

# Aplicaciones Móviles IOT: Unidad 2

**Asignatura:** “Aplicaciones Móviles para IoT”

**Sección:** "TI3042/D-B50-N4-P13-C1(F)/D"

**Nombre del docente:** David Eduardo Méndez Jeldres

**Nombre de los integrantes del grupo:** Orlando Aravena



29 de octubre de 2025

<b>Introducción</b>	<b>4</b>
<b>Problema</b>	<b>4</b>
<b>Dispositivos a usar:</b>	<b>4</b>
<b>Funcionalidades:</b>	<b>4</b>
<b>Pantalla de inicio de sesión</b>	<b>5</b>
<b>Pantalla de Registro</b>	<b>6</b>
<b>Selección de Conexión</b>	<b>7</b>
<b>Conexión por Bluetooth</b>	<b>7</b>
<b>Conexión por Wifi</b>	<b>10</b>
<b>Repositorio de Github</b>	<b>12</b>



# Introducción

El objetivo principal de este proyecto es dar solución a una problemática mediante el desarrollo de una aplicación móvil híbrida para la plataforma Android. Esta aplicación ofrece un sistema de mensajería versátil capaz de operar en dos modos de conexión distintos, garantizando la comunicación bajo diferentes condiciones.

## Problema

Las personas necesitan una forma de comunicarse en situaciones donde no hay Internet (por ejemplo, en una excursión o en un desastre) o a través de Internet de forma privada.

## Dispositivos a usar:

En práctica 2 celulares android, pero para el objetivo de esta evaluación se hará la conexión entre un celular y un dispositivo IoT.

## Funcionalidades:

- Enviar un mensaje de texto.
- Recibir un mensaje de texto.
- Conectarse a otro dispositivo (sea por Bluetooth o Internet).
- Registrar un usuario y contraseña
- Iniciar sesión con un usuario y contraseña.

A continuación se dará una explicación de cada pantalla y de las partes del código más relevantes o que requieren un conocimiento más técnico.

## Pantalla de inicio de sesión

La pantalla de inicio es intuitiva y estándar, cada variable explica su utilidad.

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val campoUsuario = findViewById<EditText>(R.id.editTextUsuario)
        val campoContrasena = findViewById<EditText>(R.id.editTextContrasena)
        val botonDeLogin = findViewById<Button>(R.id.botonLogin)
        val botonIrARegistro = findViewById<Button>(R.id.botonIrARegistro)

        botonDeLogin.setOnClickListener {
            val usuarioEscrito = campoUsuario.text.toString()
            val contrasenaEscrita = campoContrasena.text.toString()

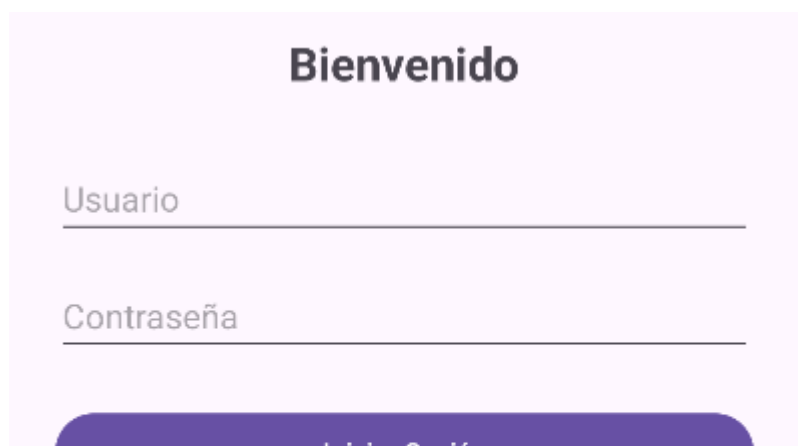
            if (validarCredenciales(usuarioEscrito, contrasenaEscrita)) {

                guardarUsuarioLogueado(usuarioEscrito)

                val intent = Intent(this, SelectConnection::class.java)
                startActivity(intent)
                finish()
            } else {
                Toast.makeText(this, "Error: Usuario o contraseña incorrectos", Toast.LENGTH_LONG).show()
            }
        }

