Proyecto

ESP-32 LoRa V2

Saúl Rodríguez Naranjo
Juan Alberto Domínguez Vázquez

Índice

- Introducción
- Desarrollo código Arduino y medición de sensores
- Backend, Frontend Java y despliegue en servidor cloud
- Conclusiones

Introducción

Para el proyecto se ha decidido diseñar y desarrollar una arquitectura cliente-servidor entre la placa con el ESP-32 incrustado y un servidor en el cloud. Si entramos en más detalle, por un lado, se ha creado un programa en Arduino encargado de inicializar, configurar y recoger los valores medidos por un sensor de partículas Honeywell y un sensor GPS.

Por otro lado, un proyecto en Java, desplegado en el servidor, será el encargado de recoger dichos datos de los sensores para mostrarlos en una página web en tiempo real.

ESP-32
+
Sensores

Servidor en el Cloud

Proyecto Java

Desarrollo código Arduino y medición de sensores

Inteligencia Ambiental

Desarrollo código Arduino y medición de sensores

Para comenzar, se definen las librerías necesarias para el correcto funcionamiento del programa.

También, se configuran los pines de control del multiplexor necesarios para su conexionado con el ESP-32 según el esquema aportado en clases, así como los pines que activan los sensores.

Por último, se configura la pantalla del ESP-32, se crea un objeto display para manejarla mediante I2C y crear un objeto ui para administrar la interfaz de usuario que se mostrará en la pantalla.

```
#include <HPMA115_Compact.h>
#include <Wire.h>
#include <HardwareSerial.h>
#include "HT_SSD1306Wire.h"
#include "HT_DisplayUi.h"
#include <TinyGPSMinus.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <time.h>

// Configuración de pines del multiplexor
int muxA = 0;   // Pin de control A
int muxB = 23;  // Pin de control B

// Pines para activación de sensores
int on_gps = 33;   // Activación GPS
int on_aire = 32;  // Activación sensor de partículas
int ledTest = 25;  // LED de estado

// OLED Configuration
#ifdef WIRELESS_STICK_V3
static SSD1306Wire display(0x3c, 500000, SDA_OLED, SCL_OLED, GEOMETRY_64_32, RST_OLED);
#else
static SSD1306Wire display(0x3c, 500000, SDA_OLED, SCL_OLED, GEOMETRY_128_64, RST_OLED);
#endif
DisplayUi ui(&display);
```

Desarrollo código Arduino y medición de sensores

Al principio, definimos los parámetros para la conexión WiFi, así como la URL en la que esta escuchando el proyecto Java para enviar los datos y una señal de status alive.

En las siguientes 3 líneas configuramos las variables para realizar una petición http a un servidor NTP y poder obtener la fecha y hora actual, usada para enviar los datos posteriormente. Después se define una instancia de WiFiClient para su uso y conexión con la red WiFi.

Por último, se crea y construye un objeto de la librería HPMA115_Compact para poder utilizar el sensor de partículas Honeywell, se instancia un objeto de la librería TinyGPSMinus para el uso del sensor GPS y la UART compartida para el GPS y el sensor de partículas.

```
28  const char* ssid = "MOVISTAR-WIFI6-6D30";
29  const char* password = [redacted]
30  const char* serverUrl = "http://212.227.87.151:8080/save-particle-data";
31  const char* serverUrlStatus = "http://212.227.87.151:8080/set-esp32-status";
32
33  // Configuración de la zona horaria y servidor NTP
34  const char* ntpServer = "pool.ntp.org";
35  const long gmtOffset_sec = 3600;      // Ajuste para la zona horaria (GMT+1 = 3600s)
36  const int daylightOffset_sec = 3600;  // Ajuste para horario de verano/invierno
37
38  WiFiClient client;
39
40  // Sensores y datos
41  HPMA115_Compact hpm = HPMA115_Compact();
42  TinyGPSMinus gps;  // Inicialización de TinyGPSMinus
43
44  // UART
45  HardwareSerial sensorSerial(2);  // UART2 compartida para GPS y partículas
```

Desarrollo código Arduino y medición de sensores

En esta parte se definen todas las variables necesarias para recoger los valores medidos por los sensores. Después comienza el setup.

En él, comenzamos la comunicación con el puerto serie a 115200 baudios y la UART con los sensores a 9600, pasándoles además los pines RX = 2 y TX = 17.

Por otro lado, se configura el OLED inicializando el display y se configura los pines de activación de los sensores y del multiplexor como salidas.

```
47  // Variables de datos
48  int pm10 = 0;
49  int pm25 = 0;
50  char lat[9];
51  char lon[10];
52  float latitudeValue = 0.0;
53  float longitudeValue = 0.0;
54  char latDirection;
55  char lonDirection;
56
57  void setup() {
58    Serial.begin(115200);
59
60    // Configuración UART compartida
61    sensorSerial.begin(9600, SERIAL_8N1, 2, 17);  // RX=2, TX=17
62
63    // Configuración del OLED
64    display.init();
65    display.clear();
66    display.display();
67    display.setContrast(255);
68
69    // Configuración de pines
70    pinMode(on_gps, OUTPUT);
71    pinMode(on_aire, OUTPUT);
72    pinMode(muxA, OUTPUT);
73    pinMode(muxB, OUTPUT);
74    pinMode(ledTest, OUTPUT);
```

Desarrollo código Arduino y medición de sensores

Continuamos en el setup escribiendo el valor necesario en los pines de activación de los sensores para desactivarlos. Después inicializamos el sensor de partículas.

En las líneas de abajo se configuran los parámetros WiFi determinados para prevenir situaciones anómalas con la conexión.

