the lamp in order to satisfy our own design desires and be able to accommodate the electronic hardware components without space limitation. For this process, we used the *SketchUp* google tool to design and generate our 3D drawings and then used the *Ultimaker Cura* software tool to prepare our 3D models for printing. Picture 3 shows this process followed from top to bottom: *SketchUp drawing, Ultimaker Cura* model, and 3D printed models.



Picture 3. 3D Printing process

Hardware Implementation

Figure 1, depicts a block diagram of SWL and how all the different hardware components come together.

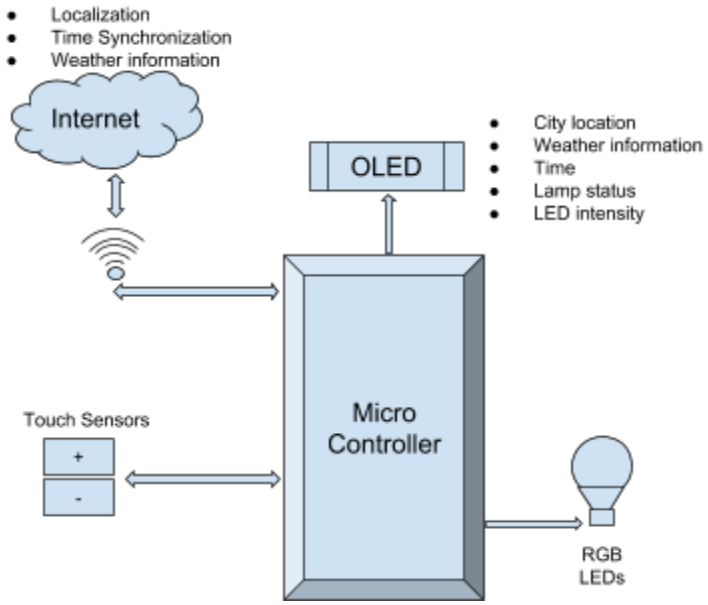
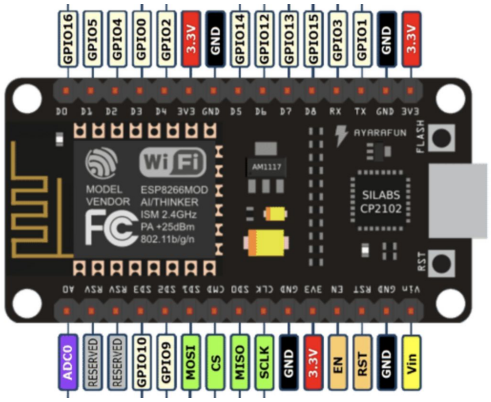


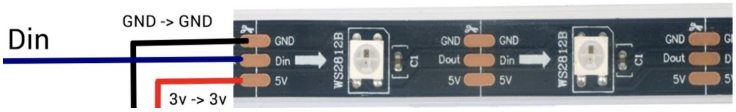
Figure 1. SWL Block Diagram

ESP8266 Wifi-Microcontroller - As shown in Picture 4, this arduino based controller provided all the necessary input and output pins needed for the different lamp functionalities. Additionally, it contains an embedded WiFi module, more specifically NodeMCU which allowed us to connect the lamp to a wireless network and retrieve the necessary weather information. [1]



Picture 4. ESP8266-Wifi Microcontroller

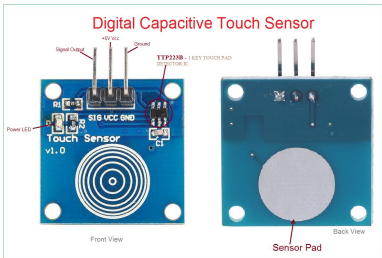
WS2812B LEDs - This multicolor LED strip (RGBW) as shown in Picture 5, provided us with enough flexibility in terms of color combinations. The communication protocol used is I2C which allowed us to map to every individual LED independently and control its RGBW values for color coding as well as intensity. [2]



Picture 5. WS2812B LEDs

OLED display SSD1306 - This is a single-chip consisting of 128 segments and 64 commons. It embeds display RAM and oscillator, minimizing the use of extra external components and reducing power consumption. Communication is established through series compatible Parallel Interface, I2C interface or Serial Peripheral Interface. [3]

Digital Touch sensor (Buttons) - It is a very simple to use sensor with three terminals that allows for an easy interfaced with a wide variety of microcontrollers. This is a sensor based on TTP223B IC digital capacitance with good response time [4]



Picture 6. TTP223B Capacitive Touch Sensor

Figure 2. Shows how the components were connected using a breaded board.