        botonIrARegistro.setOnClickListener {
            val intent = Intent(this, RegisterActivity::class.java)
            startActivity(intent)
        }
    }
}
```

Y se ve de esta manera:



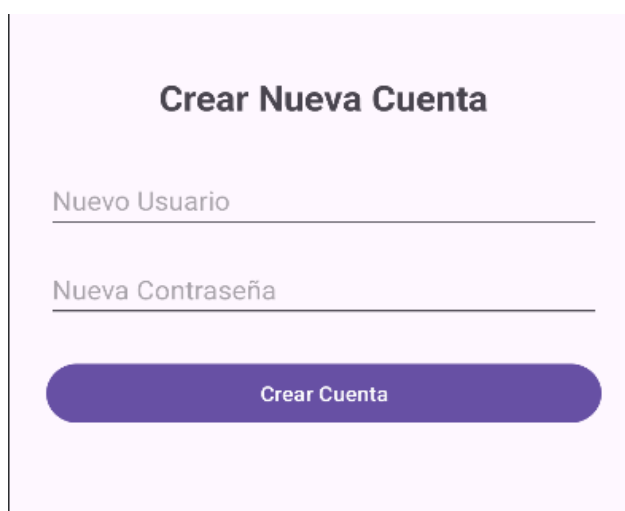
The screenshot shows a mobile application interface for a login screen. At the top, there is a dark blue header with the word "Bienvenido" in white. Below the header, there are two input fields: "Usuario" and "Contraseña", both with light blue borders and placeholder text. At the bottom, there is a large, rounded blue button with the text "Iniciar Sesión" in white.

# Pantalla de Registro

Para registrar un nuevo usuario utilizamos herramientas de seguridad, en concreto, encriptación con MasterKey y EncryptedSharedPreferences para guardar los datos.

```
try {  
  
    val masterKeyAlias = MasterKeys.getOrCreate(MasterKeys.AES256_GCM_SPEC)  
  
    val sharedPreferences = EncryptedSharedPreferences.create(  
  
        "auth_prefs",  
  
        masterKeyAlias,  
  
        this,  
  
        EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,  
  
        EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM  
  
    )  
  
    with(sharedPreferences.edit()) {  
  
        putString(nuevoUsuario, nuevaContraseña)  
  
        apply()  
  
    }  
  
    Toast.makeText(this, "¡Cuenta registrada exitosamente!",  
    Toast.LENGTH_SHORT).show()  
  
    finish()  
  
}
```

Por lo demás, es muy parecido al inicio de sesión.



**Crear Nueva Cuenta**

Nuevo Usuario

Nueva Contraseña

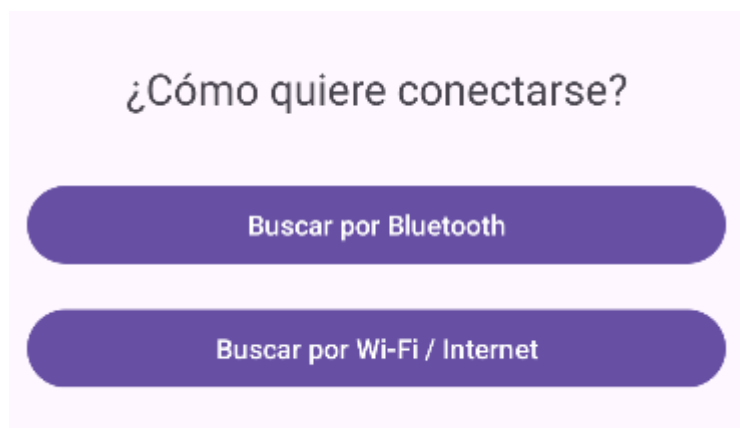
Crear Cuenta

# Selección de Conexión

Esta vista se utiliza para seleccionar de qué forma se conectará con el dispositivo foráneo.

