



Tecnológico De Estudios Superiores De San Felipe Del Progreso

Ingeniería Informática

Tecnologías e interfaces de computadoras

Manual

PRACTICAS #1,#2,#3,#4

Docente:

Luis Ángel González Flores

Integrantes:

Carlos Omar Hernández Hernández

Alexis Adolfo González Antonio

Ángel Uriel Mateo De Jesús

Ismael Antonio González

Diego Ramírez Moreno

Saul Cortes Martínez

Quinto Semestre

502

Contenido

Introducción	4
Practica #1	5
 PRACTICA #2	14
PRACTICA #3	24
PRACTICA #4	24
Conclusión	31
 Ilustración 1 PRACTICA#1 _ Prototipo	5
Ilustración 2 conexiones de cables.....	6
Ilustración 3 conexión de los componentes a Arduino	6
Ilustración 4 Arduino conectado	7
Ilustración 5 Prototipo y Arduino en conjunto.....	7
Ilustración 6 sensores y leds conectados a la protoboard	8
Ilustración 7 Función del contacto al Gas LP	9
Ilustración 8 el encendedor funciona como el gas e humo	10
Ilustración 9 funcionamiento de los 3 componentes	11
Ilustración 10 Desarrollo de código.....	12
Ilustración 11 Segunda parte del código	12
Ilustración 12 El grafico donde se aloja los resultados.....	13
Ilustración 13 PRACTICA#2 _Estructura de prototipo.....	15
Ilustración 14 Funcionamiento del circuito.....	15
Ilustración 15 Gráfica de voltaje.....	18
Ilustración 16 Grafica de voltaje y corriente	18
Ilustración 17 Panel HTML -1.....	19
Ilustración 18 Panel HTML-2	20
Ilustración 19 Controlador.....	21
Ilustración 20 Panel de DATOS.....	21
Ilustración 21 Panel del SERVICE-1.....	22
Ilustración 22 Panel del SERVICE-2	23
Ilustración 23 Panel del SERVICE-3	23
Ilustración 24 PRACTICA#4 _Prototipo	24
Ilustración 25 Estructura completa	25
Ilustración 26 CONTROL HTML.....	25
Ilustración 27 CONTROL HTML	26
Ilustración 28 PAGE_CONTROLLER	26
Ilustración 29 Led_Controller-1.....	27
Ilustración 30 Led_Controller-2.....	27
Ilustración 31 Código de Arduino	28
Ilustración 32 Simulación	29



Ilustración 33 Led apagado.....	29
Ilustración 34 Led encendido	30
Ilustración 35 Funcionamiento desde otro dispositivo	30



Introducción

Lo que se abordara en esta práctica será el deslazamiento y la integridad de cada elemento utilizar y saber los parámetros que están colocándose de manera concurrente en un sin número de protocolos que se estarán generando dentro de un solo prototipo que en simultáneamente se configura de como calcula una fuga de Gas LP, HUMO O EL C2, con gran exactitud las problemáticas serían las cuales podríamos estar expuestos a tanta contaminación de estos elementos. sin embargo, esta práctica nos ayudara a comprender los sensores de gas y su funcionamiento para futuras practicas con ello se colocará en un solo procedimiento y se graficara en un solo sistema web a partir de visual estudio.

Practica #1

Materiales:

Cables DuPont

Sensor bh1750

Auduino uno

Leds

Protoboard

Buzzer

Resistencia 3-30

Se crea y se define la estructura que se empleara en la práctica de detección de gas LP, humo y C2

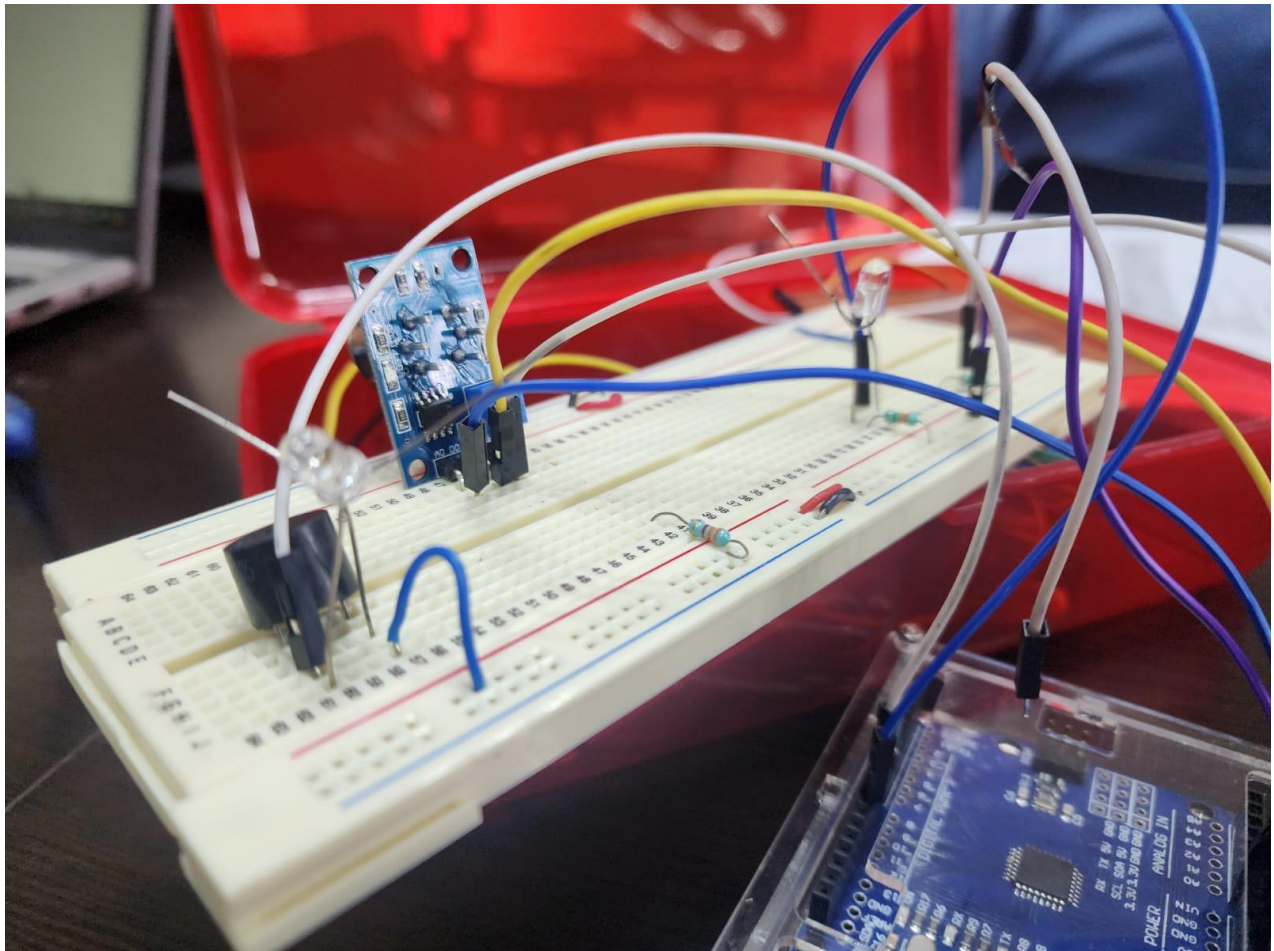


Ilustración 1 PRACTICA#1_ Prototipo



Se poene los siguientes circuitos y componentes en la protoboard.

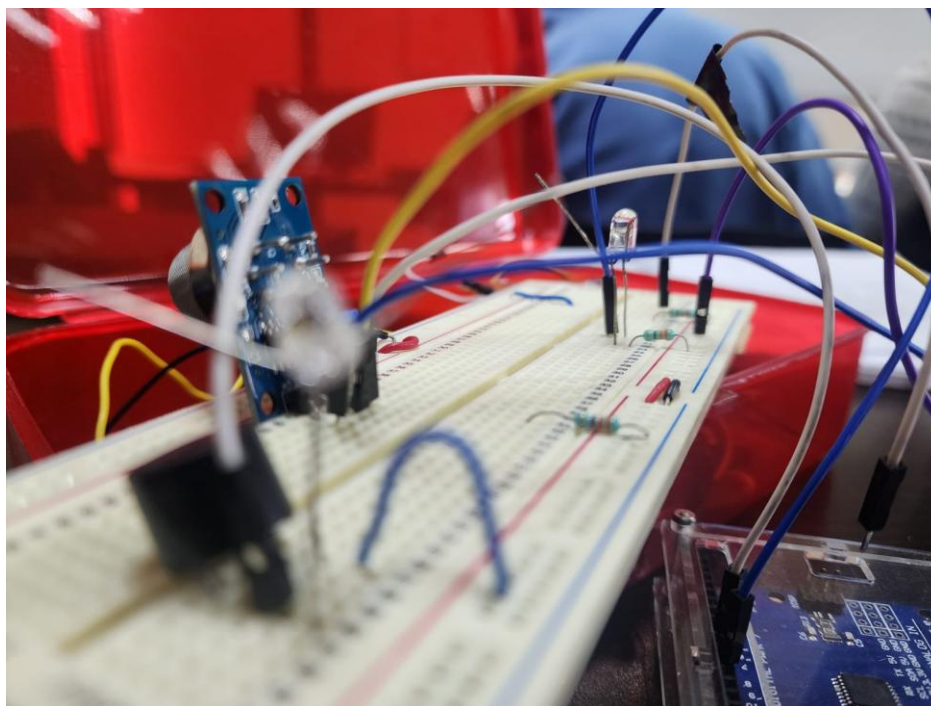


Ilustración 2 conexiones de cables

Por consiguiente, lo conectamos en el Arduino

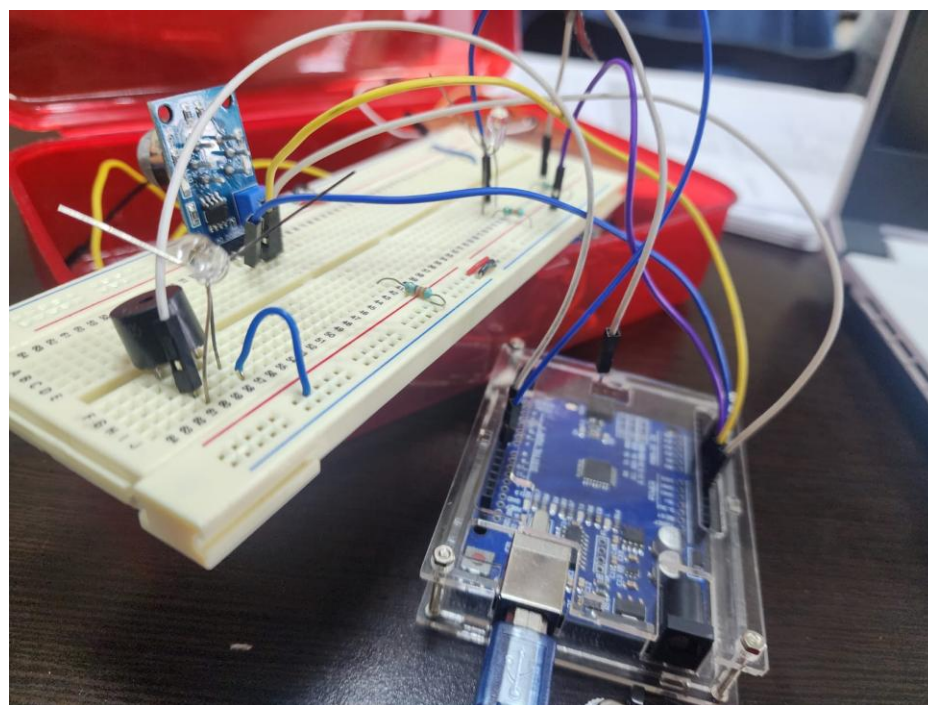


Ilustración 3 conexión de los componentes a Arduino

Este es el arduino que estamos utilizando para poder conectarlo hacia la computadora la cual se podrá crear todo el formato.