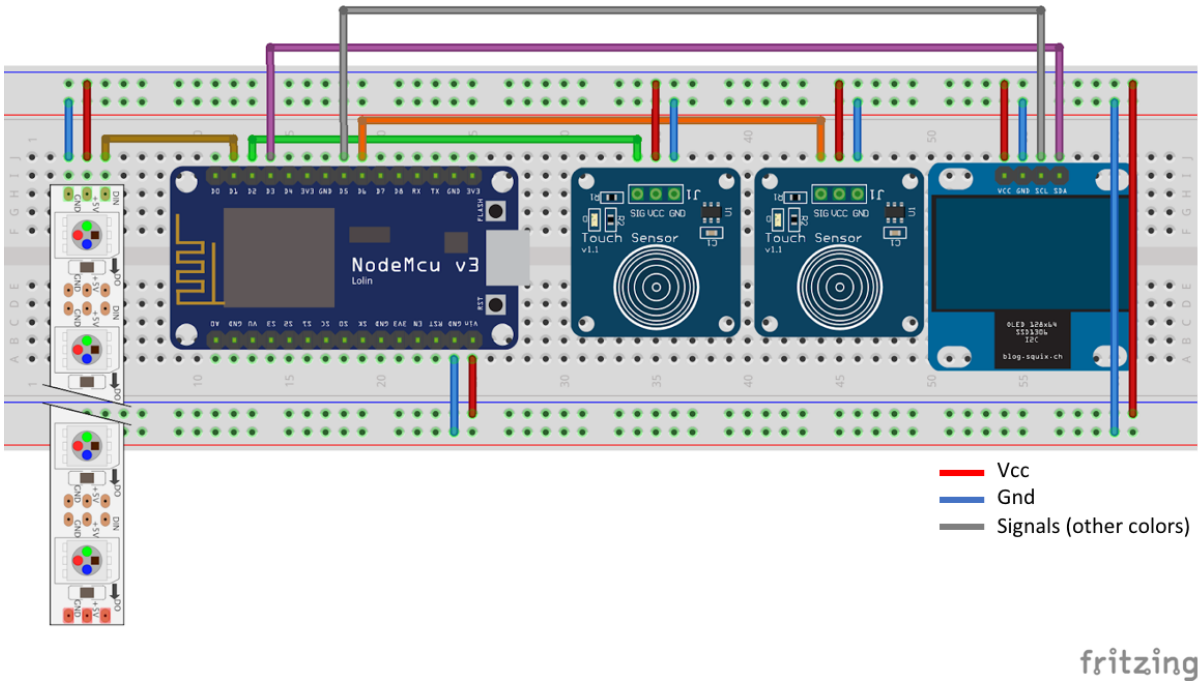


Figure 2. Breadboard Schematic

The following Table 1. details the mapping of the pins to the different hardware components in the lamp.

Table 1. ESP8266-Wifi Microcontroller

Pin Number	Type (I/O)	Connected to	Pin type	Description
D3 D5	output	OLED	SDA SCL	Display control
D2 D6	input	Touch Sensors	2x SIG	Input sensors
D1	output	LEDs	Din	LED control

Software Implementation

Libraries

The language used for programming all functionalities of SWL was C/C++ using the Arduino IDE arduino platform. The libraries presented as following were used to communicate and control the different hardware components.

WiFi-library (ESP8266WiFi) - We used this library based on WiFi.h from Arduino WiFi shield library to connect to the internet and send or receive weather information or current time from the APIs. We simply connected to an access point with the name “ssid” and password “password” by the following the command:

```
WiFi.begin(ssid, password);
```

Then we checked the status of connection by this command.

```
while (WiFi.status() != WL_CONNECTED) {}
```

API-Clock (*WorldClockClient*) - This library allow us to get the current time of the selected location by sending an *http* request to a free and open time API with the following URI:

```
https://thingspeak.com/channels/CHANNEL_ID/feeds.json?results=2&api_key=API_KEY
```

ArduinoJson - library - This library allowed to parse the data we are interested in and process the received json file to extract needed information. For instance, here we get the description of the current weather based on the corresponding element of the received json file or get the maximum or minimum temperature of the day.

LED strip (Adafruit NeoPixel) Library- This library allowed us to control each LED in particular (its parameters: color and intensity) through use of I2C communication protocol. I2C assigns a particular address to each LED enabling individual control of each LED.

LCD (OLEDDisplayUi) SSD1306- library - This library allowed us to display figures or text using easy lines of codes, where we can set the text alignment (<pixel start-row>, <pixel start-column>, <text>) and font.

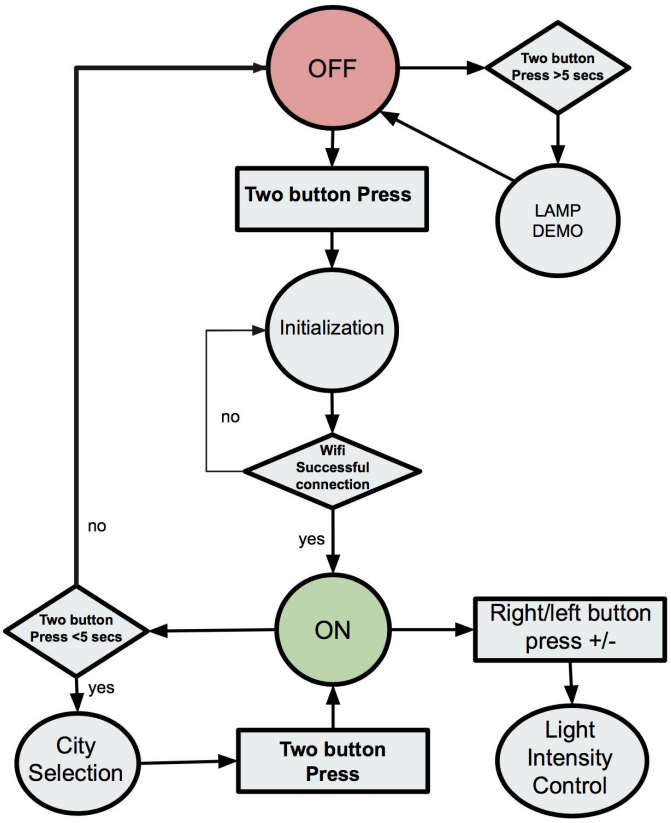
```
display.setTextAlignment(TEXT_ALIGN_CENTER);
display.setFont(ArialMT_Plain_10);
display.drawString(64, 32, "WELCOME!"); ()
```

In addition to simple text displays there are some helper functions which allow complex pattern to show as well for fun.

