Android i Bluetooth Low Energy

Laboratorium 8

Serwer

- *Tworzymy nowy projekt typu EmptyActivity dla Androida 6 o nazwie BLEServer*
- *Tworzymy te same wpisy w manifeście i dodajemy klasę Constants*
- *Serwer będzie dosyć podobny do klienta*

Serwer

- *Potrzebne importy*

```
import android.bluetooth.*
import android.bluetooth.le.BluetoothLeAdvertiser
import android.content.Context
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.content.pm.PackageManager
import android.content.Intent
import android.bluetooth.le.AdvertiseSettings
import android.bluetooth.le.AdvertiseData
import android.os.ParcelUuid
import android.bluetooth.le.AdvertiseCallback
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothGattCharacteristic
import android.bluetooth.BluetoothGattService
```

Serwer

- *Pola w klasie aktywności*

```
val TAG = "BLEServer"
var mBluetoothAdapter: BluetoothAdapter? = null
var mBluetoothLeAdvertiser: BluetoothLeAdvertiser? = null
var mBluetoothManager: BluetoothManager? = null
private val SERVICE_UUID=Constants.SERVICE_UUID
var mGattServer: BluetoothGattServer?=null
var mDevices: ArrayList<BluetoothDevice>? = null
```

Serwer

- *Piszemy metodę onCreate, w której tworzymy adapter Bluetooth*

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

    mBluetoothManager = getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
    mBluetoothAdapter = mBluetoothManager?.adapter
    mDevices = ArrayList<BluetoothDevice>()
    Log.e(TAG, msg: "Applikacja uruchomiona")
}
```

Serwer

- *I onResume, gdzie najpierw sprawdzamy dostępność BLE*

```
override fun onResume() {
    super.onResume()
    val ba=mBluetoothAdapter ?: return
    if (!ba.isEnabled) {
        val enableBtIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
        startActivity(enableBtIntent)
        finish()
        return
    }
    if (!packageManager.hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)) {
        finish()
        return
    }

    if (!ba.isMultipleAdvertisementSupported) {
        finish();
        return;
    }
```

Serwer

- *Dalej tworzymy obiekt Bluetooth Advertiser, który będzie odpowiedzialny za rozgłaszanie wokół naszej obecności*

```
mBluetoothLeAdvertiser = ba.getBluetoothLeAdvertiser()
```

- *Oraz tworzymy klasę CallBack dla usługi GattServer*

```
val gattServerCallback = GattServerCallback()
mGattServer = mBluetoothManager?.openGattServer( context: this, gattServerCallback)
```

Serwer

- *Samą klasę CallBack zdefiniujemy wewnątrz klasy aktywności*

```
private inner class GattServerCallback : BluetoothGattServerCallback() {

    override fun onConnectionStateChange(device: BluetoothDevice?, status: Int, newState: Int) {...}

    override fun onCharacteristicWriteRequest(device: BluetoothDevice?, requestId: Int,
                                              characteristic: BluetoothGattCharacteristic?,
                                              preparedWrite: Boolean, responseNeeded: Boolean,
                                              offset: Int, value: ByteArray?) {...}
}
```

- *Na razie ją tak zostawiamy*

Serwer

- *Na koniec metody onResume dodamy teraz wywołanie metody setupServer, którą musimy dopisać*

```
fun setupServer() {
    val service = BluetoothGattService(SERVICE_UUID,
            BluetoothGattService.SERVICE_TYPE_PRIMARY)
    val writeCharacteristic = BluetoothGattCharacteristic(
            Constants.CHARACTERISTIC_ECHO_UUID,
            BluetoothGattCharacteristic.PROPERTY_WRITE,
            BluetoothGattCharacteristic.PERMISSION_WRITE)
    service.addCharacteristic(writeCharacteristic)
    mGattServer?.addService(service)
}
```

Serwer

- *Nasz serwer jest identyfikowane przez pewne UUID oraz oferuje 1 tzw. charakterystykę, też identyfikowaną przez inny UUID, która oferuje usługę typu write*

Serwer

- *Dodamy teraz jeszcze następującą metodę, która uruchamia proces rozgłaszania*