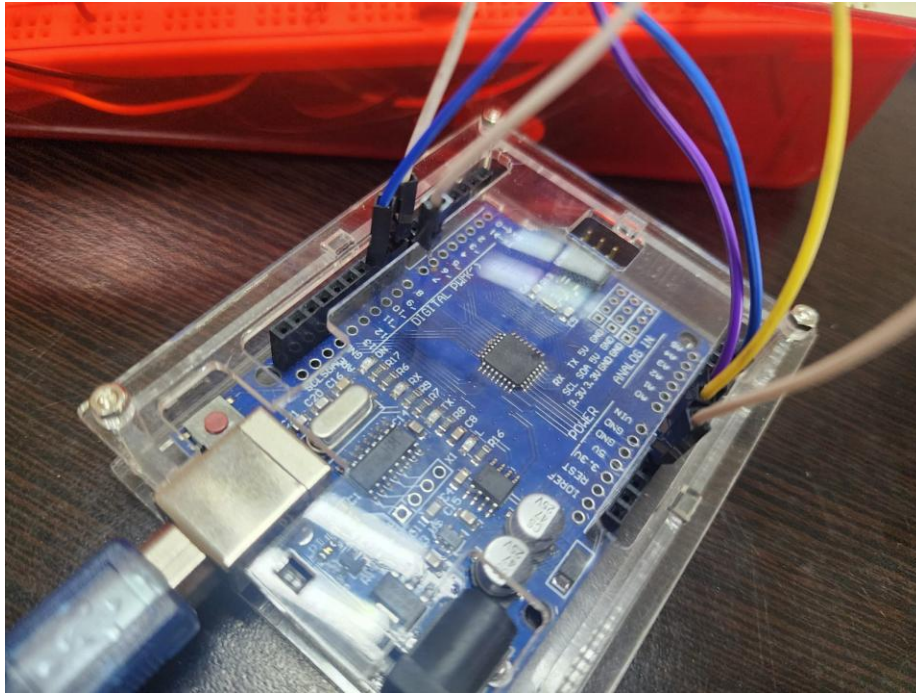


Ilustración 4 Arduino conectado

Este es el prototipo ya especificado dentro de un solo circuito listo para sus pruebas en el Arduino IDE

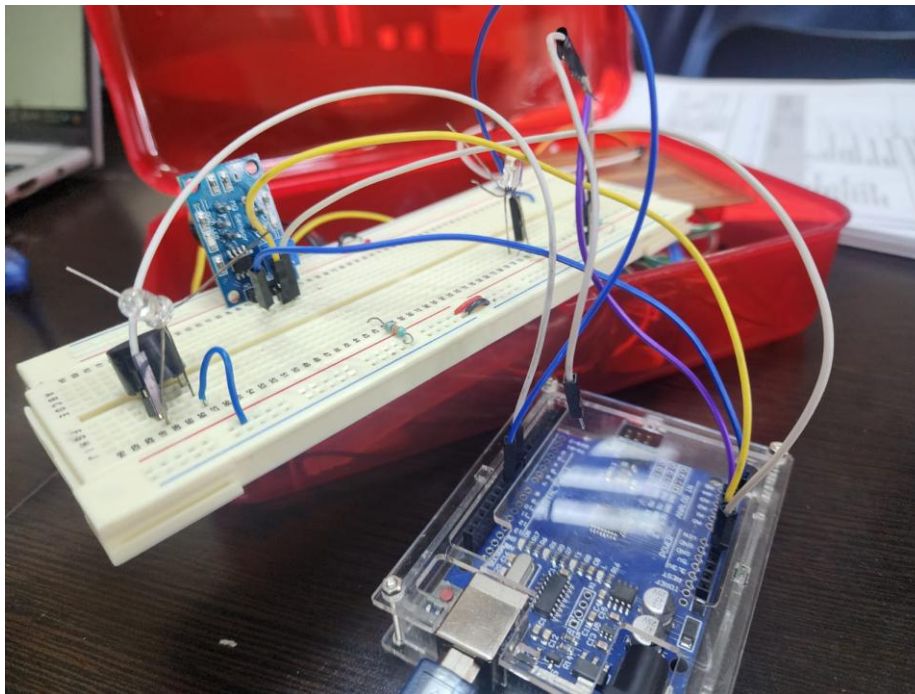


Ilustración 5 Prototipo y Arduino en conjunto

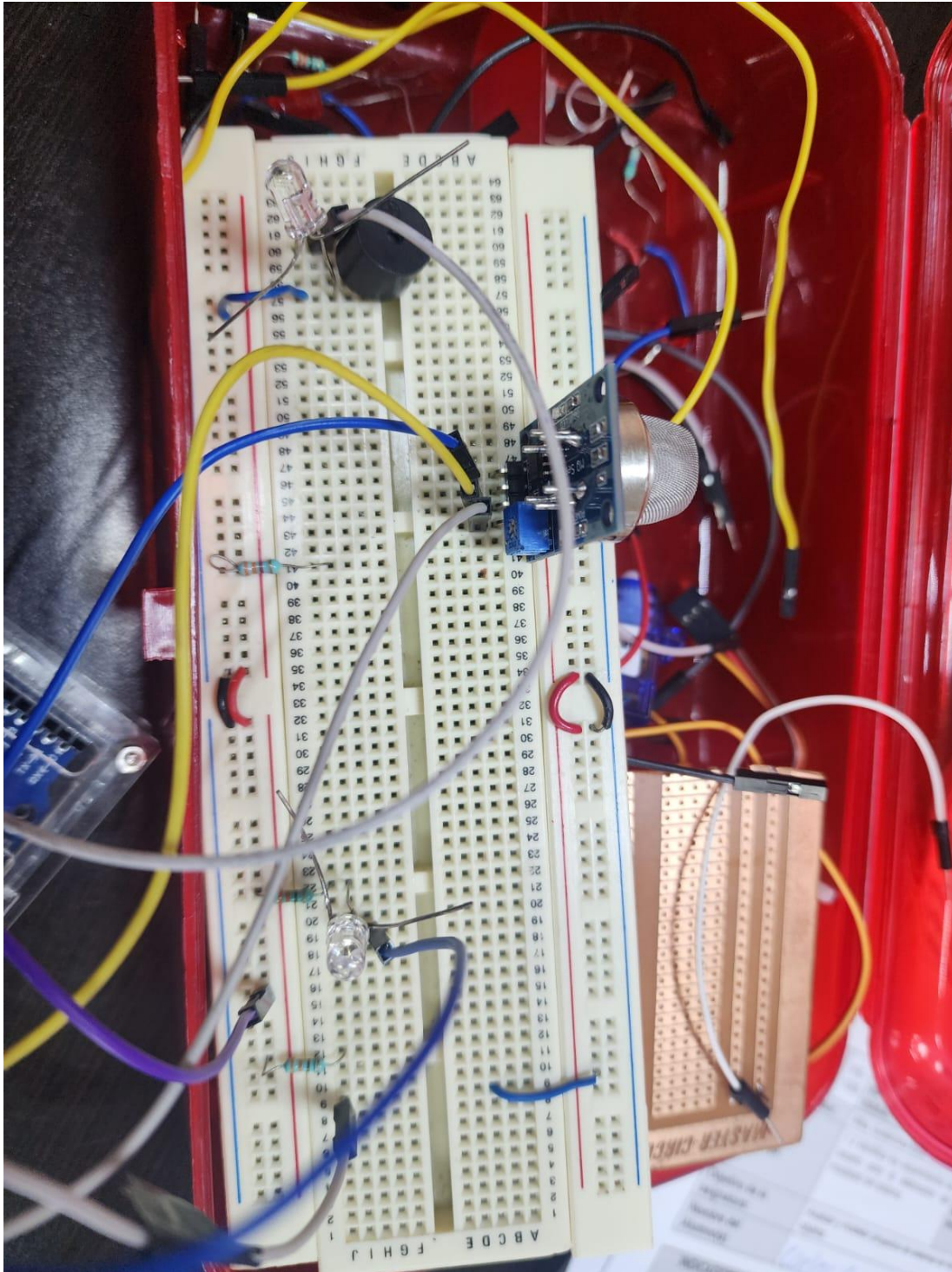


Ilustración 6 sensores y leds conectados a la protoboard



Se ve la funcionalidad del prototipo conectado a la computadora corriendo el prototipo por medio del código desde visual studio

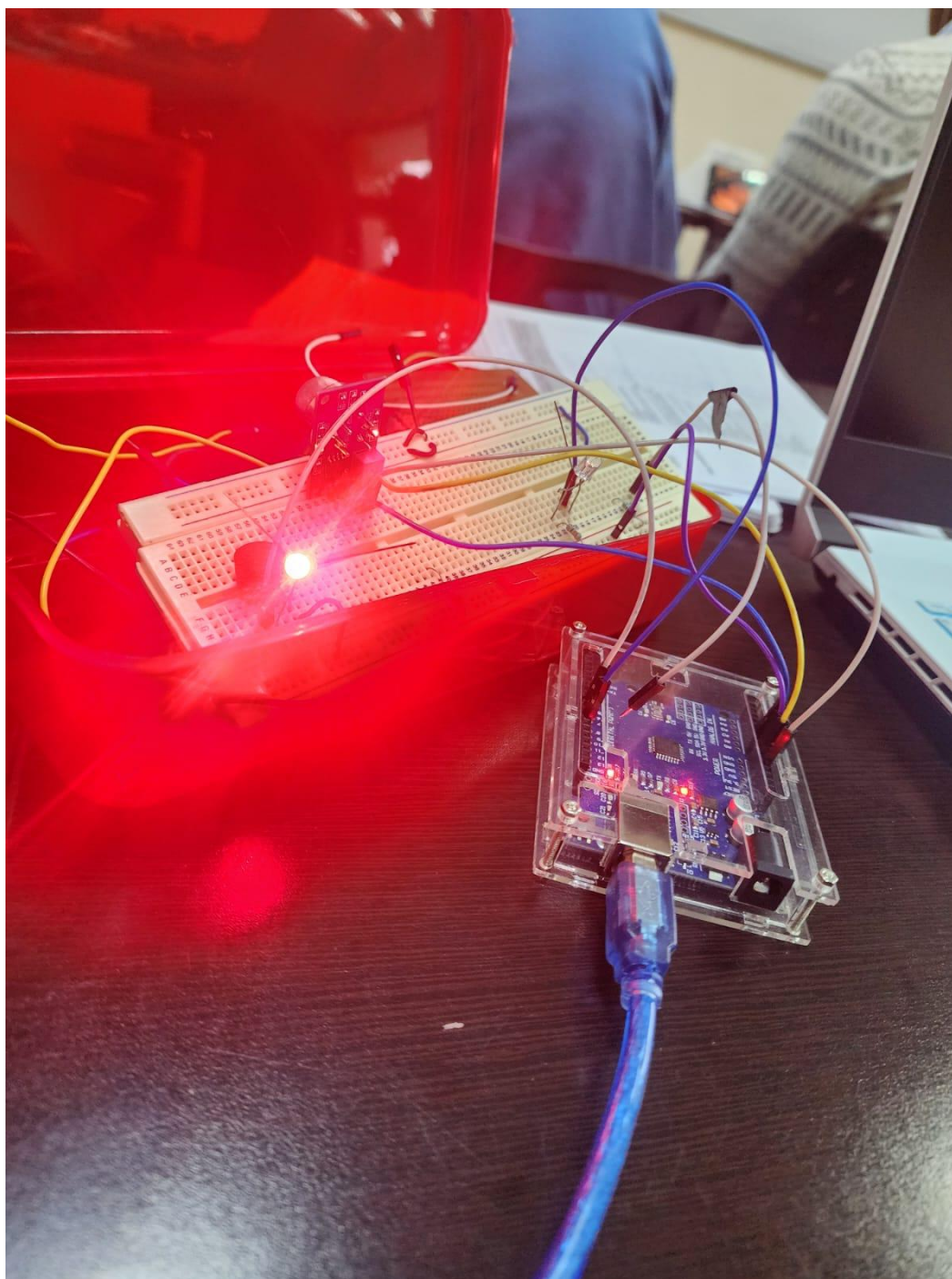


Ilustración 7 Función del contacto al Gas LP

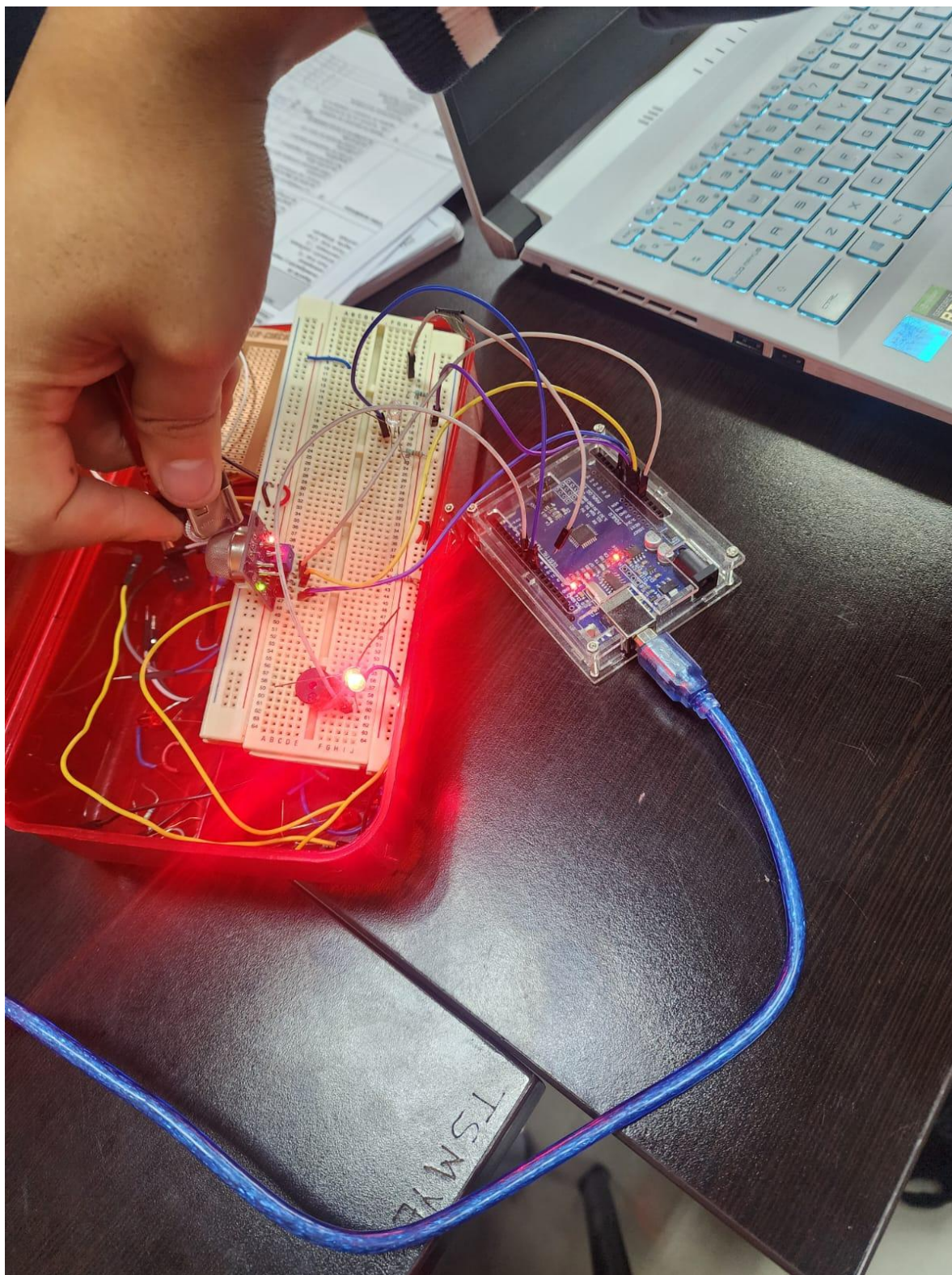


Ilustración 8 el encendedor funciona como el gas e humo



La función de hacer el método de Gas LP al funcionar prende el buzzer y el led, después de que termine el ciclo de transición de los componentes vuelve a su determinación normal.