SWL Logic: State Machine

The code written for SWL follows a state machine implementation topology as shown in Picture 7. This state machine can be broken down into three main states:

1. OFF-State
2. Initialization-State
3. On-State



Picture 7. SWL State Machine

Off-State

The Off-State depicted by the red circle in the state machine in Picture 7 is the initial state SWL enters as soon as power is applied to the unit. This can be done by connecting to any source of power rated at +5 volts and 1 Amp of current. i.e. (USB-connection). During this stage, the ESP8266 Wifi-Controller is awake waiting to receive inputs coming from the touch buttons while at the same time displaying information to the LCD screen as shown in Picture 8.

Picture 8. Off-State LCD display

From the Off-State the lamp can enter Demo-mode or Initialization-State. If touch buttons are pressed for more than 5 seconds, it will enter Demo-mode. During the Demo-mode, the lap will exercise all its possible functionalities. The LCD is commanded to display a certain weather condition i.e sunny and at the same time the LEDs will be commanded to turn yellow. This will repeat for all types of weather conditions that have been programmed in the lamp. Following the weather demonstrations, light intensity increase/decrease will also be shown with the LCD showing a progress bar accordingly. Once Demo-mode has completed the lamp returns to Off-State.
If instead, normal operation of the lamp is desired, both touch buttons would need to be pressed for less than 5 seconds and the lamp will enter Initialization-State.

Initialization-State

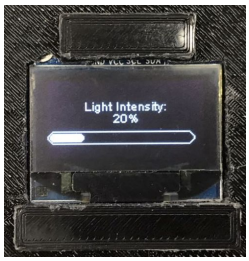
During the Initialization-State, the Wifi-Controller will start searching for a valid internet connection given a pre-configured SSID name and password. Once a successful internet connection is achieved the controller will start searching for a valid API connection. Once a valid API connection is achieved the Initialization-State is completed and the lamp enters On-State. At all times, the progress of this process is broadcasted to the LCD and no inputs from the touch buttons are enabled.

ON-State

When the lamp enters the On-State, the weather condition received at that time will trigger the lap to switch the LEDs to the color resembling that particular weather condition as shown previously. Analogously, the LCD will display additional information of the current location where the weather information is coming from as shown in Picture 9. For this particular instance Lugano.

Picture 9. On-State LCD display

Additionally, all functionalities exercised during the Demo-mode and more are available. As shown in Picture 10. From this state, the light intensity can be control through the input of the touch buttons to the controller. Similarly, the LCD will react to this action by showing a progress bar displaying the percentage intensity.



Picture 10. Light Intensity Progress Bar

Another functionality that becomes available in this state is the possibility of selecting the weather location from five predetermine cities around the world. It is possible to enter this city selection by pressing down on both buttons for less than 5 seconds. Picture 11. below shows an example of this menu. Here, the cursor can be moved up or down with either of the two touch buttons and a location can be selected by pressing on both buttons for a few seconds.

Picture 11. City Selection Menu

Selecting a city, will bring the lamp back into the On-State and the displayed will be update with the new city selected. Similarly the light in the Lamp will be updated to the

current weather of the new location. Finally, by holding down both buttons for more the 5 seconds the lamp can turn-off, or in other words, brought back to the Off-State.

Final Results

Overall, all of the functionalities targeted during the proposal stage of the project were successfully met. The results met in this project were as followed:

1. 3D printed body lamp - completed
2. Weather API implementation- completed
3. Lamp Software functionalities implementation - completed
4. Touch buttons implementation - completed
5. LCD display implementation - completed
6. LEDs implementation - completed.

The only function that was not implemented was the expansion/optional functionalities of speakers playing pre-recorded weather sounds resembling that of the weather condition such as rain or wind. Since all the promised functionality were implemented successfully our evaluation of the prototype was pretty straight forward. The evaluation consisted in the exercising of all the functionalities mentioned and show that it produced a positive response and behavior as expected.

