APPLICAZIONE ANDROID - VENDING MONITOR

Guida completa all'applicazione Android per il controllo remoto del distributore automatico via Bluetooth Low Energy (BLE).

INDICE

PANORAMICA GENERALE

L'applicazione **VendingMonitor** è un'app Android nativa che permette il controllo completo del distributore automatico tramite connessione Bluetooth Low Energy (BLE).

Funzionalità Principali

- ☑ **Connessione Automatica** - Ricerca e connessione al dispositivo "VendingM"
- ☑ **Selezione Prodotti** - 4 prodotti disponibili: Acqua, Snack, Caffè, The
- ☑ **Gestione Credito** - Visualizzazione credito inserito tramite monete
- ☑ **Conferma Acquisto** - Conferma e annullamento con resto automatico
- ☑ **Monitoraggio Sensori** - Temperatura e umidità ambientale in tempo reale
- ☑ **Stato Macchina** - Visualizzazione stato FSM (Riposo, Attesa, Erogazione, Resto, Errore)
- ☑ **Rifornimento Scorte** - Reset scorte prodotti da remoto
- ☑ **Indicatore Scorte** - Visualizzazione quantità rimanente per ogni prodotto

Requisiti di Sistema

Componente	Requisito
Android Version	Android 6.0+ (API 23+)
Bluetooth	Bluetooth Low Energy (BLE) 4.0+
Permessi	BLUETOOTH_SCAN, BLUETOOTH_CONNECT, ACCESS_FINE_LOCATION
Orientamento	Portrait
Tema	Dark Theme Material Design 3

Screenshot e Demo

📷 **Galleria Foto e Schermate App**: Visualizza su Google Photos

La galleria contiene:

- Screenshot interfaccia utente completa
- Dettagli componenti UI (pulsanti prodotto, sensori, stati)
- Schermate connessione BLE
- Demo funzionalità (selezione prodotti, conferma, resto)
- Indicatori scorte e rifornimento

🧬 TECNOLOGIE E ARCHITETTURA

Stack Tecnologico

```
┌─────────────────────────────────────────┐
│         JETPACK COMPOSE UI              │  ← Interfaccia Utente Dichiarativa
├─────────────────────────────────────────┤
│       MATERIAL DESIGN 3 THEME           │  ← Stile e Componenti
├─────────────────────────────────────────┤
│      BLUETOOTH GATT CLIENT API          │  ← Comunicazione BLE
├─────────────────────────────────────────┤
│    ANDROID RUNTIME PERMISSIONS          │  ← Gestione Permessi
└─────────────────────────────────────────┘
```

Architettura Componenti

1. UI Layer (Jetpack Compose)

- **VendingDashboard**: Componente root principale
- **ProductButton**: Pulsanti selezione prodotto
- **SensorMiniCard**: Card sensori temperatura/umidità
- **MachineStateBar**: Barra stato macchina FSM
- **StatusIndicator**: Indicatore connessione BLE
- **ActionButton**: Pulsante connessione/disconnessione

2. Business Logic Layer

- **MainActivity**: Activity principale con gestione stato
- **Composable State Variables**: Stati reattivi UI
  - `tempState`, `humState`: Sensori ambientali
  - `creditState`, `machineState`: Stato macchina
  - `scorteAcqua`, `scorteSnack`, `scorteCaffe`, `scorteThe`: Scorte prodotti
  - `connectionStatus`, `isScanning`: Stato connessione BLE

3. Communication Layer (BLE)

- **BluetoothGatt**: Client GATT per comunicazione BLE

- **ScanCallback**: Callback scansione dispositivi
- **BluetoothGattCallback**: Gestione eventi connessione e notifiche
- **Handler**: Thread UI per aggiornamenti asincroni

Pattern Architetturali

Reactive UI Pattern

```
// Variabili di stato osservabili da Compose
private var creditState by mutableIntStateOf(0)

// Quando lo stato cambia, UI si aggiorna automaticamente
creditState = newCredit  // ← UI recompone automaticamente
```