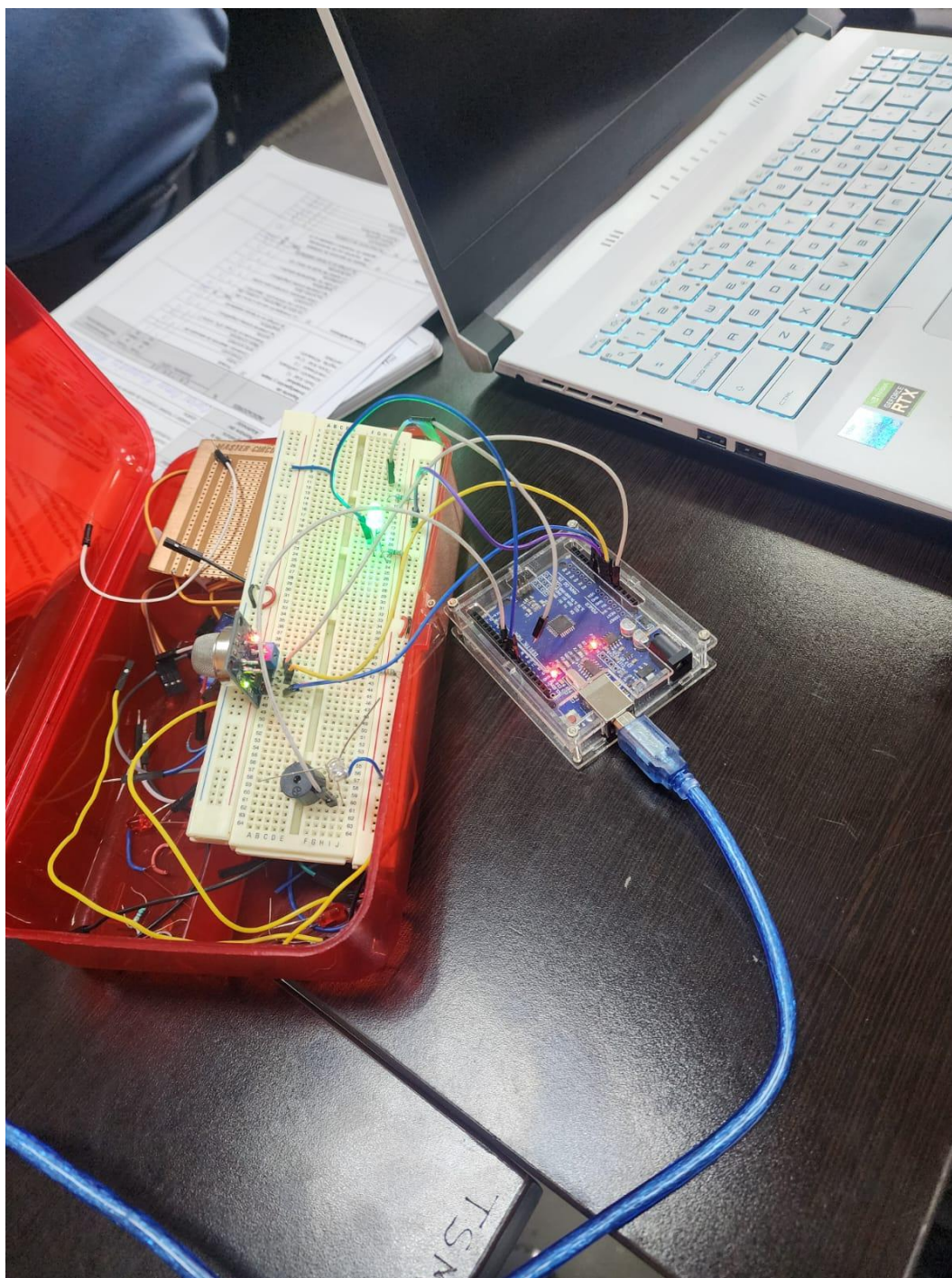


Ilustración 9 funcionamiento de los 3 componentes

Este es el diseño en código desde visual estudio para mandar datos al Arduino y proceder a hacer la función y permitir abrir la pagina web y mandar los resultados gráficos del Gas LP, C2 y humo que se estarás trabajando.

The screenshot displays an Integrated Development Environment (IDE) with a dark theme. On the left, the Explorer panel shows a project structure for 'DEMO'. The main editor area contains the source code for 'SensorController.java'. The code is written in Java and includes package declarations, imports, annotations, and logic for serial communication.

Project Structure (Explorer Panel):

- DEMO
 - .mvn
 - .vscode
 - src
 - main
 - java/com/example
 - demo/controller
 - SensorController.java
 - sauf/demo
 - DemoApplication.java
 - function.java
 - resources
 - static
 - templates
 - Index.html
 - E application.properties
 - test
 - target
 - .gitignore
 - HELP.md
 - mmw
 - mmw.cmd
 - pom.xml

Source Code (SensorController.java):

```
src > main > java > com > example > demo > controller > J SensorController.java ...
1 package com.example.demo.controller;
2
3 import com.fazecast.jSerialComm.SerialPort;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.ResponseBody;
8
9 import java.util.HashMap;
10 import java.util.Map;
11
12 @Controller
13 public class SensorController {
14
15     // Configura el puerto serial para la comunicación con Arduino
16     private SerialPort puertoSerial;
17
18     public SensorController() {
19         // Aquí especificas el puerto COM de Arduino (por ejemplo "COM3" en Windows o "/dev/ttyUSB0" en Linux)
20         puertoSerial = SerialPort.getCommPort(portDescriptors["/dev/cu.usbserial-1410"]); // Cambia este valor por el que
21         puertoSerial.setBaudRate(9600); // Asegúrate que coincida con el baud rate de tu Arduino
22         puertoSerial.openPort();
23     }
24
25     // Lectura de datos desde el puerto serial
26     @GetMapping("/lecturas")
27     @ResponseBody
28     public Map<String, Integer> obtenerLecturas() {
29         Map<String, Integer> lecturas = new HashMap<>();
30
31         if (puertoSerial.isOpen()) {
32             byte[] buffer = new byte[1024];
33             int bytesLeídos = puertoSerial.readBytes(buffer, buffer.length);
34
35             if (bytesLeídos > 0) {
36                 String datos = new String(buffer, offset:0, bytesLeídos);
37                 String[] valores = datos.trim().split(regex,"");
38
39                 if (valores.length >= 2) {
40                     // Parsear los valores recibidos (asumiendo que llegan en el formato: "gas,fuego")
41                     try {
42                         lecturas.put(key:"gas", Integer.parseInt(valores[0]));
43                         lecturas.put(key:"fuego", Integer.parseInt(valores[1]));
44                     } catch (NumberFormatException e) {
```

The status bar at the bottom indicates the current file is 'Lin 65, col 1' and provides information about the workspace and language.

Ilustración 10 Desarrollo de código

The screenshot shows an IDE with a web application project. The Explorer on the left shows the project structure, including a 'demo' folder with 'main', 'resources', and 'static' subfolders. The main editor shows the 'index.html' file, which contains a chart configuration. The chart is titled 'Gráfica de Sensores - Gas LP y Fuego' and displays two data series: 'Sensor de Gas LP' and 'Sensor de Fuego'. The chart is a line chart with a white background and a black border. The x-axis is labeled 'Time' and the y-axis is labeled 'Value'. The 'Sensor de Gas LP' series is a blue line with circular markers, and the 'Sensor de Fuego' series is a red line with circular markers. The chart is displayed in a browser window.

Ilustración 11 Segunda parte del código



Este es el diseño gráfico de los componentes que se están utilizando en esta práctica desde la percepción del Gas LP y el humo, es la parte final de cómo utilizar los sensores.

Lecturas de Gas LP y Fuego



Ilustración 12 El gráfico donde se aloja los resultados



PRACTICA #2

El INA219 es un monitor de corriente de alta precisión que se utiliza para medir el flujo de corriente a través de un resistor de derivación (shunt) en un circuito eléctrico. Este sensor permite realizar mediciones de voltaje y corriente de manera eficiente, y es especialmente útil en aplicaciones de gestión de energía, monitoreo de baterías y sistemas de energía renovable, como paneles solares. Gracias a su interfaz I2C, el INA219 puede ser fácilmente integrado en sistemas basados en microcontroladores como Arduino o ESP8266.

Materiales:

Arduino

Panel solar

Cables DuPont

Protoboard

Sensor BH1750

Sensor INA219

Características:

- Medición de corriente de alta precisión: Permite medir corrientes de hasta 3.2 A con una resolución de 1 mA.
- Medición de voltaje: Capaz de medir voltajes de hasta 26 V.
- Interfaz I2C: Facilita la comunicación con microcontroladores, permitiendo múltiples dispositivos en el mismo bus.
- Configuración de calibración programable: Permite establecer el valor del resistor de derivación para obtener lecturas precisas.
- Registro de datos: Proporciona registros de voltaje, corriente y potencia, que pueden ser leídos a través de su interfaz I2C

Así se ve la estructura del prototipo en la cual se estará viendo en esta practica 2 la cual se estará poniendo en práctica el panel solar conectada al sensor BH1750

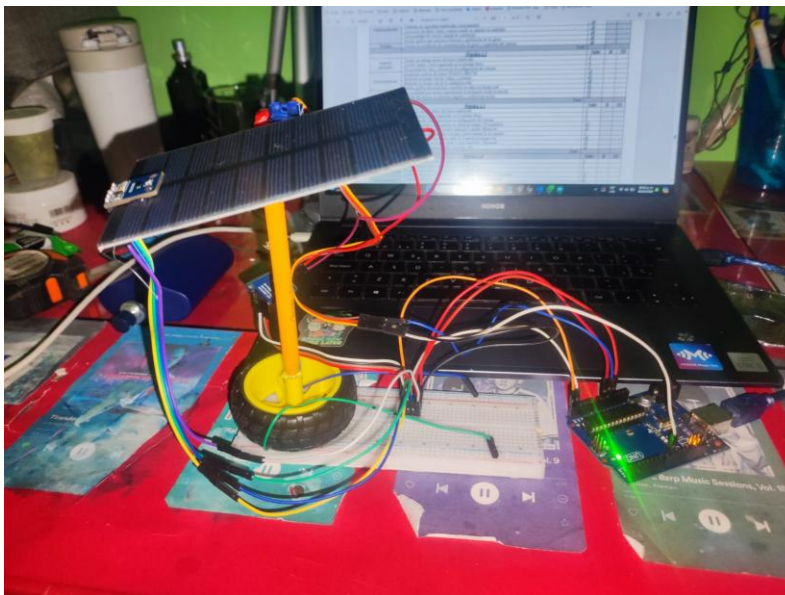


Ilustración 13 PRACTICA#2_Estructura de prototipo

El funcionamiento sobre la luz que manda al panel y manda señal al sensor BH 1750

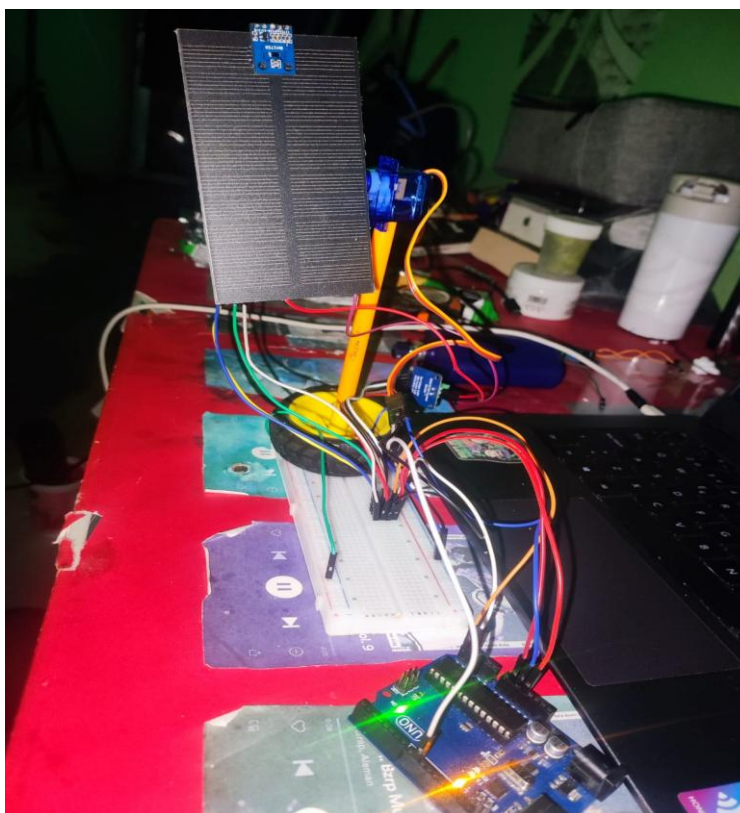


Ilustración 14 Funcionamiento del circuito

A continuación, le presentamos el código de Arduino el cual estaremos empleando en esta practica por medio de Arduino IDE.