Reflection / Discussion

We came across many different challenges during the progress of this project, but here we highlight the ones we consider the most important.
Start early- A good lessons learned was to start 3D printing as soon as possible. It takes a few trials to learn how the 3D printer behaves and the level of accuracy in provides in the models printed. We ran into many lamp body issues in terms of design and dimension causing to have to reprint a few models. It is a time consuming task, and sometimes creating the 3D models in sketchup could turn out to be a little tricky to perfect. The same lesson learn can be applied to programming. Programing for Microcontrollers like Arduino devices could be very challenging and need to be done in a very careful manner. For instance, our lamp gets weather updates every 5 minutes. Introducing a simple 5 minutes delay in the code seems like a reasonable approach but unfortunately this would also cause other tasks down the pipeline to get delayed as well. This kind of differences between normal programing and programing for

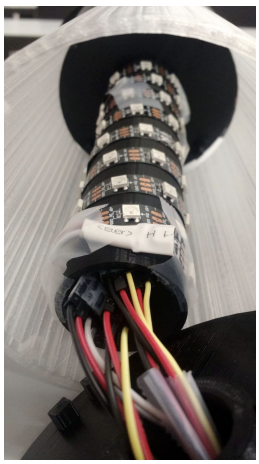
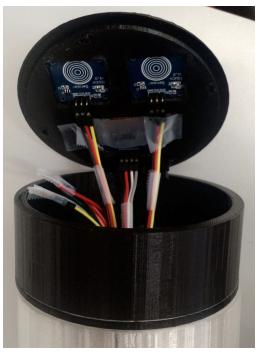
Microcontrollers was relatively new to us and caused many issues when combining the code together due to the sequential nature of the different task that need to occur. This unforeseen problem caused us to have to modify a big part of our code in order to achieve the final desire result.

Another lessons learned was during the implementation of the LEDs. It is important to be aware that the number of LEDs used will affect directly the power consumption of the device. It is extremely important to be aware of the this and whether or not the controller is able to keep up with this demand. This will also affect the tuning of the RGBW values used to achieve the different desired colors in the lamp.

Finally, we strongly believe that one of the primary reasons that lead to the success of our project was having a clear picture of the project and being able to identify and break it down into subtask. This allowed the team to work together in parallel with each person focusing on a particular subtask. As described earlier in the final results section, there we show the list of subtasks. As an example, while one team member focused on 3D printing of the model, another person focused on the API part of the problem and so forth. Although, each of us concentrated on a certain area more than other ones, we always kept an oversight of each other and learned from each others work. Thus, avoiding the risk that only one person becomes expert in a certain area and limiting progress if that group member is not able to constantly be there during the course of the project.

Appendix

Prototype Pictures

Source Code *(Implementation)*

We coded the logic of this project using C/C++ programming language and was compiled using the Arduino IDE to machine executable code. The code was then uploaded to NodeMCU version of the Arduino devices using a USB cable.

The final version of the code is commented and highlighted at the following.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>
#include <Adafruit_NeoPixel.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <WorldClockClient.h>
//#include <Adafruit_SSD1306.h>
#include "SSD1306Wire.h"
#include "OLEDDisplayUi.h"
#include "icons.h"

// All our states in our lamp
#define S_TURNOFF 1
#define S_INITIALIZATION 2
#define S_WORKING 3
#define S_CITY_SELECTION 4
```

```
// All different colors of our LEDs
#define c_c_white 1
#define c_c_blue 2
#define c_c_purple 3
#define c_c_yellow 4

// demo time limit
#define demo_time_limit 5000
#define city_selection_timer_limit 5000

#define Led_pin            D1
#define NUMPIXELS      60
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, Led_pin, NEO_RGB + NEO_KHZ800);

// Initialize the OLED display using brzo_i2c
// D3 -> SDA
// D5 -> SCL

// Initialize the OLED display using Wire library
SSD1306Wire  display(0x3c, D3, D5);
OLEDDisplayUi ui     ( &display );



int led_light_up_number = NUMPIXELS; //- 20;

int TouchSensor1 = D2; //connected to Digital pin D2
int TouchSensor2 = D6; //connected to Digital pin D6
float brightness = 255; // Initialize Brightness
float fadeAmount = 25.5;
boolean booting_up = true;

// WiFi credentials
const char* ssid = "UC_PCOMP";
const char* password =  "pr3pAr3d";

// api credentials
const String api_url = "http://api.openweathermap.org/data/2.5/weather?&units=metric";
const String api_token = "&appid=134f461a6f03f55040a08b8935cd2f85";

String cities_name [] = {"Lugano, Switzerland", "Beijing, China", "Moscow, Russia", "Quito,
Ecuador", "Tehran, Iran"};
String cities_weather_api [] = {"&q=Lugano,CH", "&q=Beijing,CN", "&q=Moscow,RU",
"&q=Quito,EC", "&q=Tehran,IR"};

// time library configuration
String cities_time [] = {"Europe/Zurich", "Asia/Shanghai", "Europe/Moscow",
"America/Guayaquil", "Asia/Tehran"};
WorldClockClient worldClockClient("en", "CH", "E, dd. MMMMM yyyy", 4, cities_time);
int current_location_index = 0;


// weather informations
String current_city_name = "";
String current_time = "";
```

```
String current_temperature = "";
String current_humidity = "";
String current_weather_desc = "";
String current_net_stat = "";
String current_api_stat = "";
String current_weather_id = "";
int current_led_color = 0;

static int state = S_TURNOFF; // initial state is (S_TURNOFF = 1) which means the "off" state.
static unsigned long start_timer;  // To store the "current" time for delays.
static unsigned long thunderstorm_timer; // timer for switching color of the LED for the
thunderstorm
static unsigned long demo_timer;
static unsigned long city_selection_timer;

int delayval = 20;// Delay for a period of time (in milliseconds).