BLE Event-Driven Pattern

```
1. startScan() → Scanner cerca dispositivo "VendingM"
2. onScanResult() → Dispositivo trovato
3. connectToDevice() → Connessione GATT
4. onConnectionStateChange() → Connesso
5. discoverServices() → Enumera servizi
6. onServicesDiscovered() → Servizi trovati
7. enableNotification() × 3 → Abilita TEMP, HUM, STATUS
8. onCharacteristicChanged() → Riceve notifiche periodiche
```

PROTOCOLLO BLE

UUID Servizi e Caratteristiche

```
Servizio Principale: 0000A000-0000-1000-8000-00805f9b34fb

├─ 0xA001: CHAR_TEMP       (Temperatura)      [READ, NOTIFY]
├─ 0xA002: CHAR_STATUS     (Credito + Stato)  [READ, NOTIFY]
├─ 0xA003: CHAR_HUM        (Umidità)          [READ, NOTIFY]
└─ 0xA004: CMD_CHAR        (Comandi)          [WRITE_NO_RESPONSE]

Descrittore Standard:
└─ 0x2902: CCCD            (Enable Notifications)
```

Caratteristica TEMPERATURA (0xA001)

Tipo: NOTIFY (periodica ogni 2 secondi)
Formato: `int32` little-endian (4 byte)
Range: 0-50°C (sensore DHT11)

```
// Parsing esempio
val temp = ByteBuffer.wrap(data).order(ByteOrder.LITTLE_ENDIAN).int
// temp = 23 (°C)
```

Caratteristica UMIDITÀ (0xA003)

Tipo: NOTIFY (periodica ogni 2 secondi)
Formato: `int32` little-endian (4 byte)
Range: 20-90% (sensore DHT11)

```
// Parsing esempio
val hum = ByteBuffer.wrap(data).order(ByteOrder.LITTLE_ENDIAN).int
// hum = 65 (%)
```

Caratteristica STATUS (0xA002)

Tipo: NOTIFY (su ogni cambio stato)
Formato: 6 byte array
Struttura:

Byte	Nome	Tipo	Range	Descrizione
0	`credito`	uint8	0-255	Credito inserito in EUR
1	`stato`	uint8	0-4	Stato FSM corrente
2	`scorte_acqua`	uint8	0-5	Pezzi rimanenti acqua
3	`scorte_snack`	uint8	0-5	Pezzi rimanenti snack
4	`scorte_caffe`	uint8	0-5	Pezzi rimanenti caffè
5	`scorte_the`	uint8	0-5	Pezzi rimanenti the

Stati FSM:

- `0` = RIPOSO (verde) - Sistema pronto
- `1` = ATTESA_MONETA (blu) - Utente presente
- `2` = EROGAZIONE (giallo) - Dispensa prodotto
- `3` = RESTO (magenta) - Restituzione credito
- `4` = ERRORE (rosso) - Temperatura critica >40°C

```
// Parsing esempio (CRITICAL: and 0xFF converte byte signed → unsigned)
creditState = data[0].toInt() and 0xFF   // 3 EUR
machineState = data[1].toInt() and 0xFF  // 1 = ATTESA_MONETA
scorteAcqua = data[2].toInt() and 0xFF   // 4 pezzi
scorteSnack = data[3].toInt() and 0xFF   // 2 pezzi
```

```
scorteCaffe = data[4].toInt() and 0xFF   // 5 pezzi
scorteThe = data[5].toInt() and 0xFF     // 3 pezzi
```

Caratteristica COMANDI (0xA004)

Tipo: WRITE_NO_RESPONSE (comando istantaneo)
Formato: 1 byte
Direzione: App → Firmware

Comandi Disponibili:

Codice	Nome	Descrizione
1	SEL_ACQUA	Seleziona prodotto ACQUA (1,00 €)
2	SEL_SNACK	Seleziona prodotto SNACK (2,00 €)
3	SEL_CAFFE	Seleziona prodotto CAFFÈ (1,00 €)
4	SEL_THE	Seleziona prodotto THE (2,00 €)
9	ANNULLA_RESTO	Annulla acquisto e restituisce credito
10	CONFERMA	Conferma acquisto ed eroga prodotto
11	RIFORNIMENTO	Reset tutte le scorte a 5 pezzi

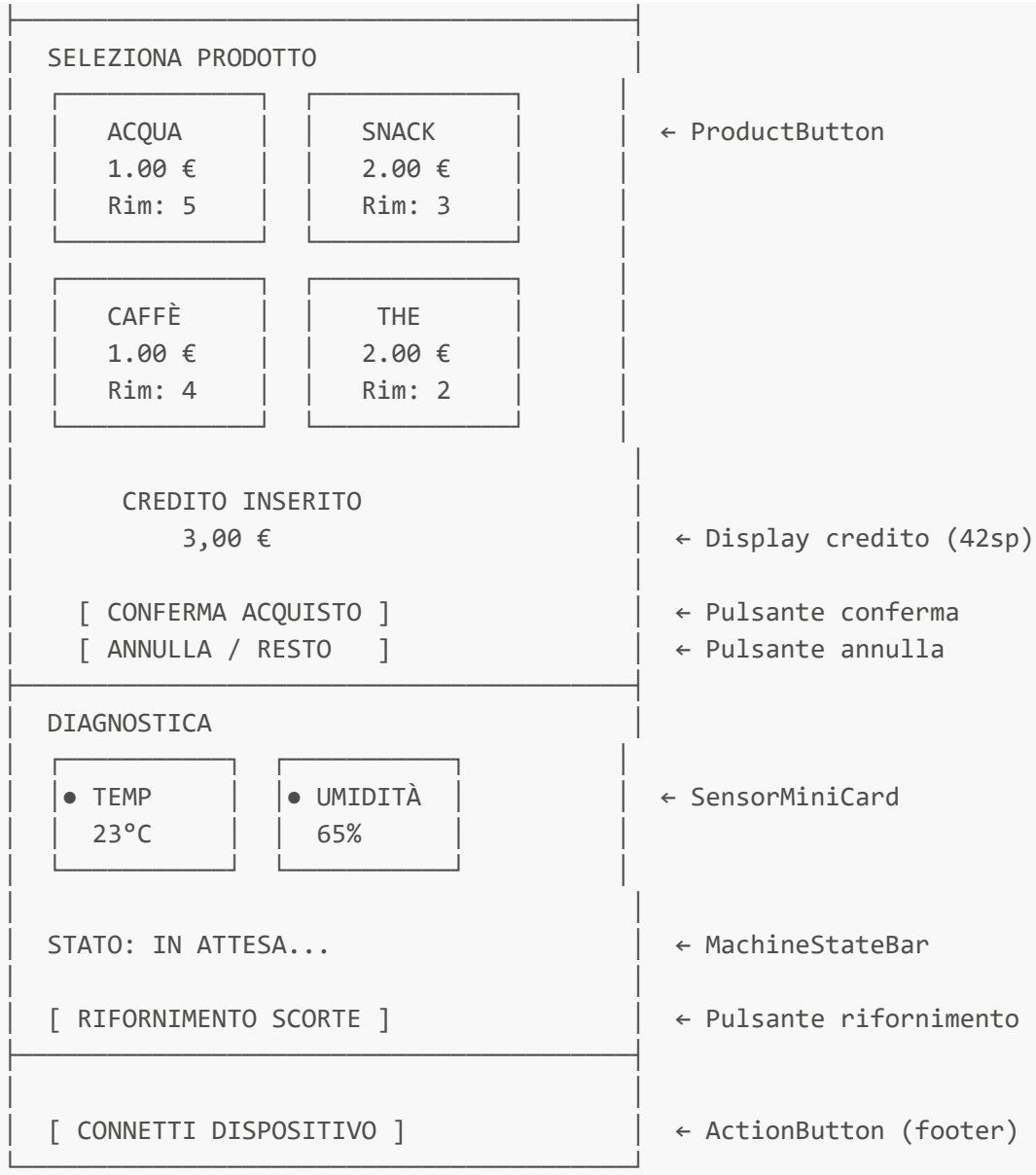
```
// Invio comando esempio
fun writeCommand(cmd: Int) {
    val payload = byteArrayOf(cmd.toByte())
    bluetoothGatt?.writeCharacteristic(
        charCmd,
        payload,
        BluetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE
    )
}

// Esempi d'uso
writeCommand(1)   // Seleziona ACQUA
writeCommand(10)  // Conferma acquisto
writeCommand(9)   // Annulla/Resto
```