Código de Arduino

```
#include <Wire.h>
#include <Adafruit_INA219.h> // Librería para el sensor de corriente y voltaje
INA219
#include <BH1750.h>          // Librería para el sensor de luz BH1750
#include <Servo.h>            // Librería para el servomotor

// Inicializamos los sensores y el servomotor
Adafruit_INA219 ina219;      // Sensor de corriente y voltaje
BH1750 lightMeter;           // Sensor de luz BH1750
Servo myServo;               // Servomotor

// Definimos las variables necesarias
int servoPin = 9;             // Pin donde está conectado el servomotor
int bestAngle = 0;            // Ángulo donde se encontró la mayor intensidad de luz
float maxLux = 0;             // Mayor valor de luz (lux)
int measurementCount = 0;     // Contador de mediciones
int scanCycle = 0;           // Ciclo de escaneo

void setup() {
  Serial.begin(9600);          // Inicia la comunicación serial
  Wire.begin();                // Inicia la comunicación I2C

                                // Inicia los sensores y el servomotor
  ina219.begin();              // Inicia el sensor de corriente y voltaje INA219
  lightMeter.begin();          // Inicia el sensor de luz BH1750
  myServo.attach(servoPin);    // Conecta el servomotor al pin 9

  myServo.write(0);            // Inicia el servomotor en el ángulo 0°

  // Mensaje de inicio en el monitor serial
  Serial.println("Iniciando el sistema...");
  Serial.println("Ángulo\tVoltaje (V)\tCorriente (mA)\tLuz (lx)");
}

void loop() {
  // Escaneo del ángulo de 0 a 180 grados en pasos de 10 para encontrar el
  // mejor ángulo
  if (scanCycle < 3) {
    for (int angle = 0; angle <= 180; angle += 10) {
      myServo.write(angle);     // Mueve el servomotor al ángulo actual
      delay(1000);              // Espera para estabilizar
    }
  }
}
```



```
// Lee los valores del INA219 y BH1750
float busVoltage = ina219.getBusVoltage_V();    // Lee el voltaje del bus (en
voltios)
float current_mA = ina219.getCurrent_mA();    // Lee la corriente (en
miliamperios)
float lux = lightMeter.readLightLevel();    // Lee la irradiancia (en lux)

// Muestra los valores en el monitor serial
Serial.print(angle);    // Ángulo actual del servomotor
Serial.print("\t");
Serial.print(busVoltage, 2);    // Voltaje con dos decimales
Serial.print("\t");
Serial.print(current_mA, 2);    // Corriente con dos decimales
Serial.print("\t");
Serial.print(lux);    // Nivel de luz en lux
Serial.println();

// ... resto del código ...
}
} else {
    // Mantener el servomotor en el mejor ángulo y continuar leyendo los valores
    del sensor
    myServo.write(bestAngle); // Mantiene el servomotor en el mejor ángulo

    // Lectura continua de los valores del INA219 en tiempo real
    for (int i = 0; i < 600; i++) { // Lee durante 5 minutos (600 ciclos de 500 ms)
        float busVoltage = ina219.getBusVoltage_V(); // Lee el voltaje
        float current_mA = ina219.getCurrent_mA(); // Lee la corriente

        // Muestra los valores en el monitor serial sin etiquetas
        Serial.print(busVoltage, 2);
        Serial.print("\t");
        Serial.println(current_mA, 2);

        delay(500); // Intervalo de 500 ms para actualizar los datos
    }

    // Restablecer variables para reiniciar el escaneo después de 5 minutos
    scanCycle = 0;    // Reinicia el ciclo de escaneo
    maxLux = 0;    // Reinicia el valor máximo de lux
    measurementCount = 0; // Reinicia el contador de mediciones
}
}
```


Graficas del cálculo de voltaje en la página web.

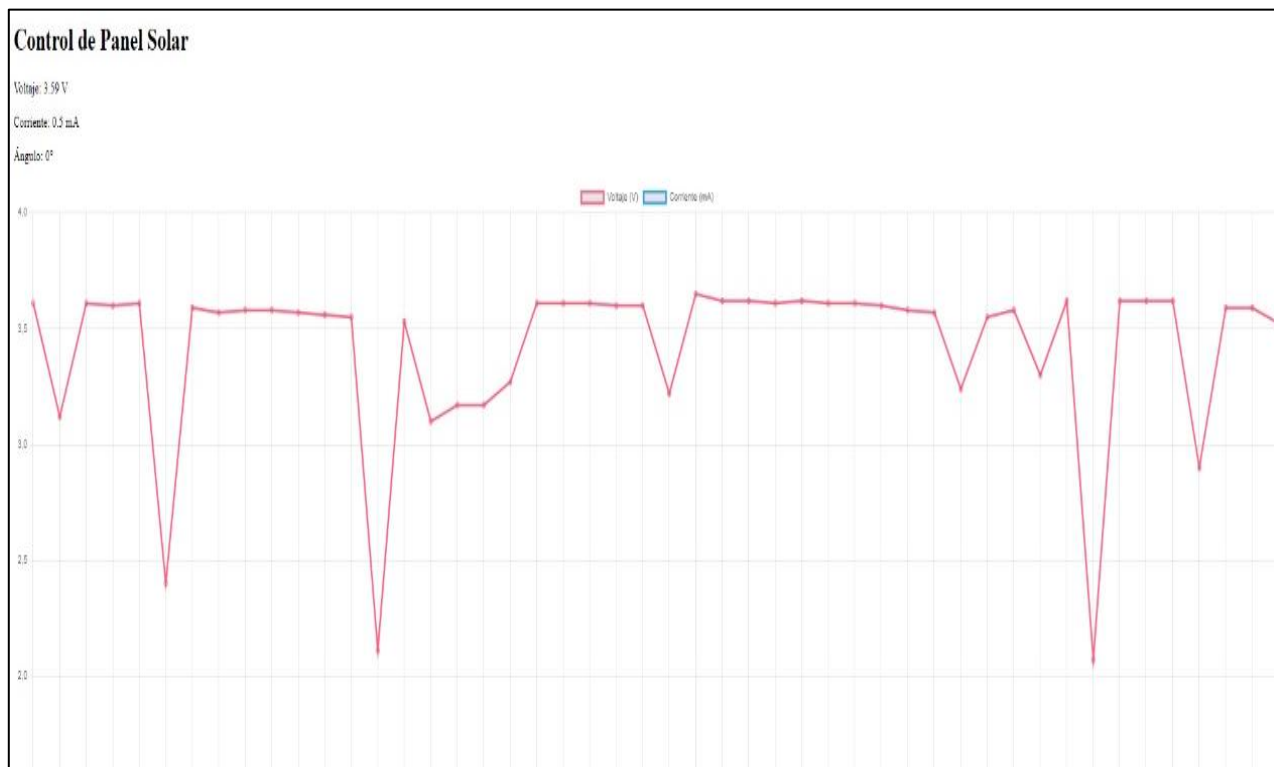


Ilustración 15 Gráfica de voltaje

Grafica de voltaje y corriente mandados directamente a la pagina web.

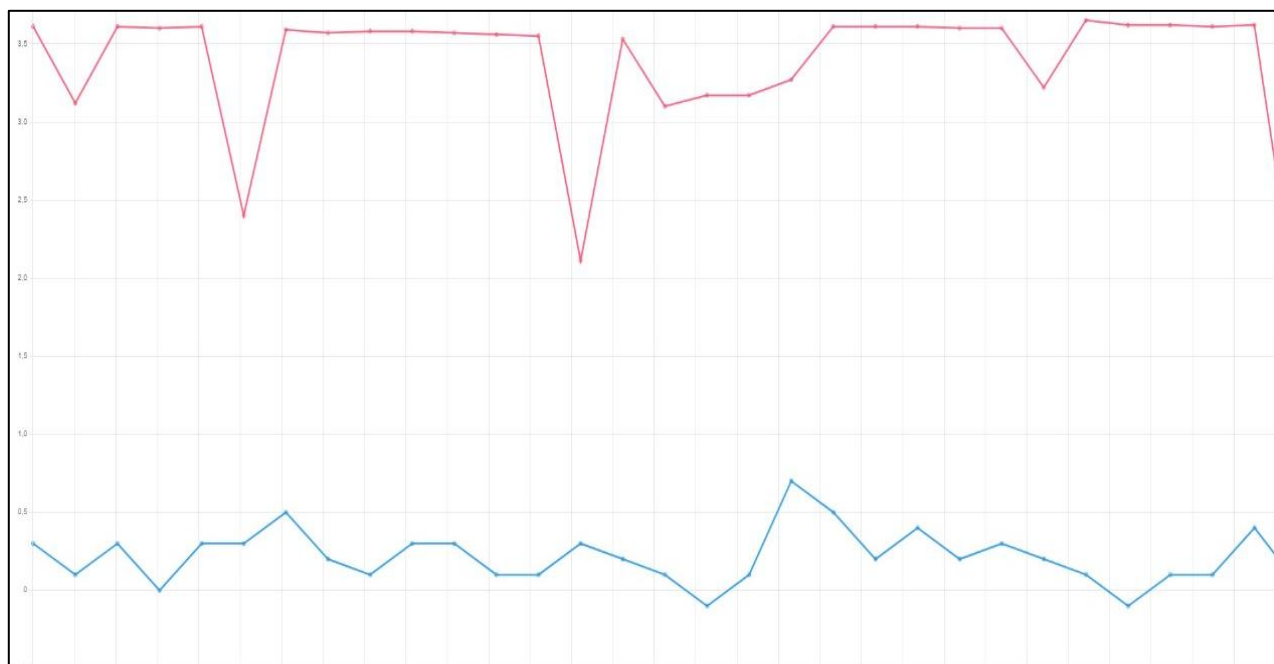
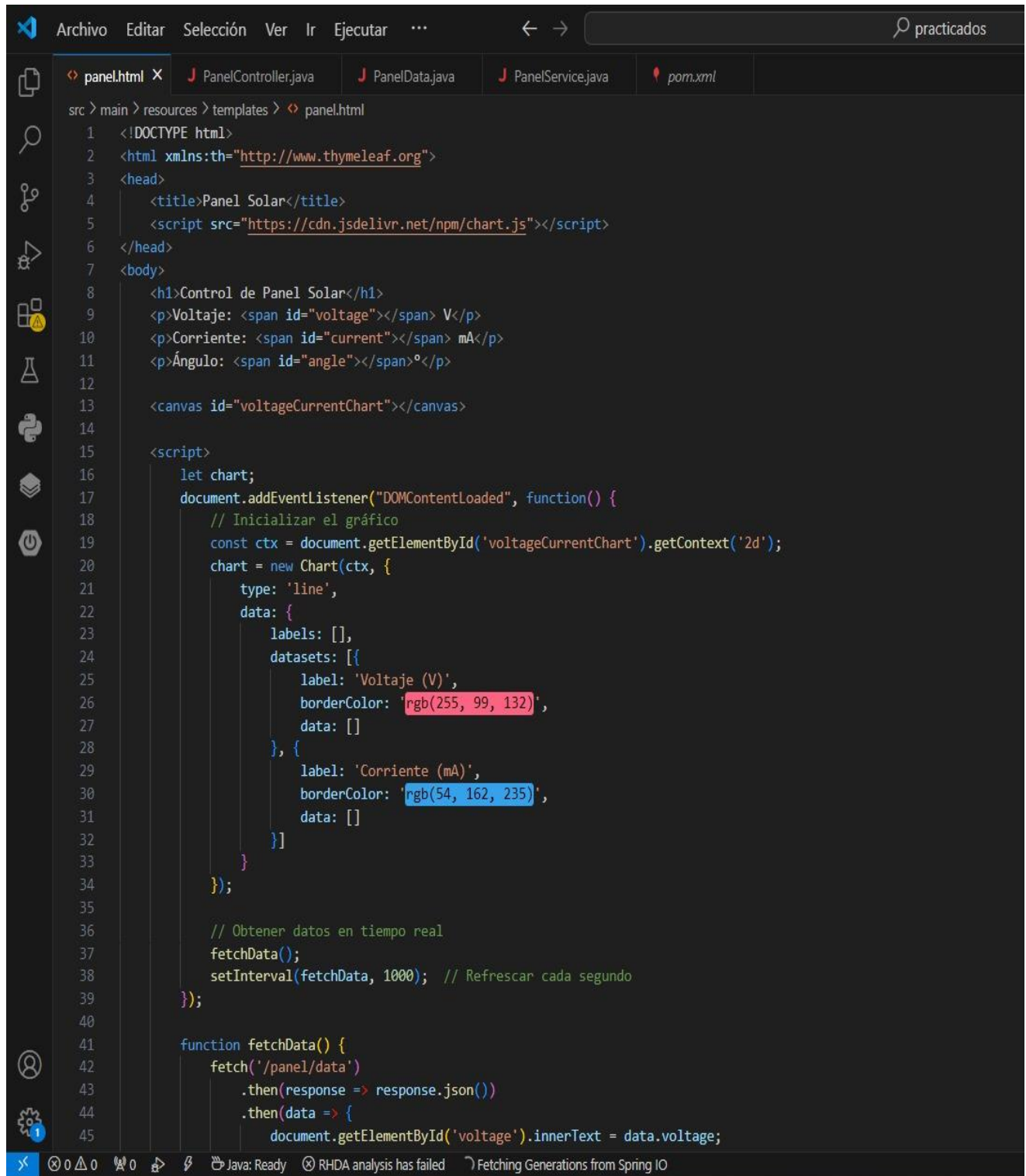