void off_all_leds(){
  for ( int led_num = 0; led_num < led_light_up_number; led_num++ ) {
    pixels.setPixelColor(led_num, pixels.Color(0, 0, 0));
  }
  pixels.show(); // This sends the updated pixel color to the hardware.
  delay(delayval); // Delay for a period of time (in milliseconds).
}

void bootup_effect(){
  int color_mode = 1;
  for ( int led_num = 0; led_num < led_light_up_number; led_num++ ) {
    if (led_num % 20 == 0) {
      if (color_mode == 1) {
        color_mode = 2;
      } else if (color_mode == 2) {
        color_mode = 3;
      } else if (color_mode == 3) {
        color_mode = 1;
      }
    }

    // set the color of leds
    if (color_mode == 1) {
      pixels.setPixelColor(led_num, pixels.Color((int)brightness, 0, 0));
    } else if (color_mode == 2) {
      pixels.setPixelColor(led_num, pixels.Color(0, (int)brightness , 0));
    } else if (color_mode == 3) {
      pixels.setPixelColor(led_num, pixels.Color(0, 0 , (int)brightness));
    }

    pixels.show(); // This sends the updated pixel color to the hardware.
    delay(delayval); // Delay for a period of time (in milliseconds).
  }
  Serial.println("starting color motion");
}

void increase_light_intensity(int sensor1) {
  //INCREASES LED intensity
  if (sensor1 == HIGH) {
    delay(800);
```

```
    while (digitalRead(TouchSensor1) == HIGH && digitalRead(TouchSensor2) != HIGH) {
      if (brightness <= 255 - fadeAmount) {

        // LED affect
        brightness = fadeAmount + brightness;
        pixels.setBrightness((int)brightness);
        pixels.show(); // This sends the updated pixel color to the hardware.
        delay(delayval);

        Serial.println("brightness");
        Serial.println((int)brightness);
      }
      display.clear();
      // LCD intensity display
      // draw the progress bar
      display.drawProgressBar(0, 32, 120, 10, brightness/2.55);

      // draw the percentage as String
      display.setTextAlignment(TEXT_ALIGN_CENTER);
      display.drawString(64, 15, String(lround(brightness/2.55)) + "%");
      display.display();
      delay(800);
      display.clear();

      start_timer = millis()+59000;
    }
  }
}

void decrease_light_intensity(boolean sensor2) {
  //DECREASES LED intensity
  if (sensor2 == HIGH) {
    Serial.println("decreasing light");
    delay(800);
    while (digitalRead(TouchSensor2) == HIGH && digitalRead(TouchSensor1) != HIGH) {
      if (brightness > fadeAmount) {
        brightness = brightness - fadeAmount;

        // LED affect
        pixels.setBrightness((int)brightness);
        pixels.show(); // This sends the updated pixel color to the hardware.
        delay(delayval);

        Serial.println("brightness");
        Serial.println((int)brightness);
      }
      display.clear();
      // LCD intensity display
      // draw the progress bar
      display.drawProgressBar(0, 42, 120, 10, brightness/2.55);

      // draw the percentage as String
      display.setTextAlignment(TEXT_ALIGN_CENTER);
      display.drawString(64, 15, "Light Intensity:");
      display.drawString(64, 25, String(lround(brightness/2.55)) + "%");
      display.display();
      delay(800);
```

```
      display.clear();

      start_timer = millis()+59000;
    }
  }
}

// LCD welcome Screen
void greeting_sc(){
  display.clear();
  // draw circle for showing for greeting on the screen
  drawCircle();
  delay(2000);
  display.clear();
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.setFont(ArialMT_Plain_10);
  display.drawString(64, 32, "WELCOME!");
  display.display();
  delay(2000);
}

// LCD goodbye Screen
void goodbye_sc(){
  display.clear();
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.setFont(ArialMT_Plain_10);
  display.drawString(64, 32, "Goodbye For Now!");
  display.display();
  delay(2000);
  display.clear();
}

// LCD system is off
void system_off_sc(){
  display.clear();
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.setFont(ArialMT_Plain_10);
  display.drawString(64, 6, "SYSTEM OFF");
  display.drawStringMaxWidth(64, 32, 128,"Press both Buttons to Initiate");
  display.drawString(64, 40, "-\n- -\n-  -\n-   -\n-    -");
  display.display();
}


void drawCircle(void) {
  for (int16_t i=0; i<display.getHeight(); i+=2) {
    display.drawCircle(display.getWidth()/2, display.getHeight()/2, i);
    display.display();
    delay(10);
  }
  delay(1000);
  display.clear();
}



bool check_api_stat() {
```

```
  HTTPClient http;
  http.begin(api_url+api_token+cities_weather_api[current_location_index]); //Specify the URL
  int httpCode = http.GET();  //Make the request
  if (httpCode > 0) { //Check for the returning code
    http.end(); //Free the resources
    return true;
  } else {
    http.end(); //Free the resources
    return false;
  }


}

void get_weather_info() {

  HTTPClient http;
  http.begin(api_url+api_token+cities_weather_api[current_location_index]); //Specify the URL
  int httpCode = http.GET();  //Make the request

  if (httpCode > 0) { //Check for the returning code
      String json = http.getString();