```
botonBluetooth.setOnClickListener {  
    val intent = Intent(this, DeviceListActivity::class.java)  
    startActivity(intent)  
}  
  
botonWifi.setOnClickListener {  
    val intent = Intent(this, MqttChatActivity::class.java)  
    startActivity(intent)  
}  
}
```

Muy autoexplicativa.



## Conexión por Bluetooth

Para la conexión Bluetooth, la aplicación primero requiere permisos de **BLUETOOTH\_SCAN** y **BLUETOOTH\_CONNECT** para Android 12 y versiones superiores, o **ACCESS\_FINE\_LOCATION** y **BLUETOOTH\_ADMIN** para versiones anteriores, ya que la búsqueda de dispositivos Bluetooth se considera una operación de localización.

Estos permisos se piden a través de AndroidManifest

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission
    android:name="android.permission.BLUETOOTH"
    android:maxSdkVersion="30" />
<uses-permission
    android:name="android.permission.BLUETOOTH_ADMIN"
    android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

La función BroadcastReceiver() es la más importante de la conexión bluetooth y funciona como la antena que recibe la señal para encontrar otros dispositivos Bluetooth.

```
private val receptorBluetooth = object : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val action: String? = intent.action
        when(action) {
            BluetoothDevice.ACTION_FOUND -> {
                val device: BluetoothDevice? =
                    intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE)
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                    if (ActivityCompat.checkSelfPermission(context,
                        Manifest.permission.BLUETOOTH_CONNECT) != PackageManager.PERMISSION_GRANTED)
                        return
                }
                val deviceName = device?.name ?: "Dispositivo Desconocido"
                val deviceAddress = device?.address

                val deviceInfo = "$deviceName\n$deviceAddress"
                if (!listaDispositivosEncontrados.contains(deviceInfo)) {
                    listaDispositivosEncontrados.add(deviceInfo)
                    Log.i(TAG, "Dispositivo encontrado: $deviceInfo")
                    actualizarLista()
                }
            }
            BluetoothAdapter.ACTION_DISCOVERY_FINISHED -> {
                Log.i(TAG, "¡Escaneo finalizado!")
                Toast.makeText(context, "Escaneo finalizado",
                    Toast.LENGTH_SHORT).show()
                if (listaDispositivosEncontrados.isEmpty()) {
                    Toast.makeText(context, "No se encontraron dispositivos",
                        Toast.LENGTH_SHORT).show()
                }
            }
        }
    }
}
```



En la siguiente función se toman los dispositivos encontrados para mostrarlos en la vista.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_device_list)

    listViewDispositivos = findViewById(R.id.listViewDispositivos)
    botonRefrescar = findViewById(R.id.botonRefrescar)

    adapter = ArrayAdapter(this, android.R.layout.simple_list_item_2,
        android.R.id.text1, listaDispositivosEncontrados)
    listViewDispositivos.adapter = adapter

    val filter = IntentFilter(BluetoothDevice.ACTION_FOUND)
    filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED)
    registerReceiver(receptorBluetooth, filter)
    Log.d(TAG, "Receptor (BroadcastReceiver) registrado")

    botonRefrescar.setOnClickListener {
        Log.d(TAG, "Botón Refrescar presionado")
        listaDispositivosEncontrados.clear()
        actualizarLista()
        revisarYPedirPermisosBluetooth()
    }

    listViewDispositivos.setOnItemClickListener { parent, view, position, id
->

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            if (ActivityCompat.checkSelfPermission(this,
                Manifest.permission.BLUETOOTH_SCAN) != PackageManager.PERMISSION_GRANTED)
                return@setOnItemClickListener
            }
            bluetoothAdapter?.cancelDiscovery()
            Log.d(TAG, "Escaneo cancelado por clic de usuario.")

            val infoCompleta = listaDispositivosEncontrados[position]
            val direccionMac = infoCompleta.takeLast(17)

            Toast.makeText(this, "Conectando a: $direccionMac",
                Toast.LENGTH_SHORT).show()

            val intent = Intent(this, ChatActivity::class.java).apply {
                putExtra("DIRECCION_MAC", direccionMac)
            }
            startActivity(intent)
        }
    }
```

La lista de dispositivos se ve así

(No tenia dispositivos para buscar así que muestro la pantalla xml)

Dispositivos Encontrados	Refrescar
Item 1 Sub Item 1	
Item 2 Sub Item 2	
Item 3 Sub Item 3	
Item 4 Sub Item 4	
Item 5 Sub Item 5	
Item 6 Sub Item 6	
Item 7 Sub Item 7	
Item 8 Sub Item 8	
Item 9 Sub Item 9	
Item 10 Sub Item 10	

## Conexión por Wifi

La conexión por wifi se salta la búsqueda de dispositivos y genera una sala de texto con hivemq que actúa como servidor para comunicarse con otros usuarios con el mismo tópico.