```
private fun startAdvertising() {
    if (mBluetoothLeAdvertiser == null) {
        return
    }
    val settings = AdvertiseSettings.Builder()
            .setAdvertiseMode(AdvertiseSettings.ADVERTISE_MODE_BALANCED)
            .setConnectable(true)
            .setTimeout(0)
            .setTxPowerLevel(AdvertiseSettings.ADVERTISE_TX_POWER_LOW)
            .build()
    val parcelUuid = ParcelUuid(SERVICE_UUID)
    val data = AdvertiseData.Builder()
            .setIncludeDeviceName(true)
            .addServiceUuid(parcelUuid)
            .build()
    mBluetoothLeAdvertiser?.startAdvertising(settings, data, mAdvertiseCallback)
}
```

Serwer

- *Metoda ta wymaga też CallBack'u, który również zdefiniujemy w klasie aktywności*

```
private val mAdvertiseCallback = object : AdvertiseCallback() {
    override fun onStartSuccess(settingsInEffect: AdvertiseSettings) {
        Log.e(TAG,  msg: "Rozgłaszam swoją obecność")
    }

    override fun onStartFailure(errorCode: Int) {
        Log.e(TAG,  msg: "Rozgłaszanie nieudane: $errorCode")
    }
}
```

Serwer

- *Wywołanie startAdvertising dodajemy oczywiście na koniec metody onResume*

- *Musimy jeszcze móc zatrzymać proces rozgłaszania - dodamy metody stopAdvertising i stopServer*

```
private fun stopServer() {
    if (mGattServer != null) {
        mGattServer?.close()
    }
}

private fun stopAdvertising() {
    if (mBluetoothLeAdvertiser != null) {
        mBluetoothLeAdvertiser?.stopAdvertising(mAdvertiseCallback)
    }
}
```

Serwer

- *I przeciążymy metodę onPause aktywności*

```
override fun onPause() {
    super.onPause()
    stopAdvertising()
    stopServer()
}
```

- *Wrócimy teraz do naszego CallBacku GattServer i treści jego metod*

Serwer

```
override fun onConnectionStateChange(device: BluetoothDevice?, status: Int, newState: Int) {
    super.onConnectionStateChange(device, status, newState)
    val dev= device ?: return
    if (newState == BluetoothProfile.STATE_CONNECTED) {
        mDevices?.add(dev);
    } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
        mDevices?.remove(dev);
    }
}
```

Tu serwer zarządza połączonymi urządzeniami

Serwer

```
override fun onCharacteristicWriteRequest(device: BluetoothDevice?, requestId: Int, characteristic: BluetoothGattCharacteristic?,
                                          preparedWrite: Boolean, responseNeeded: Boolean, offset: Int, value: ByteArray?) {
    super.onCharacteristicWriteRequest(device, requestId, characteristic, preparedWrite, responseNeeded, offset, value)
    if (characteristic?.getUuid()?.equals(Constants.CHARACTERISTIC_ECHO_UUID)!!) {
        mGattServer?.sendResponse(device, requestId, BluetoothGatt.GATT_SUCCESS,  offset: 0,  value: null);
        val length :Int?  = value?.size
        val reversed = ByteArray(length!!)
        for (i :Int  in 0 until length) {
            reversed[i] = value[length - (i + 1)]
        }
        characteristic.value = reversed
        for (device :BluetoothDevice  in mDevices!!) {
            mGattServer?.notifyCharacteristicChanged(device, characteristic,  confirm: false)
        }
    }
}
```

Nasz serwer w odpowiedzi na przesłaną informację odsyła ją odwróconą (KOT> TOK)

Klient 2

- *Wracamy do klienta*
- *W metodzie startScan dodamy filtr, który spowoduje, że będziemy szukać tylko naszego serwera*

```
textStatus.text="Uruchamiam skanowanie"

val filters = ArrayList<ScanFilter>()

if (!swAll.isChecked) {
    val scanFilter = ScanFilter.Builder().setServiceUuid(ParcelUuid(Constants.SERVICE_UUID)).build()
    filters.add(scanFilter)
}

val settings = ScanSettings.Builder().setScanMode(ScanSettings.SCAN_MODE_LOW_POWER).build()
```

Klient 2

- *W klasie BtleScanCallback modyfikujemy metodę addScanResult*