      Serial.println(json);
      DynamicJsonBuffer jsonBuffer;
      JsonObject& root = jsonBuffer.parseObject(json);

      if (!root.success()){
        display.clear();
        display.setTextAlignment(TEXT_ALIGN_CENTER);
        display.setFont(ArialMT_Plain_10);
        display.drawStringMaxWidth(64, 32, 128, "API response ParseObject() failed!!!");
        display.display();
        delay(2000);
        Serial.print("API response ParseObject() failed!!!");
        return;
      }
      JsonObject& weather = root["weather"][0];
      current_weather_id = weather["id"].as<String>();
//      current_weather_id = "200";
      current_city_name = root["name"].as<String>();
      double temperature = root["main"]["temp"];
      current_temperature = String(lround(temperature));
      current_humidity = root["main"]["humidity"].as<String>();
      current_weather_desc = weather["description"].as<String>();

      // show weather on LCD
      show_weather_on_LCD();

      // show weather on LED
      show_weather_on_LED();

  } else {
    display.clear();
    display.setTextAlignment(TEXT_ALIGN_CENTER);
    display.setFont(ArialMT_Plain_10);
    display.drawStringMaxWidth(64, 32, 128, "Error on HTTP request");
```

```
    display.display();
    delay(2000);
    Serial.println("Error on HTTP request");
  }

  http.end(); //Free the resources
}

void show_weather_on_LCD(){
  String line1 = "CITY: "+ current_city_name;
  worldClockClient.updateTime();
  String line2 = worldClockClient.getHours(current_location_index) + ":" +
worldClockClient.getMinutes(current_location_index);
  String line3 = current_temperature+"  C";
  String line4 = "HUMIDITY: "+ current_humidity +"%";
  String line5 = "DESCRIPTION: "+current_weather_desc;
  String line6 = "--------------------";
  String line7 = "Network status: Connected";
  String line8 = "API status: Connected";
  display.clear();
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0,
line1+"\n"+line2+"\n"+line3+"\n"+line4+"\n"+line5+"\n"+line6+"\n"+line7+"\n"+line8+"\n");
  display.display();
  // pring celcius character
  display.drawCircle(15, 30, 1);
  display.display();
  delay(2000);
}


void show_weather_on_LED()
{
  /*
   * Color mapping (API weather code ---> LED color)
   *
   * thunder= 2XX ---> blue<->white
   * Rain/drizzle = 5XX,3XX ---> Blue
   * Snow = 6XX ---> White
   * Clear = 800 --> Yellow
   * clouds/atmosphere = 7, 80X ---> purple
  */
  char code = current_weather_id.charAt(0);
  switch(code){
    case '2':
      Serial.println("LED color is: blue<->white blinking");
      current_led_color = c_c_blue;
      thunderstorm_timer = millis();  // Remember the current time
      leds_color_change(0, 0, 255);
      break;
    case '3':
    case '5':
      Serial.println("LED color is: Blue");
      current_led_color = c_c_blue;
      leds_color_change(0, 0, 255);
      break;
```

```
    case '6':
      Serial.println("LED color is: White");
      current_led_color = c_c_white;
      leds_color_change(100,140,140);
      break;
    case '7':
    case '8':
      if(current_weather_id.equals("800")){
        Serial.println("LED color is: Yellow");
        current_led_color = c_c_yellow;
        leds_color_change(140, 140, 0);
      } else{
        Serial.println("LED color is: purple");
        current_led_color = c_c_purple;
        leds_color_change(30,60,150);
      }
      break;
  }
}


//change the color of all leds
void leds_color_change(int r, int g, int b){
  for ( int led_num = 0; led_num < led_light_up_number; led_num++ ) {
    pixels.setPixelColor(led_num, pixels.Color(r, g, b));
  }
  pixels.show(); // This sends the updated pixel color to the hardware.
  delay(delayval); // Delay for a period of time (in milliseconds).
}

//Select color to display, or demo of all settings (TBD)
void LCD_text_display(String text){
    display.clear();
    display.setTextAlignment(TEXT_ALIGN_CENTER);
    display.setFont(ArialMT_Plain_10);
    display.drawString(64, 32, text);
    display.display();
    display.clear();
}

// check demo intruption
bool check_demo_intruption(){
  if (digitalRead(TouchSensor1) == HIGH || digitalRead(TouchSensor2) == HIGH) {
    return true;
  }else {
    return false;
  }
}


// demo main function
void Demo_mode(){
  //LCD_text_display(text, delay)
  LCD_text_display("Initializing Lamp Demo");
  delay(2000);

  //Type of Weather displays
```

```
LCD_text_display("Weather Conditions Demo");
delay(2000);

//Clear
LCD_text_display("CLEAR SKY");
leds_color_change(140, 140, 0);
delay(2000);

//Clouds
LCD_text_display("CLOUDY");
leds_color_change(47,79,79);
delay(2000);

//ThurderStorm
LCD_text_display("THUNDERSTORM");
leds_color_change(0, 0, 255);
for( int i = 0; i < 4; i++ ) {
  // blinking effect for thunderstorm
  delay(2000);
  // change the color to white
  leds_color_change(100,140,140);
  delay(100);
  leds_color_change(0, 0, 255);
}