Ilustración 16 Grafica de voltaje y corriente

Código de HTML desde visual studio



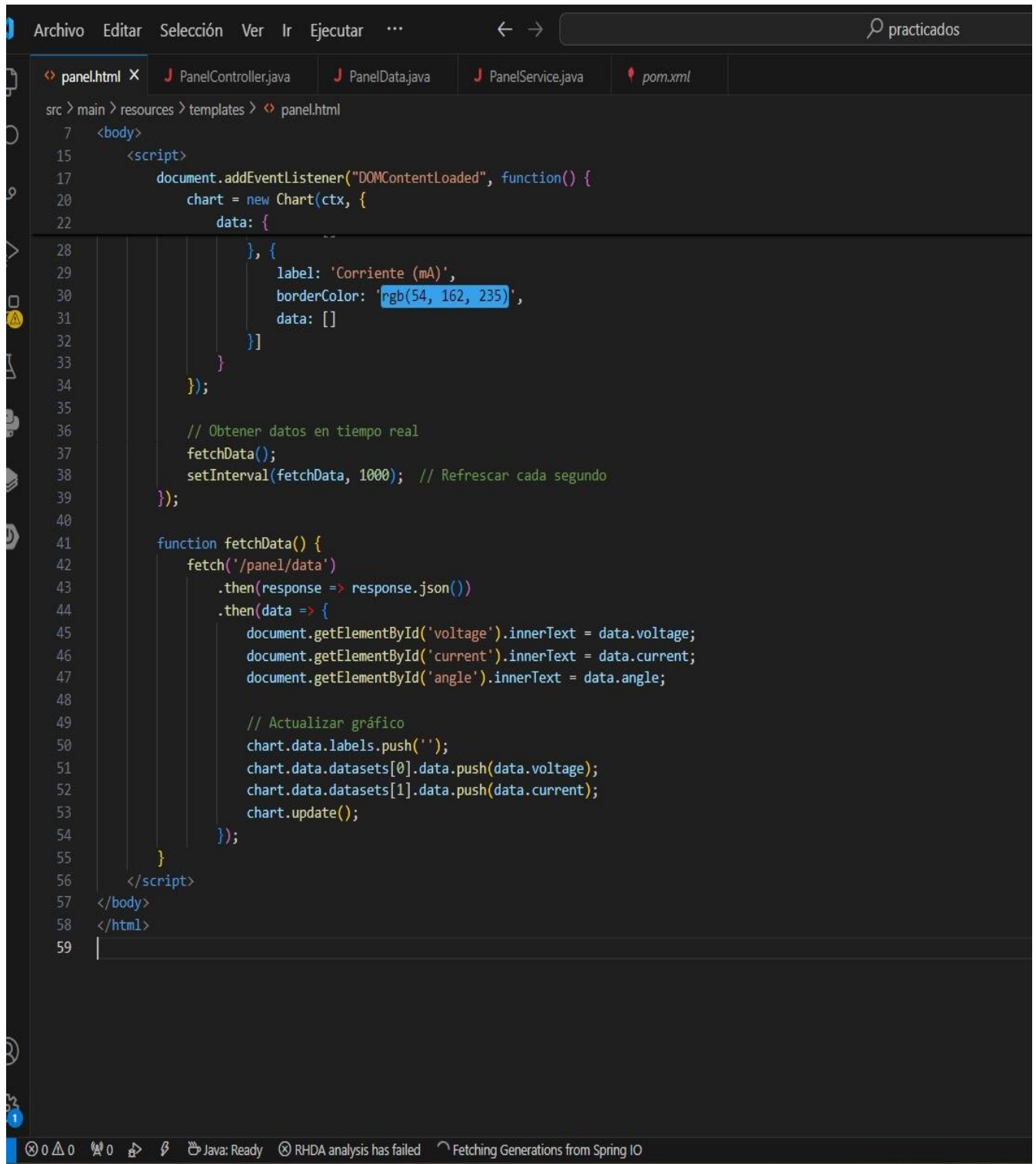
```

src > main > resources > templates > panel.html
1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <title>Panel Solar</title>
5      <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
6  </head>
7  <body>
8      <h1>Control de Panel Solar</h1>
9      <p>Voltaje: <span id="voltage"></span> V</p>
10     <p>Corriente: <span id="current"></span> mA</p>
11     <p>Ángulo: <span id="angle"></span>°</p>
12
13     <canvas id="voltageCurrentChart"></canvas>
14
15     <script>
16         let chart;
17         document.addEventListener("DOMContentLoaded", function() {
18             // Inicializar el gráfico
19             const ctx = document.getElementById('voltageCurrentChart').getContext('2d');
20             chart = new Chart(ctx, {
21                 type: 'line',
22                 data: {
23                     labels: [],
24                     datasets: [{
25                         label: 'Voltaje (V)',
26                         borderColor: 'rgb(255, 99, 132)',
27                         data: []
28                     }, {
29                         label: 'Corriente (mA)',
30                         borderColor: 'rgb(54, 162, 235)',
31                         data: []
32                     }]
33                 }
34             });
35
36             // Obtener datos en tiempo real
37             fetchData();
38             setInterval(fetchData, 1000); // Refrescar cada segundo
39         });
40
41         function fetchData() {
42             fetch('/panel/data')
43                 .then(response => response.json())
44                 .then(data => {
45                     document.getElementById('voltage').innerText = data.voltage;

```

Ilustración 17 Panel HTML -1

Segunda parte del código de HTML



```
Archivo  Editar  Selección  Ver  Ir  Ejecutar  ...  ←  →  practicos

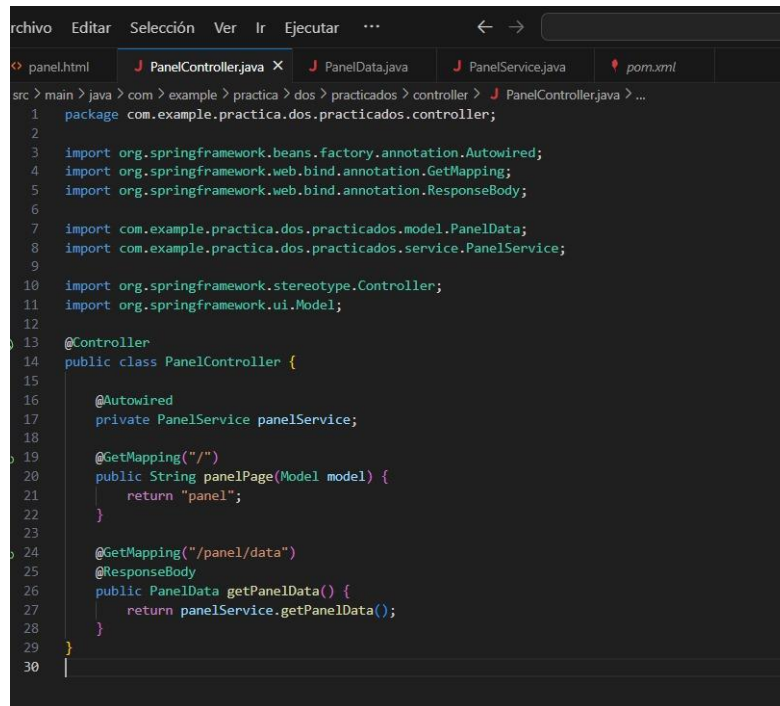
panel.html x  PanelController.java  PanelData.java  PanelService.java  pom.xml

src > main > resources > templates > panel.html
7  <body>
15  <script>
17      document.addEventListener("DOMContentLoaded", function() {
20          chart = new Chart(ctx, {
22              data: {
28                  }, {
29                  label: 'Corriente (mA)',
30                  borderColor: 'rgb(54, 162, 235)',
31                  data: []
32              }
33          });
34      };
35
36      // Obtener datos en tiempo real
37      fetchData();
38      setInterval(fetchData, 1000); // Refrescar cada segundo
39  });
40
41  function fetchData() {
42      fetch('/panel/data')
43      .then(response => response.json())
44      .then(data => {
45          document.getElementById('voltage').innerText = data.voltage;
46          document.getElementById('current').innerText = data.current;
47          document.getElementById('angle').innerText = data.angle;
48
49          // Actualizar gráfico
50          chart.data.labels.push('');
51          chart.data.datasets[0].data.push(data.voltage);
52          chart.data.datasets[1].data.push(data.current);
53          chart.update();
54      });
55  }
56  </script>
57  </body>
58  </html>
59  |

Java: Ready  RHDA analysis has failed  Fetching Generations from Spring IO
```

Ilustración 18 Panel HTML-2

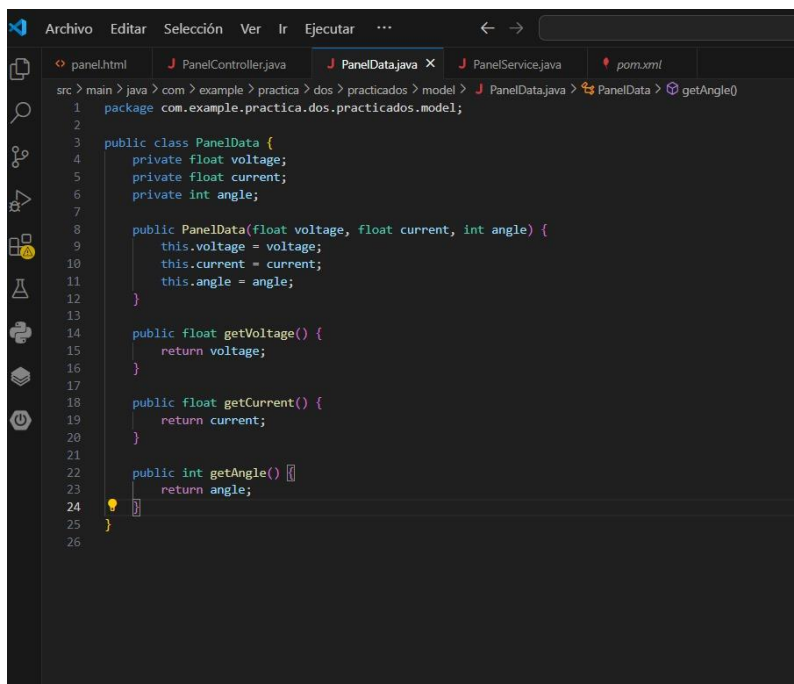
Ventana del controlador el cual es para poder controlar el circuito



```
1 package com.example.practica.dos.practicados.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.ResponseBody;
6
7 import com.example.practica.dos.practicados.model.PanelData;
8 import com.example.practica.dos.practicados.service.PanelService;
9
10 import org.springframework.stereotype.Controller;
11 import org.springframework.ui.Model;
12
13 @Controller
14 public class PanelController {
15
16     @Autowired
17     private PanelService panelService;
18
19     @GetMapping("/")
20     public String panelPage(Model model) {
21         return "panel";
22     }
23
24     @GetMapping("/panel/data")
25     @ResponseBody
26     public PanelData getPanelData() {
27         return panelService.getPanelData();
28     }
29
30 }
```