Por último, intentamos conectar a la wifi y si se produce un fallo entramos en el bucle mostrando puntos en la consola.

```
76   // Inicialización de sensores
77   digitalWrite(on_gps, HIGH);  // Desactiva GPS
78   digitalWrite(on_aire, LOW);  // Desactiva sensor partículas
79   digitalWrite(ledTest, LOW);  // Apaga LED
80   delay(1000);
81
82   // Inicialización pantalla
83   display.clear();
84   display.display();
85   Serial.println("Sistema inicializado.");
86
87   // Inicialización de sensores
88   hpm.begin(&sensorSerial);
89
90   //Parametros para la conexion WiFi
91   WiFi.mode(WIFI_STA);
92   WiFi.persistent(false);
93   WiFi.setAutoReconnect(true);
94   WiFi.setSleep(false);
95
96   Serial.println("\nConnecting");
97   WiFi.begin(ssid, password);
98
99   while (WiFi.status() != WL_CONNECTED) {
100    // failed, retry
101    Serial.print(".");
102    delay(500);
103  }
```

Desarrollo código Arduino y medición de sensores

```
105   Serial.println("You're connected to the network");
106   Serial.print("Connected, IP address: ");
107   Serial.println(WiFi.localIP());
108   Serial.println(WiFi.getHostname());
109   Serial.println();
110
111   // Configurar y sincronizar tiempo NTP
112   configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
113   Serial.println("Tiempo sincronizado con NTP");
114   delay(2000);
115
116   Serial.println("Seleccionando GPS...");
117   selectGPS();
118   readGPS();
119   while (latitudeValue == 0.0000000 || longitudeValue == 0.0000000 || latitudeValue == 99.99 || longitudeValue == 99.99) {
120     Serial.println("Esperando datos GPS válidos...");
121     readGPS();  // Volver a leer GPS
122   }
123
124   Serial.println("Seleccionando sensor de partículas...");
125   selectParticles();
126 }
```

Para finalizar la función de Setup, mostramos por consola si estamos conectados a la WiFi y la IP otorgada por el router.

Después configuramos y sincronizamos la fecha y hora con los parámetros anteriores. A continuación, procedemos a, mediante la función selectGPS(), enviar a los pines de control del multiplexor la señal 00 y activar el pin On_GPS. Siempre y cuando los valores sean válidos para evitar enviar datos erróneos.

En las últimas dos líneas pasamos a seleccionar y activar el sensor de partículas


```
216   void selectGPS() {
217     // Configuración del multiplexor para GPS (00)
218     digitalWrite(muxA, LOW);
219     digitalWrite(muxB, LOW);
220     digitalWrite(on_gps, LOW);   // Activa GPS
221     digitalWrite(on_aire, LOW);  // Desactiva sensor partículas
222     delay(2000);                 // Esperar a que el GPS esté listo
223   }
```

```
void selectParticles() {
  // Configuración del multiplexor para el sensor de partículas (01)
  digitalWrite(muxA, HIGH);
  digitalWrite(muxB, LOW);
  digitalWrite(on_gps, HIGH);   // Desactiva GPS
  digitalWrite(on_aire, HIGH);  // Activa sensor partículas
  delay(6000);
}
```

Desarrollo código Arduino y medición de sensores

Por último, mediante la función readGPS(), activamos el sensor GPS y leemos los datos del mismo. Esos datos aportados por el sensor están en formato GMM, por lo que deberán ser transformados a formato decimal (DD) mediante la función convertDecimal(char* rawCoordinate, char direction)

```
225  void readGPS() {
226
227    // Activar el GPS
228    digitalWrite(on_gps, LOW);  // Encender GPS
229    Serial.println("Leyendo GPS...");
230
231    // Leer datos del GPS
232    while (sensorSerial.available()) {
233      char c = sensorSerial.read();
234      gps.encode(c);  // Decodificar los datos recibidos
235    }
236
237    strcpy(lat, gps.get_latitude());
238    strcpy(lon, gps.get_longitude());
239
240    latDirection = lat[strlen(lat) - 1];  // 'N' o 'S'
241    lonDirection = lon[strlen(lon) - 1];  // 'E' o 'W'
242
243    // Crear nuevas cadenas eliminando el carácter de dirección
244    char latitudeWithoutDir[16];
245    char longitudeWithoutDir[16];
246
247    strncpy(latitudeWithoutDir, lat, strlen(lat) - 1);
248    latitudeWithoutDir[strlen(lat) - 1] = '\0';
250    strncpy(longitudeWithoutDir, lon, strlen(lon) - 1);
251    longitudeWithoutDir[strlen(lon) - 1] = '\0';
252
253    latitudeValue = convertToDecimal(latitudeWithoutDir, latDirection);
254    longitudeValue = convertToDecimal(longitudeWithoutDir, lonDirection);
255
256    Serial.print("Latitud sin convertir: ");
257    Serial.println(lat);
258    Serial.print("Longitud sin convertir: ");
259    Serial.println(lon);
260
261    Serial.print("Latitud (decimal): ");
262    Serial.println(latitudeValue, 6);
263    Serial.print("Longitud (decimal): ");
264    Serial.println(longitudeValue, 6);
265
266    snprintf(lat, sizeof(lat), "%.6f", latitudeValue);
267    snprintf(lon, sizeof(lon), "%.6f", longitudeValue);
268  }
```

Desarrollo código Arduino y medición de sensores

```
float convertToDecimal(const char* rawCoordinate, char direction) {
  // Convertir a String para manipular más fácilmente
  String coordinateStr = String(rawCoordinate);

  // Separar grados y minutos
  int dotIndex = coordinateStr.indexOf('.');
  int degreesLength = (dotIndex > 2) ? dotIndex - 2 : dotIndex - 1;  // Verifica si son 2 o 3 dígitos de grados
  int degrees = coordinateStr.substring(0, degreesLength).toInt();   // Extraer grados
  float minutes = coordinateStr.substring(degreesLength).toFloat();  // Extraer minutos

  // Convertir a formato decimal
  float decimal = degrees + (minutes / 60.0);

  // Aplicar el signo según la dirección
  if (direction == 'S' || direction == 'W') {
    decimal *= -1;
  }

  return decimal;
}
```

Lo primero que hacemos en esta función es convertir los valores obtenidos en el GPS a un tipo String para facilitar su manejo.

Después encontramos la posición del punto decimal en la cadena, ya que separa los minutos enteros de sus fracciones. A continuación, determinados cuántos dígitos corresponden a los grados.

Para finalizar, extraemos los grados y minutos de la cadena y las convertimos a números entero y decimal respectivamente, para aplicar la fórmula de paso a formato decimal, y dotarle del signo correspondiente según su dirección.