🎨 INTERFACCIA UTENTE

Layout Generale

```
┌──────────────────────────────────────────┐
│        VENDING MONITOR                   │  ← Header (22sp)
│        ● Connesso                        │  ← StatusIndicator
```



Color Palette (Material Design 3 Dark)

```
// Colori principali
Background:     #121212  // Sfondo principale scuro
Surface:        #1E1E1E  // Sfondo card
SurfaceDim:     #2C2C2C  // Sfondo sensori

Primary:        #BB86FC  // Viola (header, pulsante connetti)
Secondary:      #03DAC6  // Teal (accenti)

// Colori prodotti
Acqua:          #00E5FF  // Ciano brillante
Snack:          #FF4081  // Magenta
Caffè:          #FFEB3B  // Giallo
The:            #69F0AE  // Verde chiaro

// Colori stati
Success:        #4CAF50  // Verde (conferma, scorte OK)
```

```
Error:          #CF6679  // Rosso (annulla, errore)
Warning:        #FF9800  // Arancione (temperatura)
Info:           #2196F3  // Blu (rifornimento)

// Colori stati macchina
RIPOSO:         #81C784  // Verde chiaro
ATTESA:         #64B5F6  // Blu chiaro
EROGAZIONE:     #FFD54F  // Giallo chiaro
RESTO:          #BA68C8  // Magenta chiaro
ERRORE:         #E57373  // Rosso chiaro
```

Font Sizes (v2.0 - Aumentati per Leggibilità)

Componente	Font Size	Peso	Uso
Header	22sp	Black	Titolo principale
Display Credito	42sp	Bold	Credito EUR
Nome Prodotto	18sp	Bold	Testo pulsanti prodotto
Prezzo Prodotto	15sp	Normal	Prezzo prodotti
Pulsanti Azione	13sp	Bold	Testo pulsanti principali
Stato Macchina	15sp	Bold	Testo stato FSM
Sensori Valore	18sp	Bold	Temperatura/Umidità
Sensori Label	11sp	Normal	Label "TEMP", "UMIDITÀ"
Status Indicator	14sp	Normal	Testo connessione

Stati Visivi Componenti

ProductButton Stati

Stato	Sfondo	Bordo	Testo	Clickable
Disponibile	Trasparente	1dp Gray	White	✓
Selezionato	Color 20% alpha	2dp Color	Color	✓
Esaurito	DarkGray 30%	1dp DarkGray	DarkGray	✗

StatusIndicator Colori

Stato	Colore Pallino	Testo
Connesso	Verde (#00E676)	"Connesso"
Scansione...	Giallo (#FFEB3B)	"Scansione..."
Disconnesso	Rosso (#CF6679)	"Disconnesso"

Stato	Colore Pallino	Testo
Bluetooth Spento	🔴 Rosso	"Bluetooth Spento!"
Errore	🔴 Rosso	"Errore Scan: [code]"

