```
// Importación de bibliotecas necesarias

using System.Collections.Generic;  // Para usar listas y colecciones

using UnityEngine;  // Biblioteca principal de Unity

using TMPro;  // Para trabajar con TextMeshPro (texto UI avanzado)

using Newtonsoft.Json;  // Para la serialización/deserialización JSON

using UnityEngine.UI;  // Para trabajar con elementos de la interfaz de usuario

using uPLibrary.Networking.M2Mqtt;  // Cliente MQTT para la comunicación

using uPLibrary.Networking.M2Mqtt.Messages;  // Manejo de mensajes MQTT

using System.Text;  // Para la codificación de texto


// Clase principal que maneja la comunicación con un dispositivo ESP32 y procesa los datos recibidos

public class enviodedatos : MonoBehaviour

{

    // public paneles panelesScript; // Referencia comentada a otro script (no utilizada actualmente)



    // Referencia al texto UI que mostrará información sobre la repetición del ejercicio

    public TextMeshProUGUI infoRepeticionText;
```

```
// Evento que se dispara cuando los datos han sido procesados

public event System.Action<List<List<float>>> OnDatosProcesados;



// Referencia al administrador de secuencias de ejercicios

public secuenciaManager secuencia;



// Cliente MQTT para la comunicación

private MqttClient client;



// Dirección del broker MQTT (servidor intermediario)

private string brokerAddress = "broker.emqx.io";



// Puerto del broker MQTT

private int brokerPort = 1883;



// Tema para publicar mensajes hacia el ESP32

private string topicPublish = "Unity";
```

```
// Tema para suscribirse y recibir mensajes del ESP32

private string topicSubscribe = "ESP32MPU";



// Estado actual del sistema (0: detenido, 1: capturando datos)

private int state = 0;



// Botón para alternar entre estados de captura

public Button toggleButton;



// Texto del botón que cambiará según el estado

public Text buttonText;



// Panel que se muestra para validar el ejercicio

public GameObject panelValidacion;



// Indica si se están recopilando datos actualmente

private bool recopilandoDatos = false;
```

```
    // Lista para almacenar los datos acumulados del sensor
    private List<List<float>> datosAcumulados = new List<List<float>>();

    // Método que se ejecuta al iniciar el script
    void Start()
    {
        // Inicialización del cliente MQTT
        client = new MqttClient(brokerAddress, brokerPort, false, null, null, MqttSslProtocols.None);

        // Registro del método que se llamará cuando se reciba un mensaje
        client.MqttMsgPublishReceived += OnMessageReceived;

        // Creación de un ID de cliente único
        string clientId = "unityClient_" + System.Guid.NewGuid().ToString();

        // Conexión al broker MQTT
        client.Connect(clientId);
```

```
        // Suscripción al tema para recibir datos del ESP32

        client.Subscribe(new string[] { topicSubscribe }, new byte[] {
MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE });



        // Mensaje de confirmación de conexión

        Debug.Log("Conectado al servidor MQTT");



        // Asignación del método que responderá al clic del botón

        toggleButton.onClick.AddListener(ToggleState);

    }


    // Método que alterna el estado de captura de datos

    void ToggleState()

    {

        // Cambia el estado entre 0 y 1

        state = (state == 0) ? 1 : 0;



        // Convierte el estado a texto para enviarlo
```

```
        string message = state.ToString();



        // Publica el mensaje al ESP32

        client.Publish(topicPublish,                Encoding.UTF8.GetBytes(message),
MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE, false);



        // Confirmación de publicación del mensaje

        Debug.Log("Mensaje publicado: " + message);



        if (state == 1)

        {

            // Cuando el estado es 1, inicia la recopilación de datos

            recopilandoDatos = true;

            datosAcumulados.Clear();  // Limpia datos anteriores

            Debug.Log("Iniciando recopilación de datos...");

        }

        else if (state == 0)

        {

            // Cuando el estado es 0, finaliza la recopilación y procesa los datos
```

```
            recopilandoDatos = false;

            Debug.Log("Finalizando recolección de datos, activando procesamiento...");