//Rain/drizzle
LCD_text_display("RAIN/DRIZZLE");
leds_color_change(0, 0, 255);
delay(2000);

//Snow
LCD_text_display("SNOWY");
leds_color_change(100,140,140);
delay(2000);

//End of Weather Demo
LCD_text_display("End of Weather Demo");
leds_color_change(0,0,0); //turns LEDs off
delay(2000);

//LED Intensity decrease/increase
LCD_text_display("LED Intensity Demo");

// change the color to white
leds_color_change(100,140,140);
LCD_text_display("LED Intensity Decrease");
delay(2000);

float fade=25.5;
float how_bright=255;
while (how_bright>fade  ) {
  // LED affect
  how_bright =  how_bright - fade;
  pixels.setBrightness((int)how_bright);
  pixels.show(); // This sends the updated pixel color to the hardware.
  delay(delayval);
```

```
    // LCD affect
    display.clear();

    // LCD intensity display
    // draw the progress bar
    display.drawProgressBar(0, 32, 120, 10, how_bright/2.55);

    // draw the percentage as String
    display.setTextAlignment(TEXT_ALIGN_CENTER);
    display.drawString(64, 15, String(lround(how_bright/2.55)) + "%");
    display.display();
    delay(800);
    display.clear();
  }

  LCD_text_display("Min intensity reached!");
  delay(2000);

  while (how_bright<255  ) {
    // LED affect
    how_bright =  how_bright + fade;
    pixels.setBrightness((int)how_bright);
    pixels.show(); // This sends the updated pixel color to the hardware.
    delay(delayval);

    // LCD affect
    display.clear();

    // LCD intensity display
    // draw the progress bar
    display.drawProgressBar(0, 32, 120, 10, how_bright/2.55);

    // draw the percentage as String
    display.setTextAlignment(TEXT_ALIGN_CENTER);
    display.drawString(64, 15, String(lround(how_bright/2.55)) + "%");
    display.display();
    delay(800);
    display.clear();
  }

  LCD_text_display("Max intensity reached!");
  delay(2000);

  //End of LED intensity Demo
  LCD_text_display("End of LED intensity Demo");
  delay(2000);
  LCD_text_display("End of Lamp Demo");
  delay(2000);

  //TURNING SYSTEM OFF
  off_all_leds(); // Turn off all LEDs
  goodbye_sc();    // LCD turn off
  system_off_sc();  // display system is off on LCD
  state = S_TURNOFF;  //change state to OFF_STATE
}

// show all available cities on the LCD
```

```
void show_all_available_cities(){
  display.clear();
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  for( int i_cities = 0; i_cities < 5; i_cities++ ) {
     display.drawString(0, 0 + (i_cities*12), cities_name[i_cities] + "     " + ((i_cities ==
current_location_index) ? "<<<" : "") );
  }
  display.display();
}

// select next city from the list of all available city for taking time and weather data from
Internet (API)
void select_next_city(int sensor1){
  if (sensor1 == HIGH) {
    delay(300);
    while (digitalRead(TouchSensor1) == HIGH && digitalRead(TouchSensor2) != HIGH) {
      if (current_location_index == 4) {
        current_location_index = 0;
      } else{
        current_location_index++;
      }
      show_all_available_cities();
      delay(800);
    }
  }
}

// select previous city from the list of all available city for taking time and weather data
from Internet (API)
void select_previous_city(int sensor2){
  if (sensor2 == HIGH) {
    delay(300);
    while (digitalRead(TouchSensor2) == HIGH && digitalRead(TouchSensor1) != HIGH) {
      if (current_location_index == 0) {
        current_location_index = 4;
      } else{
        current_location_index--;
      }
      show_all_available_cities();
      delay(800);
    }
  }
}


void setup() {
  Serial.begin(9600);
  pinMode(TouchSensor1, INPUT);
  pinMode(TouchSensor2, INPUT);
  pinMode(Led_pin, OUTPUT);
  //  initialize LED
  pixels.begin(); // This initializes the NeoPixel library.

  // Turn off all LEDs
  off_all_leds();
```

```
  //   initialize LCD
  // Initialising the UI will init the display too.
  display.init();

  display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);

   // display system is off on LCD
  system_off_sc();
  Serial.println("end of setup part");

}

void loop() {
  switch(state)
  {
    case S_TURNOFF:
      {
        if (digitalRead(TouchSensor1) == HIGH && digitalRead(TouchSensor2) == HIGH) {
          //Change of state
          demo_timer=millis();
          state = S_INITIALIZATION;
          while (digitalRead(TouchSensor1) == HIGH || digitalRead(TouchSensor2) == HIGH) {
            delay(10);
            if (millis()-demo_timer >= demo_time_limit){
              Demo_mode();
              state = S_TURNOFF;
            }
          }
        }
        break;
      }
    case S_INITIALIZATION:
      {
        start_timer = millis();  // Remember the current time

        // LED effect
        brightness = 255;
        bootup_effect();
        delay(delayval); // Delay for a period of time (in milliseconds).

        // LCD welcome message
        greeting_sc();
        display.clear();

        delay(delayval); // Delay for a period of time (in milliseconds).