ActionButton Dinamico

Stato Connessione	Testo Pulsante	Colore	Azione
Disconnesso	"CONNETTI DISPOSITIVO"	Viola (#BB86FC)	checkPermissionsAndScan()
Scansione...	"CONNETTI DISPOSITIVO"	Viola	stopScan()
Connesso	"DISCONNETTI"	Rosso (#CF6679)	disconnectDevice()

⚙️ CONFIGURAZIONE E BUILD

Prerequisiti

1. **Android Studio** (versione Flamingo 2022.2.1 o successiva)
2. **Gradle** 8.0+
3. **Kotlin** 1.9.0+
4. **Jetpack Compose BOM** 2024.01.00+

Permessi Richiesti (AndroidManifest.xml)

```
<!-- Permessi Bluetooth (Android 12+) -->
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"
    android:usesPermissionFlags="neverForLocation" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />

<!-- Permessi Bluetooth Legacy (Android < 12) -->
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

<!-- Permessi Location (necessari per BLE scan) -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

<!-- Feature BLE richiesta -->
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"
/>
```

Build Configuration

```
// build.gradle (app level)
android {
    compileSdk = 34
```

```
    defaultConfig {
        applicationId = "com.example.vendingmonitor"
        minSdk = 23        // Android 6.0+
        targetSdk = 34     // Android 14
        versionCode = 2
        versionName = "2.0"
    }

    buildFeatures {
        compose = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = "1.5.3"
    }
}

dependencies {
    // Jetpack Compose
    implementation(platform("androidx.compose:compose-bom:2024.01.00"))
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.material3:material3")
    implementation("androidx.activity:activity-compose:1.8.2")

    // Core Android
    implementation("androidx.core:core-ktx:1.12.0")
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")
}
```

Build e Installazione

```
# 1. Clone repository
git clone https://github.com/santoromarco74/VendingMachine.git
cd VendingMachine

# 2. Apri progetto in Android Studio
# File → Open → seleziona cartella 'app'

# 3. Build progetto
./gradlew assembleDebug

# 4. Installa su dispositivo via USB
./gradlew installDebug

# 5. Oppure genera APK per distribuzione
./gradlew assembleRelease
# APK generato in: app/build/outputs/apk/release/app-release.apk
```

Configurazione Firma APK (Release)

```
// build.gradle (app level)
android {
    signingConfigs {
        release {
            storeFile file("keystore.jks")
            storePassword "your_password"
            keyAlias "vending_key"
            keyPassword "your_password"
        }
    }

    buildTypes {
        release {
            signingConfig signingConfigs.release
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
}
```

⚡ FUNZIONALITÀ DETTAGLIATE

Sequenza Connessione BLE

```
1. Utente preme "CONNETTI DISPOSITIVO"
   ↓
2. checkPermissionsAndScan()
   ├─ Verifica permessi BLUETOOTH_SCAN, BLUETOOTH_CONNECT, LOCATION
   ├─ Se mancanti → richiede permessi
   └─ Se concessi → startScan()
   ↓
3. startScan()
   ├─ Avvia BluetoothLeScanner
   ├─ Cerca dispositivi con nome contenente "Vending" (case-insensitive)
   ├─ Timeout 10 secondi
   └─ onScanResult() → dispositivo trovato
   ↓
4. connectToDevice(device)
   ├─ bluetoothGatt = device.connectGatt(...)
   └─ onConnectionStateChange() → STATE_CONNECTED
   ↓
5. discoverServices()
   └─ onServicesDiscovered() → servizio 0xA000 trovato
   ↓
6. enableNotification() × 3
   ├─ Abilita CHAR_TEMP (0xA001)
   ├─ Abilita CHAR_HUM (0xA003)
   └─ Abilita CHAR_STATUS (0xA002)
```

```
   ↓
7. Connessione completa!
   ├─ StatusIndicator: 🟢 "Connesso"
   ├─ Notifiche periodiche attive
   └─ writeCommand(1) → Seleziona ACQUA come default
```