            ProcesarDatos(datosAcumulados);  // Procesa los datos acumulados

        }


        // Actualiza el texto del botón según el estado

        if (buttonText.text == "Evaluar")

        {

            buttonText.text = "Pausar";

        }

        else

        {

            buttonText.text = "Evaluar";

            panelValidacion.SetActive(true);  // Muestra el panel de validación

            // Actualiza el texto informativo con detalles de la sesión y ejercicio

            infoRepeticionText.text = $"Se realizó correctamente la serie. {secuencia.sesionActual} del ejercicio {secuencia.ejercicios[secuencia.ejercicioActualIndex]} ";

        }
```

```
    }

    // Método que se llama cuando se recibe un mensaje MQTT
    void OnMessageReceived(object sender, MqttMsgPublishEventArgs e)
    {
        // Convierte el mensaje recibido de bytes a texto
        string message = Encoding.UTF8.GetString(e.Message);
        Debug.Log("Mensaje recibido del ESP: " + message);

        if (recopilandoDatos)
        {
            try
            {
                // Intenta deserializar el mensaje JSON a la clase SensorData
                SensorData sensorData = JsonConvert.DeserializeObject<SensorData>(message);
                if (sensorData != null && sensorData.data != null && sensorData.data.Count > 0)
                {
                    // Agrega los datos recibidos a la lista acumulada
```

```
                datosAcumulados.AddRange(sensorData.data);
            }
        }
        catch (System.Exception ex)
        {
            // Registra cualquier error durante el procesamiento del mensaje
            Debug.LogError("Error al procesar el mensaje MQTT: " + ex.Message);
        }
    }
}

// Método para procesar los datos acumulados
void ProcesarDatos(List<List<float>> datos)
{
    // Verifica que haya datos para procesar
    if (datos.Count == 0)
    {
        Debug.LogError("Error: No hay datos para procesar.");
```

```
    return;
}

// Crea listas para cada tipo de dato del sensor
List<float> pitch1 = new List<float>();
List<float> roll1 = new List<float>();
List<float> yaw1 = new List<float>();
List<float> pitch2 = new List<float>();
List<float> roll2 = new List<float>();
List<float> yaw2 = new List<float>();
List<float> emg = new List<float>();
List<float> tiempo = new List<float>();

// Procesa cada fila de datos y los separa en las listas correspondientes
foreach (var fila in datos)
{
    if (fila.Count == 8)
    {
        pitch1.Add(fila[0]);
```

```
            roll1.Add(fila[1]);
            yaw1.Add(fila[2]);
            pitch2.Add(fila[3]);
            roll2.Add(fila[4]);
            yaw2.Add(fila[5]);
            emg.Add(fila[6]);
            tiempo.Add(fila[7]);
        }
        else
        {
            // Error si la fila no tiene los 8 elementos esperados
            Debug.LogError("Error: Se esperaba un vector de 8 elementos.");
            return;
        }
    }

    // Si hay suscriptores al evento, notifica con los datos procesados
    if (OnDatosProcesados != null)
```

```
        {
            Debug.Log("Llamando a ActualizarDatos en secuencia manager...");
            OnDatosProcesados(new List<List<float>> { pitch1, roll1, yaw1, pitch2, roll2, yaw2, emg, tiempo });
        }
        else
        {
            Debug.LogError("Secuencia manager no está asignado.");
        }
    }

    // Método para reiniciar la captura de datos
    public void ReiniciarCaptura()
    {
        state = 1;  // Activa el estado de captura
        recopilandoDatos = true;
        datosAcumulados.Clear();  // Limpia los datos anteriores
        client.Publish(topicPublish, Encoding.UTF8.GetBytes("1"), MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE, false);
```

```
    Debug.Log("Captura reiniciada para repetir la sesión.");
}

// Método que se ejecuta cuando se destruye el objeto
void OnDestroy()
{
    // Desconecta el cliente MQTT si está conectado
    if (client != null && client.IsConnected)
    {
        client.Disconnect();
    }
}

// Clase interna para deserializar los datos del sensor
[System.Serializable]
public class SensorData
{
    public List<List<float>> data;  // Lista de listas que contiene los datos del sensor
}
```

```
}
```