        // connect through WiFi
        WiFi.begin(ssid, password);
        int counter = 0;
        while (WiFi.status() != WL_CONNECTED) {
          delay(500);
          Serial.print(".");
          display.clear();
          display.drawString(64, 10, "Connecting to WiFi");
          display.drawXbm(46, 30, 8, 8, counter % 3 == 0 ? activeSymbol : inactiveSymbol);
```

```
        display.drawXbm(60, 30, 8, 8, counter % 3 == 1 ? activeSymbol : inactiveSymbol);
        display.drawXbm(74, 30, 8, 8, counter % 3 == 2 ? activeSymbol : inactiveSymbol);
        display.display();
        counter++;
      }

      // show success message on screen
      display.clear();
      display.setTextAlignment(TEXT_ALIGN_CENTER);
      display.setFont(ArialMT_Plain_10);
      display.drawStringMaxWidth(64, 23, 128, "WiFi Connected Successfully!");
      display.display();
      Serial.println("WiFi Connected Successfully!");
      delay(2000);

      if(check_api_stat()){
        // show success message on screen
        display.clear();
        display.setTextAlignment(TEXT_ALIGN_CENTER);
        display.setFont(ArialMT_Plain_10);
        display.drawStringMaxWidth(64, 23, 128, "API Connected Successfully!");
        display.display();
        Serial.println("API Connected Successfully!");
        delay(2000);
      }else{
        // show success message on screen
        display.clear();
        display.setTextAlignment(TEXT_ALIGN_CENTER);
        display.setFont(ArialMT_Plain_10);
        display.drawString(64, 32, "<<<API Connection Error>>>");
        display.display();
        Serial.println("<<<API Connection Error>>>");
        delay(2000);
      }

      // get current time from wifi
      worldClockClient.updateTime();
      Serial.println("Current time is: " + worldClockClient.getHours(current_location_index)
+ ":" + worldClockClient.getMinutes(current_location_index));

      get_weather_info();
      state = S_WORKING;
      break;
    }
  case S_WORKING:
    {
      // blinking effect for thunderstorm
      if( current_weather_id.charAt(0) == '2'){
        if( (millis() - thunderstorm_timer) > 2000){
          thunderstorm_timer = millis();
          if(current_led_color == c_c_blue){
            // change the color to white
            leds_color_change(100,140,140);
            current_led_color = c_c_white;
            thunderstorm_timer = millis()+ 1000;
          } else{
            // change the color to blue
```

```
            leds_color_change(0, 0, 255);
            current_led_color = c_c_blue;
          }
        }
      }

      if( (millis() - start_timer) > 60000){
        start_timer = millis();
        get_weather_info();
      }

      //Detects double button pressed for TURNING OFF the lamp or Switch to City Selection
Mode
      if (digitalRead(TouchSensor1) == HIGH && digitalRead(TouchSensor2) == HIGH) {
        //Change the state to whether to TURN_OFF state or CITY_SELECTION_STATE
        city_selection_timer=millis();

        while (digitalRead(TouchSensor1) == HIGH || digitalRead(TouchSensor2) == HIGH) {
          delay(10);
        }
        if (millis()-city_selection_timer >= city_selection_timer_limit){
          // Turn off all LEDs
          off_all_leds();

          // LCD turn off
          goodbye_sc();

          // display system is off on LCD
          system_off_sc();

          state = S_TURNOFF;
          break;
        } else{
          state = S_CITY_SELECTION;

          // show all cities on LCD
          show_all_available_cities();
          break;
        }
      }
      increase_light_intensity(digitalRead(TouchSensor1));
      decrease_light_intensity(digitalRead(TouchSensor2));
      break;
    }
  case S_CITY_SELECTION:
    {
      // listen to touchsensor1 for switching between cities
      select_next_city(digitalRead(TouchSensor1));

      // listen to touchsensor2 for switching between cities
      select_previous_city(digitalRead(TouchSensor2));

      // listen to both touchsensors for swtiching back to working state
      if (digitalRead(TouchSensor1) == HIGH && digitalRead(TouchSensor2) == HIGH) {
        while (digitalRead(TouchSensor1) == HIGH || digitalRead(TouchSensor2) == HIGH) {
          delay(10);
        }
```

```
          get_weather_info();
          state = S_WORKING;
        }
        break;
      }
    default:
      {
        state = S_TURNOFF;
        break;
      }
  }
}
```

References

1. NodeMCU ESP8266 wifi (Microcontroller)
   1.1. https://www.roboshala.com/nodemcu-pinout

2. WS2812B LED Strip (60-LED Strip)
   2.1. https://arduino.stackexchange.com/questions/32017/esp8266-nodemcu-first-ws2812-led-lights-up-green
   2.2. https://howtomechatronics.com/tutorials/arduino/how-to-control-ws2812b-individually-addressable-leds-using-arduino/

3. OLED display (SSD1306)
   3.1. http://wiki.sunfounder.cc/index.php?title=OLED-SSD1306_Module

4. STTP223B IC capacitive sensors (Touch Sensor)
   4.1. http://www.theorycircuit.com/digital-capacitive-touch-sensor-arduino-interface/

5. GitHub repository of the project with all codes and reports and also the breadboard schematic created with Fritzing.
   5.1. https://github.com/AmirDavoodi/SimulatorWeatherLamp