Desarrollo código Arduino y medición de sensores

Pasamos a ver la función loop, donde la magia ocurre.

En primer lugar, procedemos a leer del sensor de partículas mediante la función readParticles que explicaremos a continuación.

Si los datos medidos son 0, no nos interesan así que nos quedaremos en un bucle mientras se de esa condición, esperando a medir datos correctos.

Después mostraremos las medidas por el OLED del ESP-32, mediante la función displayOLED().

```
void loop() {

  Serial.println("Leyendo partículas...");
  readParticles();

  while (pm10 == 0 || pm25 == 0) {
    Serial.println("Esperando datos de partículas válidos...");
    readParticles();  // Volver a leer partículas
  }

  Serial.println("Mostrando datos en OLED...");
  displayOled();
```

Desarrollo código Arduino y medición de sensores

En la función readParticles(), comenzamos deteniendo el autoenvío del sensor de partículas, leemos los nuevos datos que vaya recogiendo, tanto del contaminante PM10, como PM2.5, verificamos que no sean negativos y los mostramos en el puerto serie

```
void readParticles() {
  Serial.println("Iniciando comunicación con el sensor de partículas...");
  hpm.stopAutoSend();  // Detener autoenvío (si estaba activo)
  delay(50);

  hpm.readParticleMeasurementResults();
  delay(50);
  hpm.isNewDataAvailable();

  pm10 = hpm.getPM10();
  pm25 = hpm.getPM25();

  // Verificar si los datos son válidos
  if (pm10 < 0 || pm25 < 0) {
    Serial.println("Advertencia: Datos inválidos recibidos del sensor de partículas.");
    pm10 = 0;
    pm25 = 0;
  }

  Serial.print("PM10: ");
  Serial.print(pm10);
  Serial.print(" µg/m3, PM2.5: ");
  Serial.print(pm25);
  Serial.println(" µg/m3");
}
```

Desarrollo código Arduino y medición de sensores

En la función displayOLED() verificamos que el objeto OLED este inicializado y mostramos los datos de pm10 y pm25 por la pantalla del ESP-32, así como la latitud y la longitud medida por el GPS.

```
void displayOled() {
  // Verificar que el objeto OLED está inicializado
  if (!display.init()) {
    Serial.println("Error: El OLED no está inicializado.");
    return;
  }

  if (pm10 < 0 || pm25 < 0) {
    Serial.println("Error: Datos de partículas no válidos.");
    pm10 = 0;
    pm25 = 0;
  }

  display.clear();
  display.setFont(ArialMT_Plain_10);

  // Información a mostrar
  String gpsInfo = "Lat: " + String(latitudeValue, 5) + " Lon: " + String(longitudeValue, 5);
  String particlesInfo = "PM10: " + String(pm10) + " PM2.5: " + String(pm25);
  Serial.println(gpsInfo);

  // Mostrar en la pantalla
  display.drawString(10, 5, gpsInfo);          // Datos GPS
  display.drawString(10, 30, particlesInfo);   // Datos de partículas
  display.display();
}
```

Desarrollo código Arduino y medición de sensores

Continuamos con la función loop():

Aquí vamos a realizar la primera petición http al proyecto Java para, en este caso, enviar una cadena de status al servidor para hacerle saber si la conexión con el ESP-32 se ha interrumpido. Esto se realiza cada segundo.

```
if (WiFi.status() == WL_CONNECTED) {
  //Peticion http para revisar status del ESP-32
  HTTPClient httpSTATUS;

  httpSTATUS.begin(serverUrlStatus);
  httpSTATUS.addHeader("Content-Type", "application/json");
  String body = "{";
  body += "\"status\":\"CONNECTED\"";
  body += "}";

  Serial.println("JSON STATUS Enviado:");
  Serial.println(body);
  Serial.println("Sending HTTP POST request for STATUS...");
  int httpResponseCodeStatus = httpSTATUS.POST(body);

  // Verificar la respuesta del servidor
  if (httpResponseCodeStatus > 0) {
    String responseStatus = httpSTATUS.getString();
    Serial.println("Respuesta del servidor: " + responseStatus);
  } else {
    Serial.println("Error en la peticion: " + String(httpResponseCodeStatus));
  }
  httpSTATUS.end();

} else {
  Serial.println("WiFi Disconnected");
}
```

Desarrollo código Arduino y medición de sensores

Aquí realizamos la segunda petición http a la url definida comienzo del programa para recoger los datos medidos por los sensores en formato JSON.

Enviamos tanto, la latitud como la longitud medida por el GPS, como los valores de los contaminantes, así como la fecha y hora actual de la medida. Dicha fecha es obtenida mediante la función getCurrentTimestamp().

```
if (WiFi.status() == WL_CONNECTED && millis() - lastSensorDataMillis >= 30000) {

  lastSensorDataMillis = millis();

  //Petición HTTP para enviar los datos de los sensores
  HTTPClient http;

  // Configurar la URL del servidor
  http.begin(serverUrl);
  http.addHeader("Content-Type", "application/json");

  // Crear el cuerpo JSON
  String timestamp = getCurrentTimestamp();  // Obtener hora actual
  String jsonBody = "{";
  jsonBody += "\"measurementTimestamp\":\"" + String(timestamp) + "\",";
  jsonBody += "\"pm10\":" + String(pm10) + ",";
  jsonBody += "\"pm25\":" + String(pm25) + ",";
  jsonBody += "\"latitude\":\"" + String(latitudeValue, 6) + "\",";
  jsonBody += "\"longitude\":\"" + String(longitudeValue, 6) + "\"";
  jsonBody += "}";

  Serial.println("JSON Enviado:");
  Serial.println(jsonBody);

  // Enviar solicitud POST
  Serial.println("Sending HTTP POST request...");
  int httpResponseCode = http.POST(jsonBody);
```

```
  // Verificar respuesta
  if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    String response = http.getString();
    Serial.println("Response: " + response);
  } else {
    Serial.print("Error on sending POST: ");
    Serial.println(http.errorToString(httpResponseCode).c_str());
  }

  http.end();    // Finalizar conexión
} else {
  //Serial.println("WiFi Disconnected");
}
delay(1000);  // Esperar 1 segundos antes de enviar nuevamente
```

Desarrollo código Arduino y medición de sensores

Dicha función devuelve una cadena con la fecha y hora formateada en el formato YYYY-DD-MM T HH:mm:ss.