Ilustración 19 Controlador

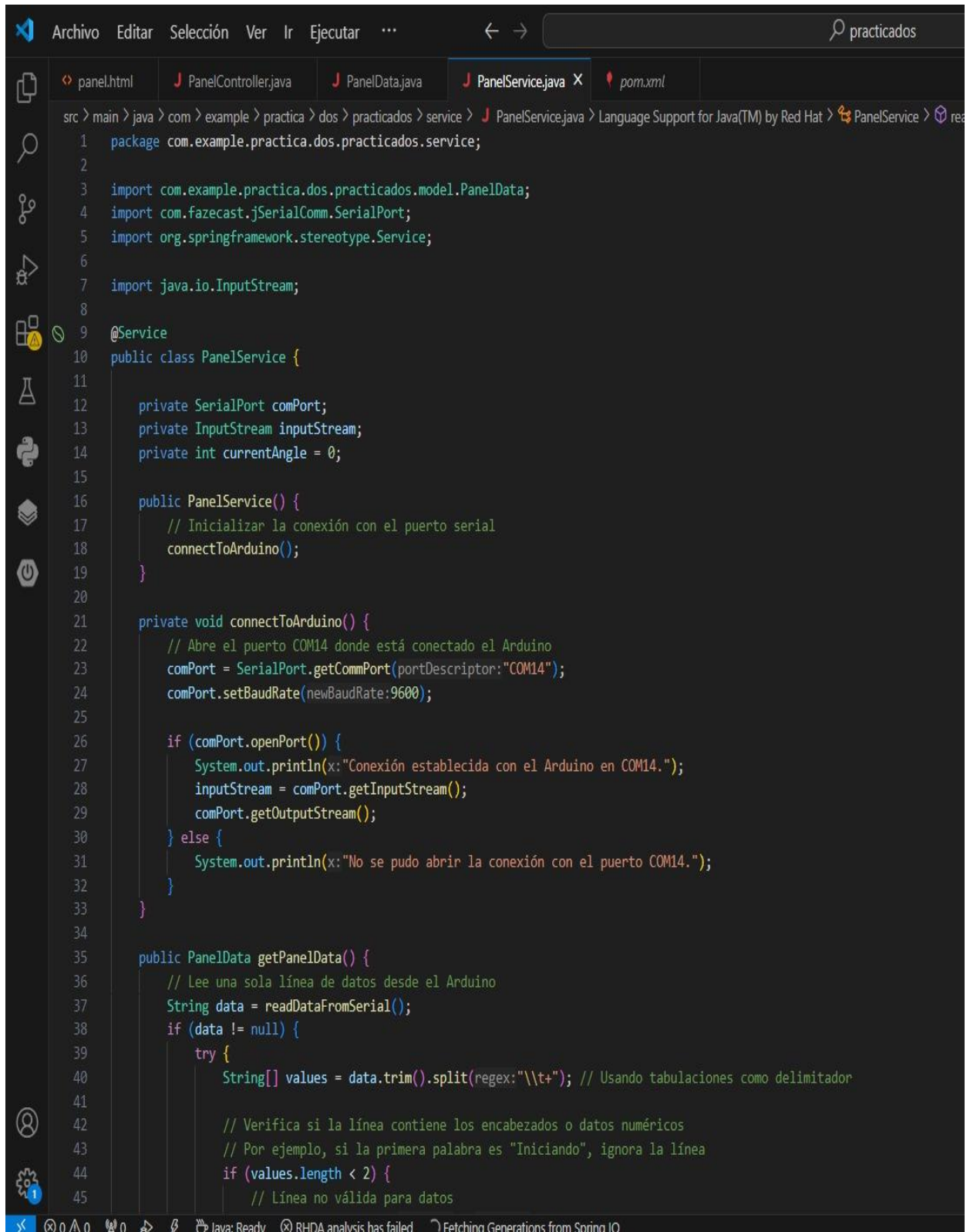
La siguiente ventana es del panel de datos el cual se encargara de capturas los datos de la grafica



```
1 package com.example.practica.dos.practicados.model;
2
3 public class PanelData {
4     private float voltage;
5     private float current;
6     private int angle;
7
8     public PanelData(float voltage, float current, int angle) {
9         this.voltage = voltage;
10        this.current = current;
11        this.angle = angle;
12    }
13
14    public float getVoltage() {
15        return voltage;
16    }
17
18    public float getCurrent() {
19        return current;
20    }
21
22    public int getAngle() {
23        return angle;
24    }
25
26 }
```

Ilustración 20 Panel de DATOS

El panel **SERVICE** es la que tendra conexión establecida con arduino



```
src > main > java > com > example > practica > dos > practicados > service > J PanelService.java > Language Support for Java(TM) by Red Hat > PanelService > res
1 package com.example.practica.dos.practicados.service;
2
3 import com.example.practica.dos.practicados.model.PanelData;
4 import com.fazecast.jSerialComm.SerialPort;
5 import org.springframework.stereotype.Service;
6
7 import java.io.InputStream;
8
9 @Service
10 public class PanelService {
11
12     private SerialPort comPort;
13     private InputStream inputStream;
14     private int currentAngle = 0;
15
16     public PanelService() {
17         // Inicializar la conexión con el puerto serial
18         connectToArduino();
19     }
20
21     private void connectToArduino() {
22         // Abre el puerto COM14 donde está conectado el Arduino
23         comPort = SerialPort.getCommPort(portDescriptor:"COM14");
24         comPort.setBaudRate(newBaudRate:9600);
25
26         if (comPort.openPort()) {
27             System.out.println(x:"Conexión establecida con el Arduino en COM14.");
28             inputStream = comPort.getInputStream();
29             comPort.getOutputStream();
30         } else {
31             System.out.println(x:"No se pudo abrir la conexión con el puerto COM14.");
32         }
33     }
34
35     public PanelData getPanelData() {
36         // Lee una sola línea de datos desde el Arduino
37         String data = readDataFromSerial();
38         if (data != null) {
39             try {
40                 String[] values = data.trim().split(regex:"\\t+"); // Usando tabulaciones como delimitador
41
42                 // Verifica si la línea contiene los encabezados o datos numéricos
43                 // Por ejemplo, si la primera palabra es "Iniciando", ignora la línea
44                 if (values.length < 2) {
45                     // Línea no válida para datos
```

Ilustración 21 Panel del SERVICE-1



Es la segunda parte del código de **SERVICE**

```
src > main > java > com > example > practica > dos > practicos > service > PanelService.java > Language Support for Java(TM) by Red Hat > PanelService > G
10 public class PanelService {
35     public PanelData getPanelData() {
45         // Línea no válida para datos
46         return new PanelData(voltage:0, current:0, currentAngle);
47     }
48
49     // Intenta analizar los valores
50     float voltage = 0;
51     float current = 0;
52
53     // Dependiendo del formato, extrae los valores correctamente
54     if (values[0].matches(regex:"\\d+")) { // Si el primer valor es numérico (ángulo)
55         // Formato inicial sin etiquetas
56         if (values.length == 3) {
57             voltage = Float.parseFloat(values[1]); // Voltaje en index 1
58             current = Float.parseFloat(values[2]); // Corriente en index 2
59         }
60     } else {
61         // Formato con etiquetas, e.g., "Voltaje (V): x.xx\tCorriente (mA): y.yy"
62         for (String value : values) {
63             if (value.startsWith(prefix:"Voltaje")) {
64                 String[] parts = value.split(regex:"\\s");
65                 if (parts.length == 2) {
66                     voltage = Float.parseFloat(parts[1].trim());
67                 } else if (value.startsWith(prefix:"Corriente")) {
68                     String[] parts = value.split(regex:"\\s");
69                     if (parts.length == 2) {
70                         current = Float.parseFloat(parts[1].trim());
71                     }
72                 }
73             }
74         }
75     }
76
77     return new PanelData(voltage, current, currentAngle);
78 } catch (NumberFormatException e) {
79     // Log el error y retorna valores por defecto
80     e.printStackTrace();
81 }
82
83 return new PanelData(voltage:0, current:0, currentAngle); // Retorna valores por defecto en caso de error
84 }
85
86 @SuppressWarnings("unused")
87 private float readVoltageFromArduino() {
```

Ilustración 22 Panel del SERVICE-2

Parte final del código de **SERVICE**

```
src > main > java > com > example > practica > dos > practicos > service > PanelService.java > Language Support for Java(TM) by Red Hat > PanelService > readCurrentFromArduino()
87 private float readVoltageFromArduino() {
88     return 0; // Return a default value in case of an error
89 }
90
91 @SuppressWarnings("unused")
92 private float readCurrentFromArduino() {
93     String data = readDataFromSerial();
94     if (data != null) {
95         try {
96             // Split the string by spaces or other delimiters (adjust depending on your data format)
97             String[] values = data.trim().split(regex:"\\s+");
98
99             // Parse the second value as current
100             return Float.parseFloat(values[2]); // Assuming 3rd value (index 2) is current
101         } catch (Exception e) {
102             e.printStackTrace();
103         }
104     }
105     return 0; // Return a default value in case of an error
106 }
107
108 private String readDataFromSerial() {
109     try {
110         // Leer los datos del puerto serial, asegurarse de leer hasta el final de la línea
111         StringBuilder dataBuilder = new StringBuilder();
112         int availableBytes = inputStream.available();
113         while (availableBytes > 0) {
114             char receivedChar = (char) inputStream.read();
115             dataBuilder.append(receivedChar);
116             if (receivedChar == '\n') {
117                 break;
118             }
119         }
120         availableBytes = inputStream.available(); // Update availableBytes after reading
121     } catch (Exception e) {
122         e.printStackTrace();
123     }
124     return dataBuilder.toString().trim(); // Retornar los datos leídos
125 }
126
127 }
128
129 }
130
131 }
132
133 }
134
135 }
136
137 }
138
139 }
140
141 }
```

Ilustración 23 Panel del SERVICE-3



PRACTICA #3

PRACTICA #4

Tenemos el siguiente prototipo del sensor SP que será de manera inalámbrica y mover el circuito o prender un led por medio de un Host.

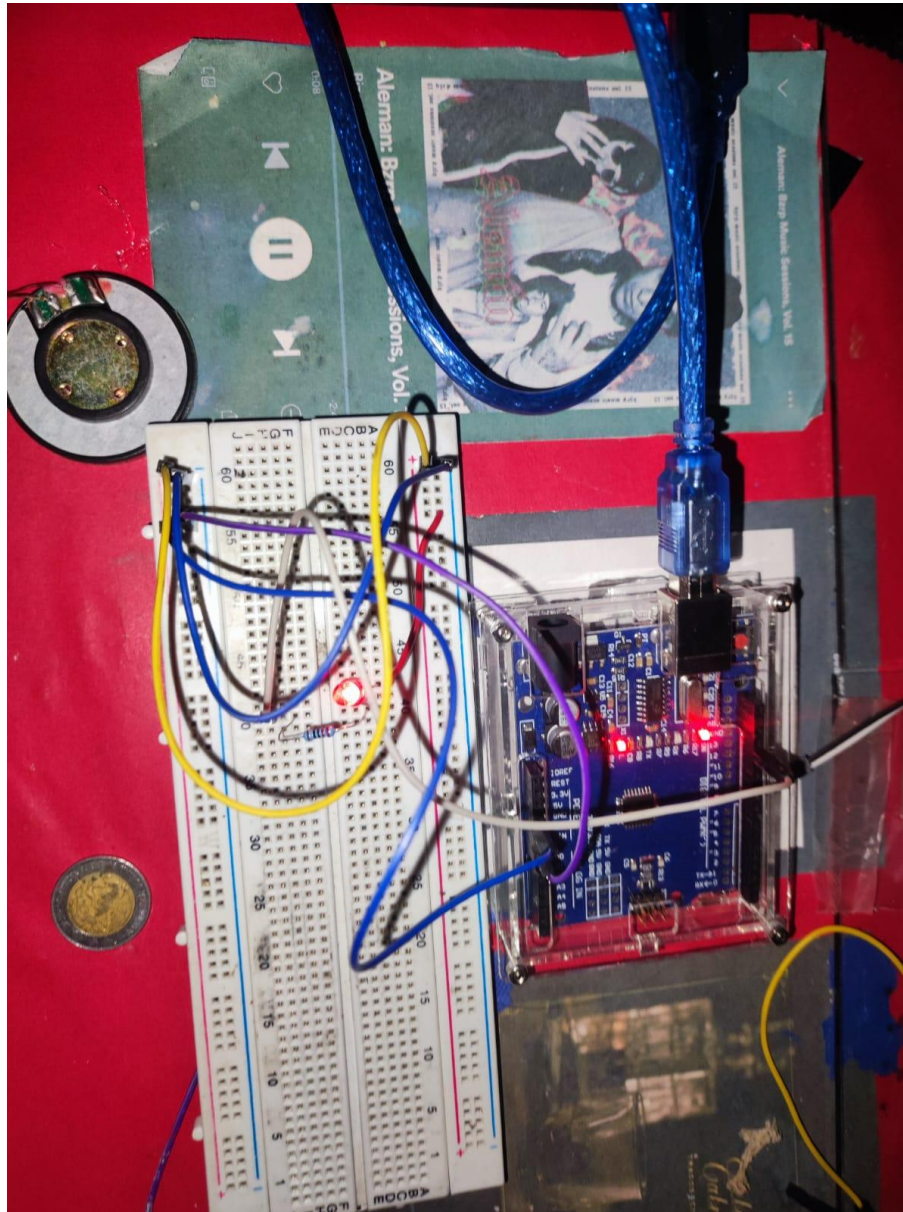


Ilustración 24 **PRACTICA#4**_Prototipo

Un esquemas detallado del circuito que estaremos empleando en esta practica de mover un led desde el host por medio del SP conectado desde el arduino.

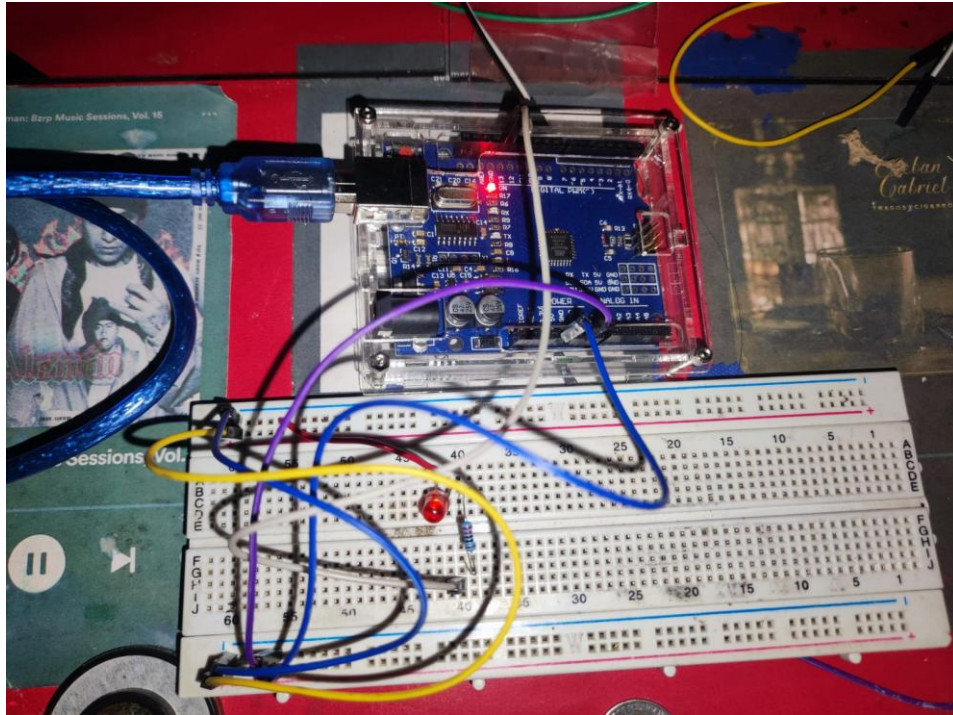


Ilustración 25 Estructura completa

Codigo en visual studio del controlador del circuito en el HTML, el cual hara la funcion de controlar el led desde el host.