Gestione Permessi Runtime (Android 6+)

```
// Permessi dinamici richiesti alla prima connessione
val permissionsLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestMultiplePermissions()
) { perms ->
    if (perms.values.all { it }) {
        // Tutti i permessi concessi → avvia scan
        startScan()
    } else {
        // Permessi negati → mostra messaggio
        Toast.makeText(this, "Permessi necessari!", Toast.LENGTH_SHORT).show()
    }
}

// Verifica permessi prima di scansione
fun checkPermissionsAndScan(launcher: ActivityResultLauncher<Array<String>>) {
    val permsNeeded = mutableListOf<String>()

    // Android 12+: BLUETOOTH_SCAN, BLUETOOTH_CONNECT
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        if (checkSelfPermission(BLUETOOTH_SCAN) != GRANTED)
            permsNeeded.add(BLUETOOTH_SCAN)
        if (checkSelfPermission(BLUETOOTH_CONNECT) != GRANTED)
            permsNeeded.add(BLUETOOTH_CONNECT)
    }

    // Tutti: ACCESS_FINE_LOCATION
    if (checkSelfPermission(ACCESS_FINE_LOCATION) != GRANTED)
        permsNeeded.add(ACCESS_FINE_LOCATION)

    if (permsNeeded.isEmpty()) startScan()
    else launcher.launch(permsNeeded.toTypedArray())
}
```

Gestione Notifiche BLE

```
// NOTA CRITICA: Android 13+ e precedenti usano API diverse!

// Android 13+ (API 33+)
override fun onCharacteristicChanged(
    gatt: BluetoothGatt,
    characteristic: BluetoothGattCharacteristic,
```

```
    value: ByteArray  // ← Nuovo parametro value
) {
    processData(characteristic.uuid, value)
}

// Android < 13 (API < 33)
@Deprecated("Deprecated in Java")
override fun onCharacteristicChanged(
    gatt: BluetoothGatt,
    characteristic: BluetoothGattCharacteristic
) {
    @Suppress("DEPRECATION")
    val value = characteristic.value  // ← value è proprietà
    processData(characteristic.uuid, value)
}
```

Parsing Dati BLE

CRITICAL FIX: Conversione Byte Signed → Unsigned

```
// ❌ ERRORE COMUNE: byte signed interpretato come negativo
val credito = data[0].toInt()
// Se data[0] = 0xFF (255 unsigned), diventa -1 signed!

// ☑ CORRETTO: and 0xFF maschera bit di segno
val credito = data[0].toInt() and 0xFF
// 0xFF (byte signed -1) → 255 (int unsigned) ✓
```

Parsing Int32 Little-Endian

```
// Temperatura/Umidità: 4 byte int32 little-endian
val temp = ByteBuffer.wrap(data)
    .order(ByteOrder.LITTLE_ENDIAN)  // ← STM32 usa little-endian
    .int

// Esempio: [0x17, 0x00, 0x00, 0x00] → 23°C
```

Gestione Scorte Prodotti

```
// Indicatore visivo scorte in ProductButton
Text(
    text = if (stock == 0) "ESAURITO" else "Rim: $stock",
    fontSize = 11.sp,
    color = if (stock == 0) Color.Red else Color(0xFF4CAF50),
    fontWeight = FontWeight.Bold
)
```

```
// Disabilitazione click se esaurito
Card(
    modifier = Modifier.clickable(enabled = stock > 0) { onClick() },
    // ...
)

// Aggiornamento da notifica STATUS
scorteAcqua = data[2].toInt() and 0xFF  // Byte 2
scorteSnack = data[3].toInt() and 0xFF  // Byte 3
scorteCaffe = data[4].toInt() and 0xFF  // Byte 4
scorteThe = data[5].toInt() and 0xFF    // Byte 5
```