Para ello usamos la petición http al servidor NTP que nos devuelve la hora actual según los parámetros de configuración explicados anteriormente.

```
String getCurrentTimestamp() {
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Error: No se pudo obtener la hora actual.");
    return "";
  }

  char timestamp[25];
  strftime(timestamp, sizeof(timestamp), "%Y-%m-%dT%H:%M:%S", &timeinfo);
  // Mostrar fecha y hora
  Serial.print("Fecha y hora actual: ");
  Serial.println(String(timestamp));
  return String(timestamp);
}
```

Backend, Frontend Java y despliegue en servidor cloud

Inteligencia Ambiental

Backend (SpringBoot + MySQL)

Tal y como se ha explicado con anterioridad, la placa ESP32 realiza peticiones con el protocolo HTTP a un servidor dedicado (backend) para que este almacene y muestre los datos recopilados.

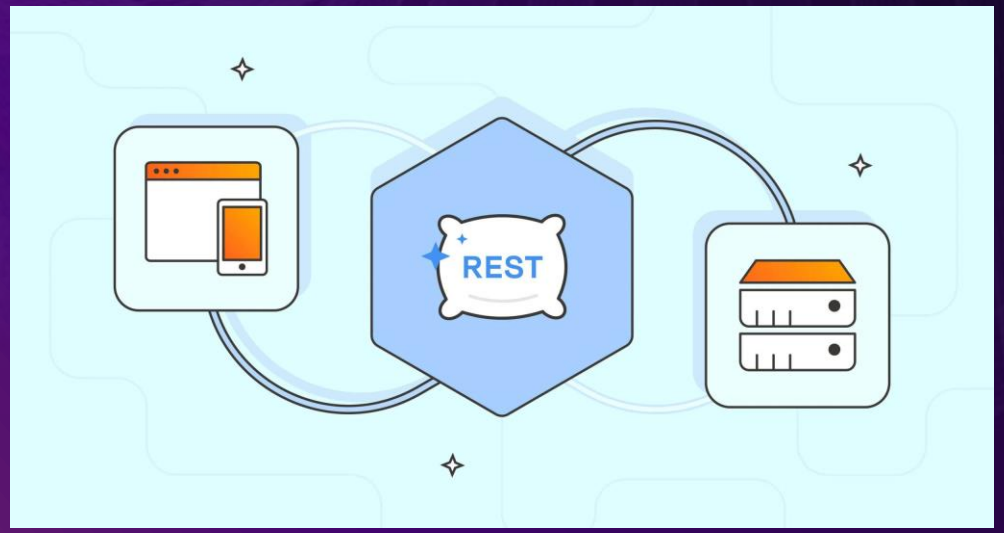
Para la elaboración de este backend se ha optado por el framework Spring de Java, más concretamente en su variante SpringBoot, que permite con gran facilidad el desarrollo de aplicaciones web. Además, se ha utilizado como base de datos MySQL.

Se han escogido ambas tecnologías por su gran extensión en el ámbito profesional y su fiabilidad.

Backend (SpringBoot + MySQL)

Se han seguido dos enfoques diferentes en cuanto a la arquitectura del proyecto:

- Arquitectura REST para la comunicación con la placa ESP32.
- Arquitectura SSR (Server Side Rendering) para la aplicación web.



Backend (SpringBoot + MySQL)

Spring es un framework que permite ambos tipos de arquitectura gracias a dos tipos de anotaciones a nivel de clase “@RestController” y “@Controller”.

Dichas anotaciones lo que definen es el tipo de “controlador” que es la clase con la que estamos trabajando. El framework otorga unas características distintas a cada tipo de controlador.

Un controlador no es mas que una clase Java que maneja peticiones HTTP a través de una o varias URLs definidas.

Backend (SpringBoot + MySQL)

Para la comunicación con la placa ESP32 se ha utilizado un @RestController, que como su nombre indica implementa un controlador siguiendo la arquitectura REST.

La arquitectura REST es desacoplada y carente de estado, el servidor desconoce el estado del cliente y viceversa, dependiendo exclusivamente del paso de mensajes, en este caso de tipo JSON (JavaScript Object Notation) mediante HTTP. Es un estándar muy utilizado en el ámbito web.

Mensaje JSON

spring boot

Backend (SpringBoot + MySQL)

La placa ESP32 envía mensajes en formato JSON al controlador de SpringBoot mediante una petición POST con los siguientes datos:

- measurementTimestamp: Fecha y hora en la que se ha hecho la medición
- pm10: Cantidad de partículas PM10 medida
- pm25: Cantidad de partículas PM2.5 medida
- latitude: latitud donde se ha tomado la medida (dada por el GPS)
- longitude: longitud donde se ha tomado la medida (dada por el GPS)

Estos mensajes se envían a la url del servidor "/save-particle-data" (por ejemplo, http://localhost:8080/save-particle-data) donde SpringBoot se encarga de transformar el mensaje JSON en una clase Java y posteriormente se insertan los datos en la base de datos MySQL

Backend (SpringBoot + MySQL)

```java
@RestController
public class ESP32DataReceiverController
{

    @Autowired
    private ParticleDataService particleDataService;

    @Autowired
    private ESP32ConnectionStatusService esp32ConnectionStatusService;

    @PostMapping("/save-particle-data")
    public String saveParticleData(@RequestBody ParticleData particleData)
    {
        particleDataService.saveParticleData(particleData);
        return "Particle data saved";
    }
```

```java
@Entity
@Table(name = "PARTICLE_DATA")
public class ParticleData
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(name = "measurement_timestamp")
    private Timestamp measurementTimestamp;

    @Column(name = "PM10")
    private Integer pm10;

    @Column(name = "PM2_5")
    private Integer pm25;

    @Column(name = "latitude")
    private String latitude;

    @Column(name = "longitude")
    private String longitude;
```

Backend (SpringBoot + MySQL)

Mensaje JSON

```
{
   measurementTimestamp = "2025-01-10T15:30:00"
   pm10 = 42
   pm25 = 35
   latitude = 37.7749
   longitude = -122.4194
}
```

Se envía mediante una petición POST HTTP al controlador de SpringBoot

Se insertan los datos en la Base de Datos MySQL

MySQL

spring boot

Frontend (SpringBoot + Thymeleaf)

SpringBoot implementa de forma nativa el patrón de diseño MVC (Modelo-Vista-Contolador) que es el utilizado junto a la arquitectura SSR (Server Side Rendering) para mostrar los datos mediante una aplicación web.