```
private fun addScanResult(result: ScanResult) {
    if (!swAll.isChecked) {
        stopScan()
        val bluetoothDevice = result.getDevice()
        val deviceAddress = bluetoothDevice.address
        Log.e(TAG,  msg: "Znalazłem urządzenie $deviceAddress")
        connectDevice(bluetoothDevice)
    }
    else {

        val device = result.device
        val deviceAddress = device.address
        mScanResults?.put(deviceAddress, device)
        Log.e(TAG, msg: "Znalazłem urządzenie $deviceAddress")

    }
}
```

Klient 2

- *Dodajemy metodę connectDevice do aktywności*

```
private fun connectDevice(device: BluetoothDevice) {
    val gattClientCallback = GattClientCallback()
    mGatt = device.connectGatt( context: this, autoConnect: false,gattClientCallback)
}
```

- *I nową klasę wewnętrzną GattClientCallback*

```
private inner class GattClientCallback : BluetoothGattCallback() {...}
```

Klient 2

- *W klasie GattClientCallback przeciążamy metodę onConnectionStateChange*

```
override fun onConnectionStateChange(gatt: BluetoothGatt, status: Int, newState: Int) {
    super.onConnectionStateChange(gatt, status, newState)
    if (status == BluetoothGatt.GATT_FAILURE) {
        disconnectGattServer()
        return
    } else if (status != BluetoothGatt.GATT_SUCCESS) {
        disconnectGattServer()
        return
    }
    if (newState == BluetoothProfile.STATE_CONNECTED) {
        mConnected = true
        gatt.discoverServices()
    } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
        disconnectGattServer()
    }
}
```

Klient 2

- *I podobnie metodę onServicesDiscovered*

```
override fun onServicesDiscovered(gatt: BluetoothGatt?, status: Int) {
    super.onServicesDiscovered(gatt, status)
    if (status != BluetoothGatt.GATT_SUCCESS) {
        return
    }
    val gat=gatt ?: return
    val service = gat.getService(Constants.SERVICE_UUID)
    val characteristic = service?.getCharacteristic(Constants.CHARACTERISTIC_ECHO_UUID)
    characteristic?.writeType = BluetoothGattCharacteristic.WRITE_TYPE_DEFAULT
    mInitialized = gat.setCharacteristicNotification(characteristic, enable: true)
}
```

Klient 2

- *Dodajemy też do aktywności metodę do rozłączania się z serwerem disconnectGattServer*

```
fun disconnectGattServer() {
    mConnected = false
    val gatt=mGatt ?: return
    gatt.disconnect()
    gatt.close()
}
```

- *Możemy teraz wysłać do serwera komunikat*

Klient 2

- *Dodajemy do aktywności metodę sendMessage i upewniamy się, że jest podpięta pod guzik Wyślij*

```
fun sendMessage(v:View) {
    if (!mConnected || !mInitialized) {
        return;
    }
    val service = mGatt?.getService(Constants.SERVICE_UUID);
    val characteristic = service?.getCharacteristic(Constants.CHARACTERISTIC_ECHO_UUID);
    val message = messageEditText.text.toString()
    var messageBytes = ByteArray( size: 0)
    messageBytes = message.toByteArray(Charset.forName( charsetName: "UTF-8"))
    characteristic?.setValue(messageBytes)
    val success = mGatt?.writeCharacteristic(characteristic)
}
```

Klient 2

- *Serwer nam odpowie, więc musimy przechwycić odpowiedź*

- *Przeciążamy metodę onCharacteristicChanged w klasie GattClientCallback*

```
override fun onCharacteristicChanged(gatt: BluetoothGatt?, characteristic: BluetoothGattCharacteristic?) {
    super.onCharacteristicChanged(gatt, characteristic)
    val messageBytes = characteristic?.value
    val message=String(messageBytes!!, Charset.forName( charsetName: "UTF-8"))
    textStatus.text="Otrzymano wiadomość: $message"
}
```

Teraz powinno działać :-)