Aquí dibuja la pantalla y botones

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_mqtt_chat)  
  
    chatHistory = findViewById(R.id.textViewChatHistory)  
    messageBox = findViewById(R.id.editTextMensaje)  
    sendButton = findViewById(R.id.botonEnviar)
```

Averigua quien usa la app

```
val prefs = getSharedPreferences("app_prefs", Context.MODE_PRIVATE)
myUsername = prefs.getString("LOGGED_IN_USER", "Usuario") ?: "Usuario"
```

Genera un ID aleatorio

```
myClientId = UUID.randomUUID().toString()
```

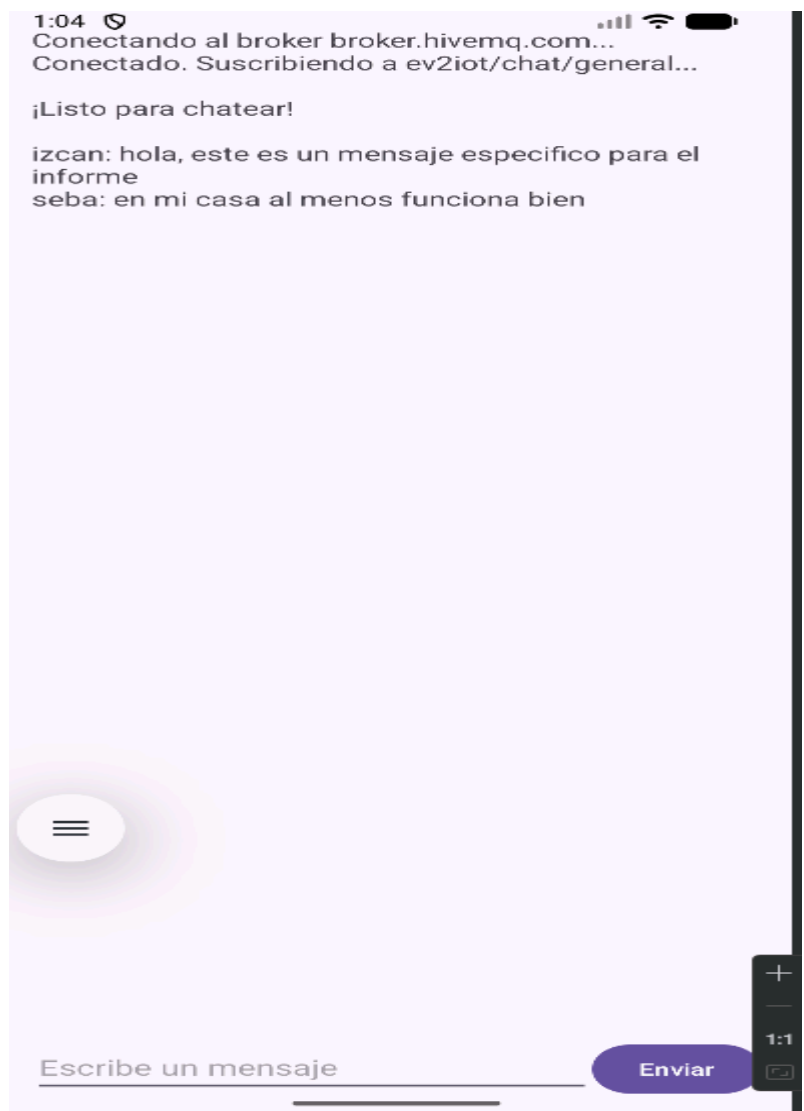
Se arma el servidor

```
client = MqttClient.builder()
    .useMqttVersion5()
    .serverHost(brokerUri)
    .serverPort(1883)
    .identifier(myClientId)
    .buildAsync()
```

Inicia conexión

```
connectAndSubscribe()
}
```

Este es el resultado Final:



# Repositorio de Github

Mi repositorio de Github para revisar los documentos utilizados para el informe: <https://github.com/1zcan/app-IoT>