SSR no es más que una arquitectura enfocada a que todo el contenido HTML se genere en el servidor y se sirva de forma directa al cliente, otorga menos dinamismo que otras arquitecturas como REST, aunque se pueden combinar como se detallará próximamente.

Para generar el contenido HTML desde el servidor, y siguiendo el patrón MVC se ha utilizado el motor de plantillas Thymeleaf que es capaz de comunicarse con objetos Java.

Thymeleaf

Frontend (SpringBoot + Thymeleaf)

Todo el frontend de la aplicación web ha sido implementado de esta forma, poniendo de ejemplo la URL principal:

```java
@Controller
public class ESP32ParticleVisualizationController
{
    @Autowired
    private ParticleDataService particleDataService;

    @Autowired
    private ESP32ConnectionStatusService esp32ConnectionStatusService;

    @GetMapping("/")
    public String getESP32ParticleVisualizationWelcomePage(Model model)
    {
        ESP32ConnectionStatus esp32ConnectionStatus = esp32ConnectionStatusService.getESP32ConnectionStatus();

        if(esp32ConnectionStatus.getStatus().equals(ESP32ConnectionStatus.STATUS_CONNECTED))
        {
            model.addAttribute("status", ESP32ConnectionStatus.STATUS_CONNECTED);
            model.addAttribute("esp32_status", "conectado");
            model.addAttribute("esp32_status_color", "green");
        }
        else if(esp32ConnectionStatus.getStatus().equals(ESP32ConnectionStatus.STATUS_DISCONNECTED))
        {
            model.addAttribute("status", ESP32ConnectionStatus.STATUS_DISCONNECTED);
            model.addAttribute("esp32_status", "desconectado");
            model.addAttribute("esp32_status_color", "red");
        }

        return "particle_visualization";
    }
```

En la página principal se indica si la placa se encuentra conectada y midiendo.

Esto se hace de forma dinámica añadiendo el estado que se encuentra en la base de datos como atributo al modelo (Model model).

En función del estado también se utilizará un texto (conectado/desconectado) y un color distinto, que también se transfieren al modelo (plantilla Thymeleaf) como atributos.

Frontend (SpringBoot + Thymeleaf)

En la plantilla Thymeleaf estos atributos se recogen mediante propiedades que comienzan con el prefijo “th:”. Thymeleaf también permite sentencias de muchos tipos, como por ejemplo condicionales como “th:if” tal y como se aprecia en el ejemplo.

```
<div class="welcome-content-div">
    <p>¡Bienvenido!</p>
    <p>
        Aquí podras consultar los datos de lecturas de partículas realizado con la placa
        Heltec ESP32 LoRa V2.
    </p>
    <img th:if="${status == 'DISCONNECTED'}" class="esp32-image" th:src="@{/images/esp32_offline.png}" alt="Heltec ESP32 offline image"/>
    <img th:if="${status == 'CONNECTED'}" class="esp32-image" th:src="@{/images/esp32_online.png}" alt="Heltec ESP32 online image"/>
    <p>
        Estado del dispositivo: <span th:style="'color: ' + ${esp32_status_color}" th:text="${esp32_status}"></span>
    </p>
</div>
```

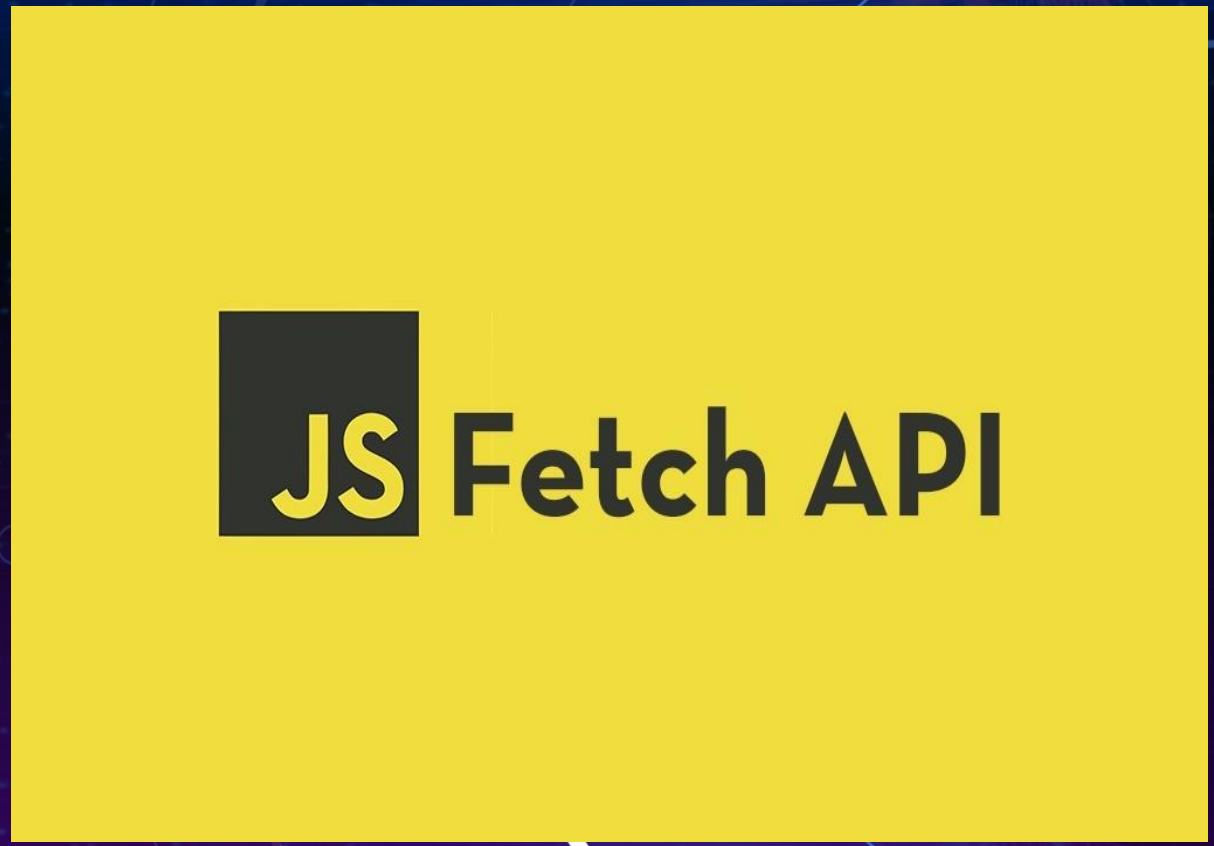
En función del estado se mostrará una imagen u otra en tiempo real cada vez que se cargue la página.