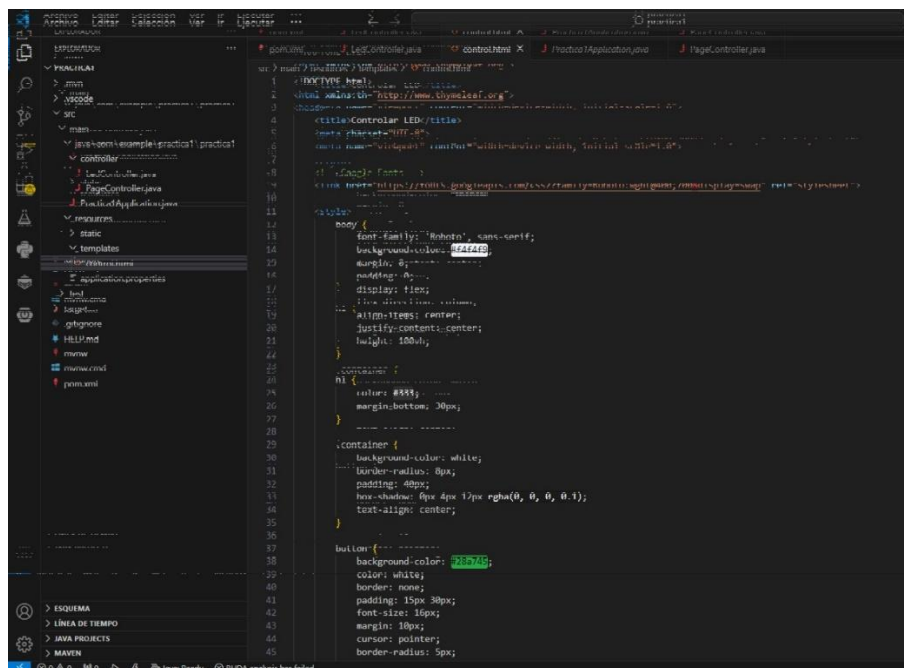
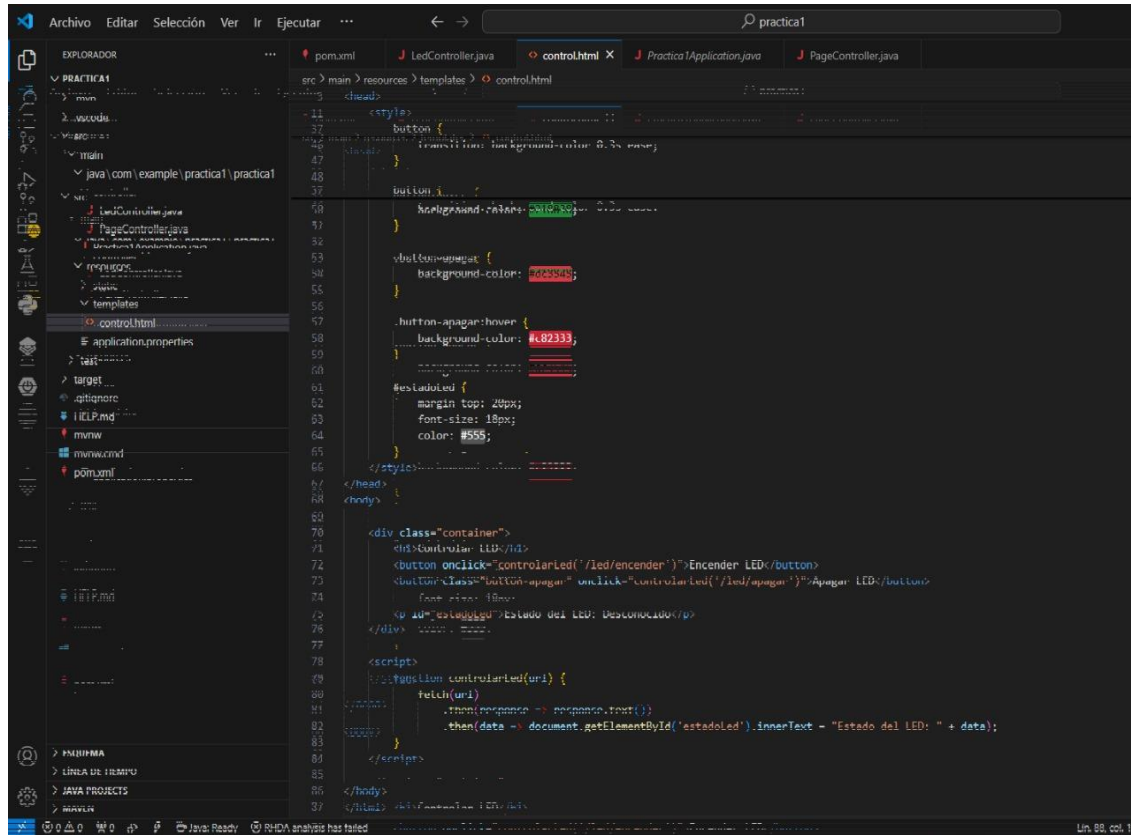


Ilustración 26 CONTROL HTML

En el siguiente paso ayudara a encender y apagar led, por medio del boton desde otra computadora o dispositivo android por medio de wifi o bluetooth.



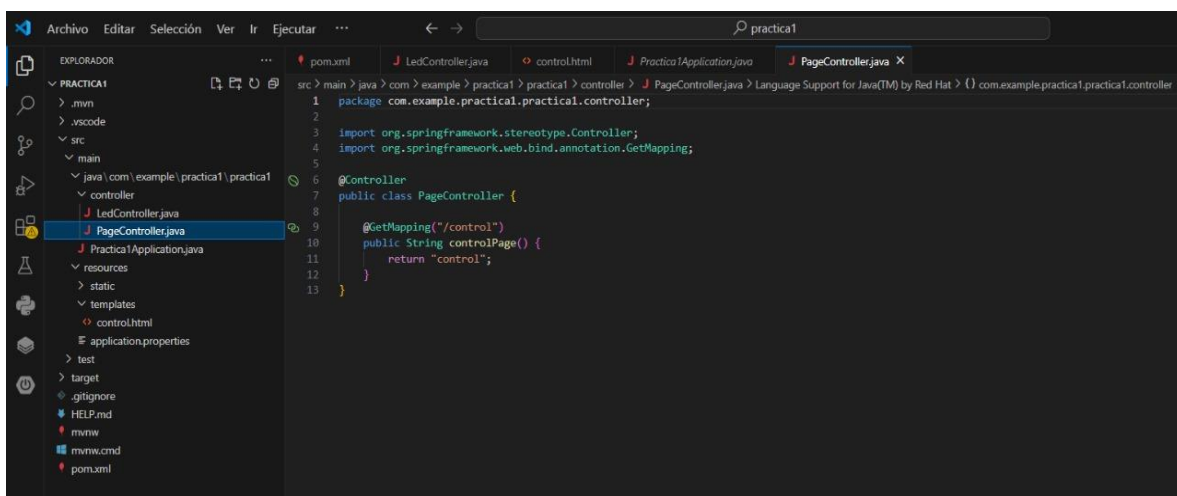
```

11 <style>
12   button {
13     background-color: #000000;
14     color: white;
15     padding: 5px 10px;
16     border: 1px solid black;
17     text-align: center;
18     width: 100px;
19     height: 30px;
20     margin: 5px;
21   }
22   .button-encender {
23     background-color: #000000;
24     color: white;
25   }
26   .button-apagar {
27     background-color: #000000;
28     color: white;
29   }
30   .button-apagar:hover {
31     background-color: #808080;
32   }
33   #estadoLed {
34     margin-top: 20px;
35     font-size: 18px;
36     color: #555;
37   }
38 </style>
39
40 <div class="container">
41   <div class="encender">
42     <button onclick="controlLed('/led/encender')">Encender LED</button>
43   </div>
44   <div class="apagar">
45     <button class="button-apagar" onclick="controlLed('/led/apagar')">Apagar LED</button>
46   </div>
47   <div class="estado">
48     <div id="estadoLed">Estado del LED: Desconocido</div>
49   </div>
50 </div>
51
52 <script>
53   function controlLed(uri) {
54     fetch(uri)
55       .then(response => response.text())
56       .then(data => document.getElementById('estadoLed').innerText = "Estado del LED: " + data);
57   }
58 </script>
59
60 </body>
61 </html>

```

Ilustración 27 CONTROL HTML

Registro final del controlador de led hará reiniciar los pasos que se estarán utilizando con un return, al presionar un botón de encender y apagar led-



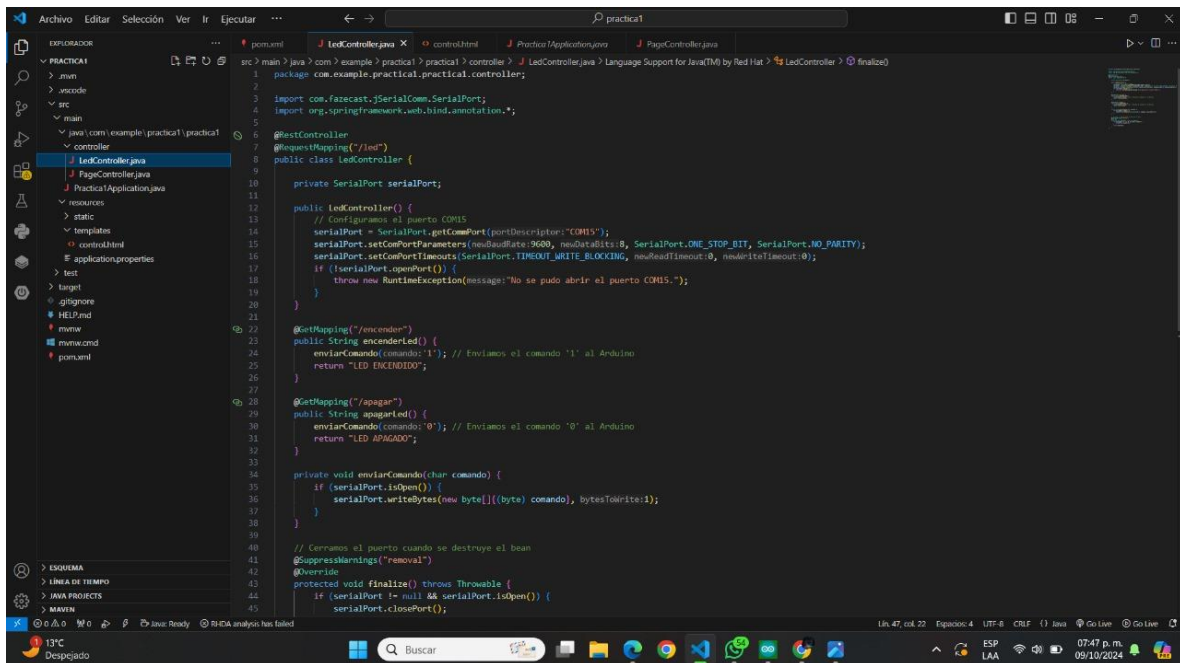
```

1 package com.example.practical.practical.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5
6 @Controller
7 public class PageController {
8
9   @GetMapping("/control")
10   public String controlPage() {
11     return "control";
12   }
13 }

```

Ilustración 28 PAGE_CONTROLLER

El siguiente código es el controlador de led comandado desde el puerto serial, con los botones de encender led que envía el comando '1' a Arduino y apagar el led con el comando '0' al Arduino