Frontend (SpringBoot + Thymeleaf)

Además de SSR para la aplicación web se ha utilizado también en ciertas partes de la misma arquitectura REST para cargar los datos de partículas en tiempo real.

Es el caso por ejemplo del área de datos recientes (url: /datos-recientes). Esta área está diseñada para visualizar en tiempo real las mediciones que la placa ESP32 realice, por lo cual es necesario que se recarguen los datos cada ciertos segundos.

Esto sería imposible de lograr mediante Server Side Rendering, ya que implicaría la acción manual del usuario al recargar la página. Para hacerlo dinámico es necesario valerse de la API FETCH de JavaScript, que permite la ejecución de peticiones HTTP al servidor.



Frontend (SpringBoot + Thymeleaf)

```javascript
fetch('/get-recent-particle-data')
        .then(response => {
            if (!response.ok) {
                throw new Error(`Error: ${response.status}`);
            }
            return response.json();
        })
        .then(data => {
            console.log('Datos recibidos:', data);

            particleDataTable = document.getElementById("particle_data_table");
            particleDataTableBody = particleDataTable.getElementsByTagName("tbody")[0];

            //We first delete all child nodes of particle data table
            particleDataTableBody.innerHTML = '';

            for (let i = 0; i < data.length; i++)
            {
                tableRow = createTableRow(data[i]);

                if(i % 2 == 0)
                {
                    tableRow.classList.add("particle_data_table_odd_row");
                }

                particleDataTableBody.appendChild(tableRow);
            }
        })
        .catch(error => {
            console.error('Error al realizar la petición:', error);
        });
```

```java
@GetMapping("/get-recent-particle-data")
public List<ParticleData> getMostRecentParticleData()
{
    return particleDataService.findMostRecentData();
}
```

```java
@Transactional(readOnly = false)
public List<ParticleData> findMostRecentData()
{
    return this.particleDataRepository.findTop50ByOrderByMeasurementTimestampDesc();
}
```

Frontend (SpringBoot + Thymeleaf)

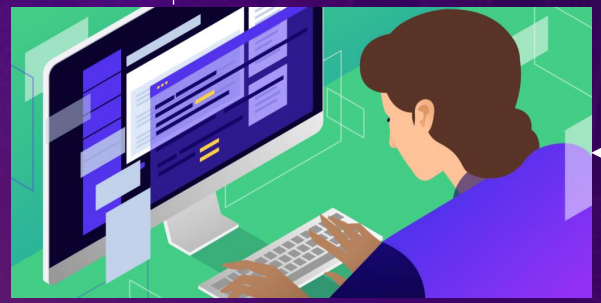
Realiza petición GET mediante API FETCH de los datos más recientes

Convierte los datos recientes a formato JSON y los envía al frontend

Consulta los datos más recientes

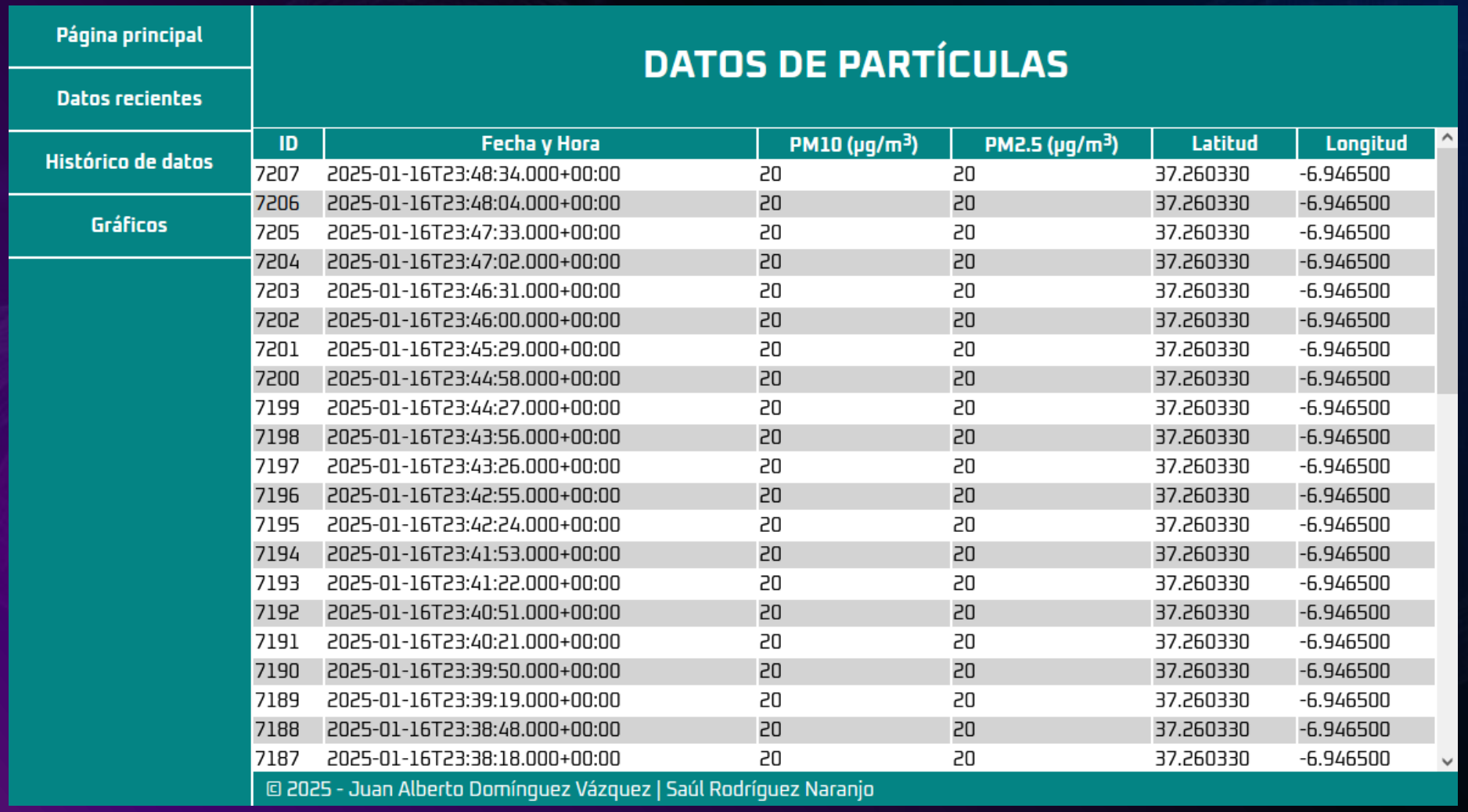
Devuelve los datos más recientes al servidor