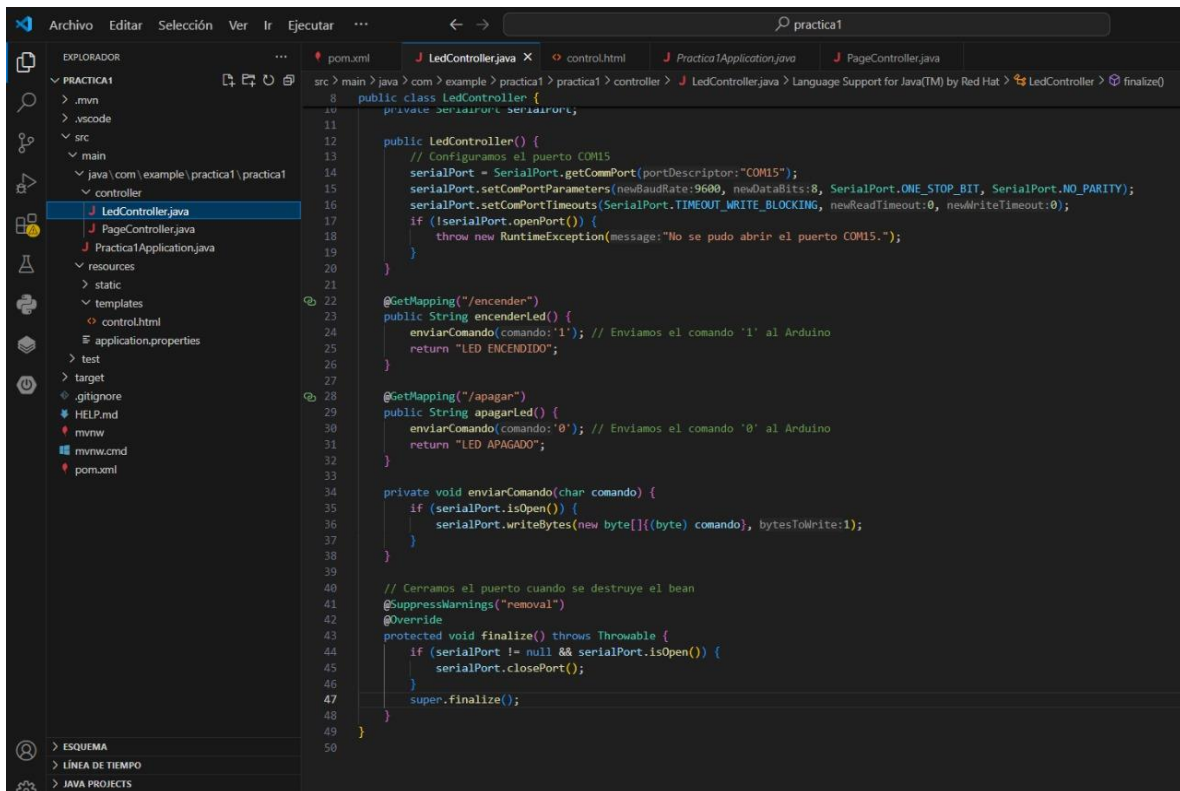


```

1 package com.example.practica1.controller;
2
3 import com.fasterxml.jackson.databind.ObjectMapper;
4 import org.springframework.web.bind.annotation.*;
5
6 @RestController
7 @RequestMapping("/led")
8 public class LedController {
9
10     private SerialPort serialPort;
11
12     public LedController() {
13         // Configuramos el puerto COM15
14         serialPort = SerialPort.getCommPort(portDescriptor:"COM15");
15         serialPort.setCommPortParameters(newBaudRate(9600, newDataBits(8, SerialPort.ONE_STOP_BIT, SerialPort.NO_PARITY));
16         serialPort.setCommPortTimeouts(SerialPort.TIMEOUT_WRITE_BLOCKING, newReadTimeout(0, newWriteTimeout(0));
17         if (!serialPort.openPort()) {
18             throw new RuntimeException(message:"No se pudo abrir el puerto COM15.");
19         }
20     }
21
22     @GetMapping("/encender")
23     public String encenderLed() {
24         enviarComando(comando:"1"); // Enviamos el comando '1' al Arduino
25         return "LED ENCENDIDO";
26     }
27
28     @GetMapping("/apagar")
29     public String apagarLed() {
30         enviarComando(comando:"0"); // Enviamos el comando '0' al Arduino
31         return "LED APAGADO";
32     }
33
34     private void enviarComando(char comando) {
35         if (serialPort.isOpen()) {
36             serialPort.writeBytes(new byte[] {(byte) comando}, bytesToWrite:1);
37         }
38     }
39
40     // Cerramos el puerto cuando se destruye el bean
41     @SuppressWarnings("removal")
42     @Override
43     protected void finalize() throws Throwable {
44         if (serialPort != null && serialPort.isOpen()) {
45             serialPort.closePort();
46         }
47     }
48 }

```

Ilustración 29 Led_Controller-1



```

8 public class LedController {
9     private SerialPort serialPort;
10
11     public LedController() {
12         // Configuramos el puerto COM15
13         serialPort = SerialPort.getCommPort(portDescriptor:"COM15");
14         serialPort.setCommPortParameters(newBaudRate(9600, newDataBits(8, SerialPort.ONE_STOP_BIT, SerialPort.NO_PARITY));
15         serialPort.setCommPortTimeouts(SerialPort.TIMEOUT_WRITE_BLOCKING, newReadTimeout(0, newWriteTimeout(0));
16         if (!serialPort.openPort()) {
17             throw new RuntimeException(message:"No se pudo abrir el puerto COM15.");
18         }
19     }
20
21     @GetMapping("/encender")
22     public String encenderLed() {
23         enviarComando(comando:"1"); // Enviamos el comando '1' al Arduino
24         return "LED ENCENDIDO";
25     }
26
27     @GetMapping("/apagar")
28     public String apagarLed() {
29         enviarComando(comando:"0"); // Enviamos el comando '0' al Arduino
30         return "LED APAGADO";
31     }
32
33     private void enviarComando(char comando) {
34         if (serialPort.isOpen()) {
35             serialPort.writeBytes(new byte[] {(byte) comando}, bytesToWrite:1);
36         }
37     }
38
39     // Cerramos el puerto cuando se destruye el bean
40     @SuppressWarnings("removal")
41     @Override
42     protected void finalize() throws Throwable {
43         if (serialPort != null && serialPort.isOpen()) {
44             serialPort.closePort();
45         }
46     }
47     super.finalize();
48 }

```

Ilustración 30 Led_Controller-2



Código en arduino IDE usado para la práctica de encender un led desde otro dispositivo

Ilustración 31 Código de Arduino

Esquema de como tendria que ir conectado los led y las resistencias con los cables al arduino uno

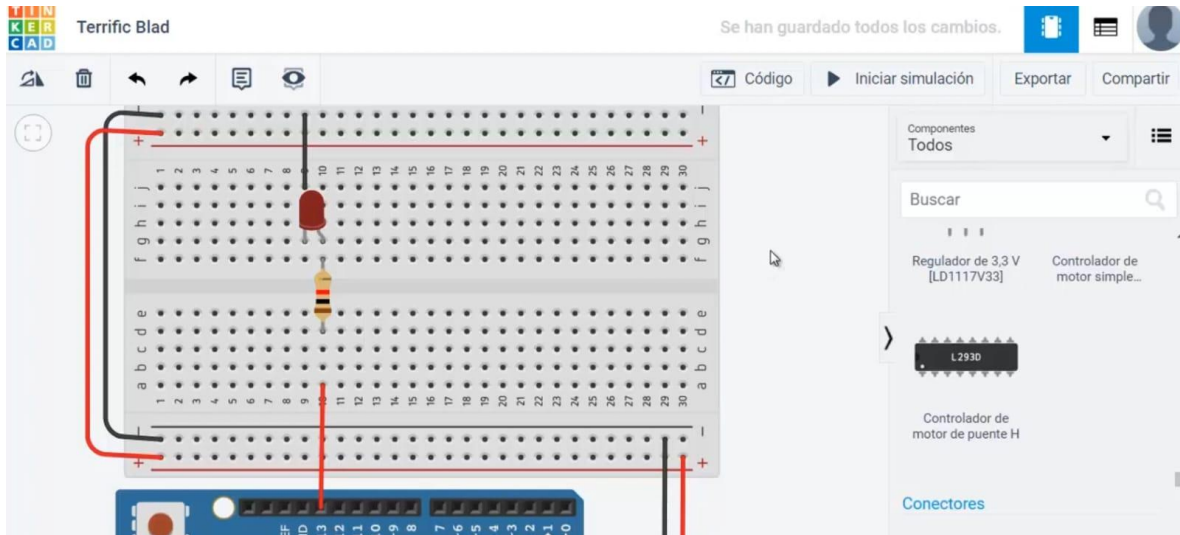


Ilustración 32 Simulación

Se dirige al controlador web con el siguiente link: localhost:9525/control y muestran los botones de encender led y apagar led, asi mismo se ve el estado del del led que esta APAGADO.

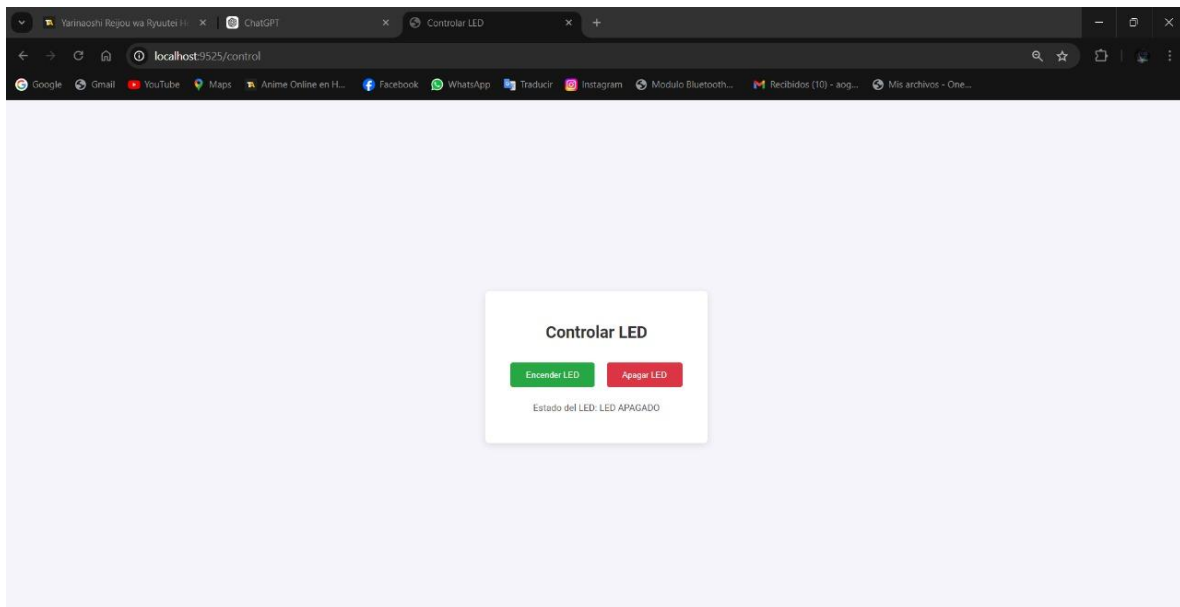


Ilustración 33 Led apagado

En esta siguiente ventana encontramos el controlador pero con el estado de led encendido.

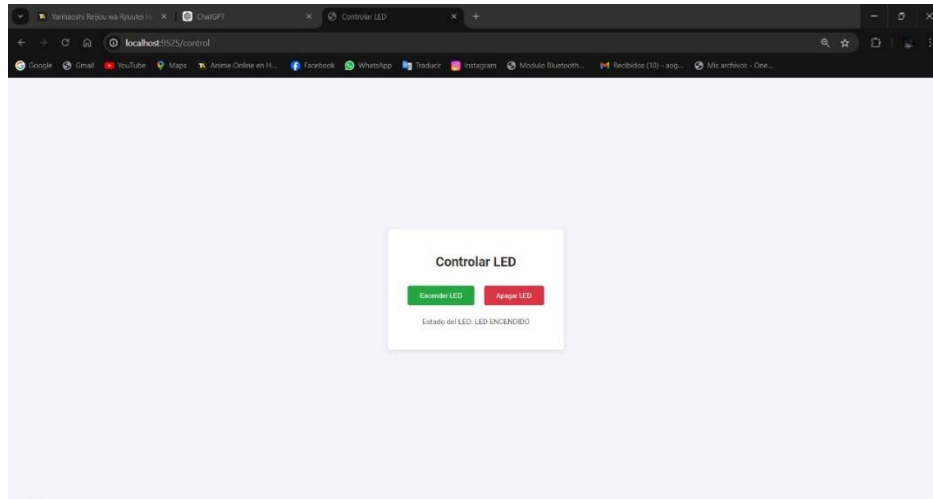


Ilustración 34 Led encendido

La parte final es poner a prueba el localhost por medio de un dispositivo alterno al que estamos utilizando por este metodo utilizamos un dispositivo Android.

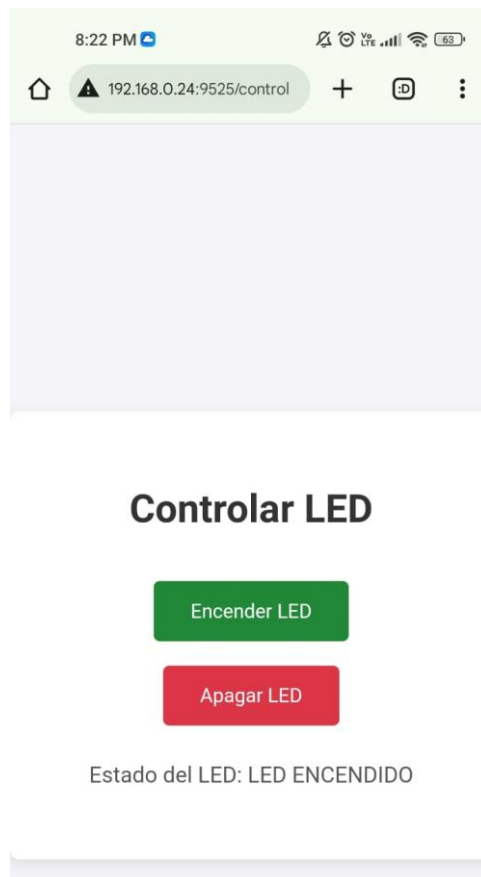


Ilustración 35 Funcionamiento desde otro dispositivo



Conclusión

Reconocemos el arduo trabajo en equipo y para llevar a cabo un prototipo de esta mejora, es tener cada quien su diferente cambio entre lo particular nos sirvió para conocer y ver como funciona un sensor BH-1750 y como se comporta desde el Arduino y desde un sistema creado desde visual studio y la importancia de saber como se utilizan los elementos a un bajo flujo de propiedades