Solicita datos recientes



Recarga el contenido HTML de forma dinámica para mostrar los datos recibidos

Frontend (SpringBoot + Thymeleaf)



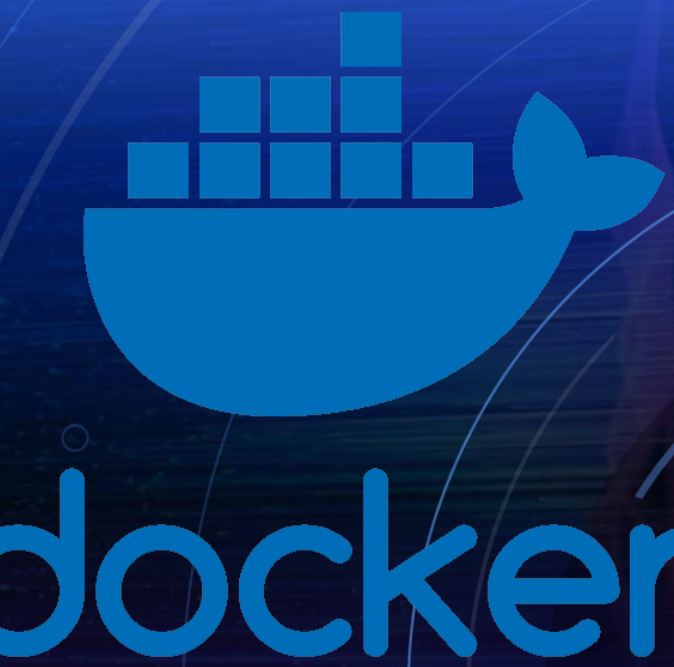
Despliegue en la nube

Para mayor facilidad a la hora de comunicar la placa ESP32 con nuestra aplicación web y por ende recopilar los datos con mayor eficacia se ha optado por desplegar tanto la base de datos como la aplicación en sí misma en la nube.

Nos hemos valido para ello de un servidor VPS (Virtual Private Server) contratado con la empresa IONOS con Ubuntu 22.04 como sistema operativo.

La base de datos MySQL ha sido desplegada mediante un contenedor Docker ya que así residirá en un entorno reducido y aislado sin afectar al sistema operativo del servidor VPS.

La aplicación SpringBoot se ha desplegado usando la utilidad systemctl de Ubuntu, para inicializar servicios del sistema.



Despliegue en la nube

Docker es un proyecto de código abierto que permite la utilización de máquinas virtuales muy reducidas y livianas para aislar un entorno concreto preparado para una aplicación concreta, es decir, juntar la aplicación con todas sus dependencias sin afectar al sistema operativo principal.

Estas máquinas virtuales reducidas es lo que se conoce como "contenedores". Para instalar Docker es necesario consultar su documentación: https://docs.docker.com/engine/install/

Una vez instalado Docker, la puesta en marcha del contenedor MySQL se llevaría a cabo con los siguientes comandos:

- *$ docker pull mysql:latest* → Descarga la imagen de la última versión disponible
- *$ docker run --name test-mysql -e MYSQL_ROOT_PASSWORD=strong_password -d mysql* → Crea y ejecuta un contenedor llamado test-mysql utilizando la imagen "mysql" anteriormente descargada

Despliegue en la nube

En cuanto a la aplicación web con SpringBoot, se ha creado un repositorio en github que posteriormente se ha clonado en el servidor VPS. Una vez clonado para desplegarlo como servicio del sistema mediante systemctl se realizan los siguientes pasos:

- *mvn package* → Se compila el proyecto Java con Maven
- *sudo nano /etc/systemd/system/springboot-esp32.service* → Se crea un fichero en la ruta /etc/systemd/system/ con el nombre del servicio y la extensión .service
- *sudo systemctl start springboot-esp32.service* → Se inicia el servicio creado

El contenido del fichero springboot-esp32.service es el siguiente:

```
[Unit]
Description=Spring Boot Application for ESP32
After=network.target

[Service]
User=root
ExecStart=/opt/jdk-21.0.5+11/bin/java -jar /esp32/ESP32-Detector-Particulas/target/esp32-0.0.1-SNAPSHOT.jar
SuccessExitStatus=143
TimeoutStopSec=10
Restart=always

[Install]
WantedBy=multi-user.target
```

Despliegue en la nube

Para revisar los logs del servicio utilizaremos el comando *journalctl -u springboot-esp32.service.* El servidor se encuentra disponible en la url http://212.227.87.151:8080/

```
Jan 15 02:29:40 ubuntu java[3805473]:   .   ____          _            __ _ _
Jan 15 02:29:40 ubuntu java[3805473]:  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
Jan 15 02:29:40 ubuntu java[3805473]: ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
Jan 15 02:29:40 ubuntu java[3805473]:  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
Jan 15 02:29:40 ubuntu java[3805473]:   '  |____| .__|_| |_|_| |_\__, | / / / /
Jan 15 02:29:40 ubuntu java[3805473]:  =========|_|==============|___/=/_/_/_/
Jan 15 02:29:40 ubuntu java[3805473]:  :: Spring Boot ::                (v3.4.1)
Jan 15 02:29:40 ubuntu java[3805473]: 2025-01-15T02:29:40.207+01:00  INFO 3805473 --- [esp32] [           main] com.uhu.esp32.Esp32Application          : Starting Esp32Application v0.0.1-SNAPSHOT using Java 21.0.5 with PID 3805473 (/es
Jan 15 02:29:40 ubuntu java[3805473]: 2025-01-15T02:29:40.210+01:00  INFO 3805473 --- [esp32] [           main] com.uhu.esp32.Esp32Application          : No active profile set, falling back to 1 default profile: "default"
Jan 15 02:29:40 ubuntu java[3805473]: 2025-01-15T02:29:40.843+01:00  INFO 3805473 --- [esp32] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
Jan 15 02:29:40 ubuntu java[3805473]: 2025-01-15T02:29:40.899+01:00  INFO 3805473 --- [esp32] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 47 ms. Found 2 JPA repository interfa
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.397+01:00  INFO 3805473 --- [esp32] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.410+01:00  INFO 3805473 --- [esp32] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.411+01:00  INFO 3805473 --- [esp32] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.34]
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.436+01:00  INFO 3805473 --- [esp32] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.436+01:00  INFO 3805473 --- [esp32] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1166 ms
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.560+01:00  INFO 3805473 --- [esp32] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.722+01:00  INFO 3805473 --- [esp32] [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@15639d09
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.724+01:00  INFO 3805473 --- [esp32] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.768+01:00  INFO 3805473 --- [esp32] [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.812+01:00  INFO 3805473 --- [esp32] [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.6.4.Final
Jan 15 02:29:41 ubuntu java[3805473]: 2025-01-15T02:29:41.850+01:00  INFO 3805473 --- [esp32] [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
Jan 15 02:29:42 ubuntu java[3805473]: 2025-01-15T02:29:42.071+01:00  INFO 3805473 --- [esp32] [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
Jan 15 02:29:42 ubuntu java[3805473]: 2025-01-15T02:29:42.138+01:00  WARN 3805473 --- [esp32] [           main] org.hibernate.dialect.Dialect            : HHH000511: The 5.6.46 version for [org.hibernate.dialect.MySQLDialect] is no long
Jan 15 02:29:42 ubuntu java[3805473]: 2025-01-15T02:29:42.151+01:00  INFO 3805473 --- [esp32] [           main] org.hibernate.orm.connections.pooling    : HHH10001005: Database info:
Jan 15 02:29:42 ubuntu java[3805473]:         Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)']
Jan 15 02:29:42 ubuntu java[3805473]:         Database driver: undefined/unknown
Jan 15 02:29:42 ubuntu java[3805473]:         Database version: 5.6.46
Jan 15 02:29:42 ubuntu java[3805473]:         Autocommit mode: undefined/unknown
Jan 15 02:29:42 ubuntu java[3805473]:         Isolation level: undefined/unknown
Jan 15 02:29:42 ubuntu java[3805473]:         Minimum pool size: undefined/unknown
Jan 15 02:29:42 ubuntu java[3805473]:         Maximum pool size: undefined/unknown
Jan 15 02:29:42 ubuntu java[3805473]: 2025-01-15T02:29:42.737+01:00  INFO 3805473 --- [esp32] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to
Jan 15 02:29:42 ubuntu java[3805473]: 2025-01-15T02:29:42.740+01:00  INFO 3805473 --- [esp32] [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
Jan 15 02:29:42 ubuntu java[3805473]: 2025-01-15T02:29:42.937+01:00  INFO 3805473 --- [esp32] [           main] o.s.d.j.r.query.QueryEnhancerFactory     : Hibernate is in classpath; If applicable, HQL parser will be used.
Jan 15 02:29:43 ubuntu java[3805473]: 2025-01-15T02:29:43.474+01:00  WARN 3805473 --- [esp32] [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be
Jan 15 02:29:43 ubuntu java[3805473]: 2025-01-15T02:29:43.504+01:00  WARN 3805473 --- [esp32] [           main] .s.s.UserDetailsServiceAutoConfiguration :
Jan 15 02:29:43 ubuntu java[3805473]: Using generated security password: e367d8b8-46e7-446e-9f0c-c5907410e77c
Jan 15 02:29:43 ubuntu java[3805473]: This generated password is for development use only. Your security configuration must be updated before running your application in production.
Jan 15 02:29:43 ubuntu java[3805473]: 2025-01-15T02:29:43.526+01:00  INFO 3805473 --- [esp32] [           main] r$InitializeUserDetailsManagerConfigurer : Global AuthenticationManager configured with UserDetailsService bean with name in
Jan 15 02:29:43 ubuntu java[3805473]: 2025-01-15T02:29:43.985+01:00  INFO 3805473 --- [esp32] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
Jan 15 02:29:44 ubuntu java[3805473]: 2025-01-15T02:29:44.000+01:00  INFO 3805473 --- [esp32] [           main] com.uhu.esp32.Esp32Application          : Started Esp32Application in 4.153 seconds (process running for 4.62)
```

CONCLUSIONES

Inteligencia Ambiental

CONCLUSIONES

- Como valoraciones finales del proyecto, podemos decir que hemos aprendido a inicializar, configurar y usar distintos sensores incrustados en una placa mediante un microcontrolador como es el ESP-32 que ha servido de centro neurálgico de todo el proyecto.
- Por otro lado, y para darle más valor al proyecto, el hecho de haber usado un servidor en el cloud, nos ha ofrecido la posibilidad de aprender a desarrollar y desplegar un proyecto web en Java en dicho entorno, cosa que hasta ahora nuca habíamos hecho y en combinación con el programa en Arduino nos ha dado una valiosa experiencia para nuestra carrera.

FIN

Muchas gracias