Rifornimento Scorte Remoto

```
// Pulsante rifornimento invia comando BLE 11
Button(
    onClick = {
        Toast.makeText(this, "Rifornimento in corso...",
Toast.LENGTH_SHORT).show()
        writeCommand(11)  // ← Reset scorte firmware a 5 pezzi
    }
) {
    Text("RIFORNIMENTO SCORTE")
}

// Firmware riceve comando e resetta:
// scorteAcqua = 5
// scorteSnack = 5
// scorteCaffe = 5
// scorteThe = 5
// → Invia notifica STATUS aggiornata → App aggiorna UI
```

🔧 TROUBLESHOOTING

Problemi Comuni e Soluzioni

1. App non trova dispositivo BLE

Sintomi: StatusIndicator mostra "Nessun Disp. Trovato" dopo 10 secondi

Cause possibili:

- ❌ Bluetooth smartphone disattivato
- ❌ Permessi location non concessi
- ❌ Firmware non in esecuzione su Nucleo
- ❌ Distanza troppo elevata (>10 metri)

Soluzioni:

```
1. Verifica Bluetooth attivo: Impostazioni → Bluetooth → ON
2. Concedi permessi location: Impostazioni → App → VendingMonitor → Permessi
3. Verifica firmware: LED LD2 Nucleo deve lampeggiare (advertising BLE attivo)
4. Avvicina smartphone a Nucleo (<5 metri per connessione stabile)
5. Riavvia scan: Premi "DISCONNETTI" e poi "CONNETTI DISPOSITIVO"
```

2. Credito mostra valori negativi

Sintomo: Display mostra "-1,00 €" invece di "255,00 €"

Causa: Manca conversione `and 0xFF` per byte signed → unsigned

Soluzione:

```
// Verifica in processData() linea 1196:
creditState = data[0].toInt() and 0xFF  // ← DEVE avere "and 0xFF"
```

3. Temperature/Umidità sempre a 0

Sintomi: SensorMiniCard mostra "0°C" e "0%" anche se sensore funziona

Cause possibili:

- ❌ Notifiche CHAR_TEMP/CHAR_HUM non abilitate
- ❌ Sensore DHT11 disconnesso o guasto
- ❌ Firmware non invia dati sensori

Debugging:

```
// 1. Verifica log Android (Logcat)
android.util.Log.d("VendingMonitor", "Temp aggiornata: $temp°C")

// 2. Controlla callback onServicesDiscovered()
// Deve chiamare enableNotification(gatt, charTemp)

// 3. Verifica firmware invia notifiche ogni 2 secondi
// printf("[BLE] Temp: %d°C, Hum: %d%%\n", temperatura, umidita);
```

4. Pulsante "CONFERMA ACQUISTO" sempre disabilitato

Sintomo: Pulsante grigio anche con credito inserito

Causa: Condizione `enabled = creditState >= 1` non soddisfatta

Debugging:

```
// Verifica valore creditState
android.util.Log.d("VendingMonitor", "creditState = $creditState")

// Se creditState = 0 ma hai inserito monete:
// → Verifica notifica STATUS arrivi correttamente
// → Controlla firmware invii data[0] = credito
```

5. Disconnessione improvvisa BLE

Sintomi: StatusIndicator passa da "Connesso" a "Disconnesso" senza azione utente

Cause possibili:

- ❌ Distanza eccessiva (segnale debole)
- ❌ Interferenze radio (WiFi 2.4GHz, altri BLE)
- ❌ Firmware crashato o watchdog reset
- ❌ Timeout connessione (>30s senza comunicazione)

Soluzioni:

```
1. Mantieni distanza <5 metri
2. Evita ostacoli metallici tra smartphone e Nucleo
3. Verifica firmware non vada in HardFault (LED LD2 lampeggia rapidamente rosso)
4. Controlla alimentazione stabile (vedi WIRING.md)
5. Riconnetti manualmente premendo "CONNETTI DISPOSITIVO"
```

6. Scorte prodotto non si aggiornano dopo rifornimento

Sintomo: Premendo "RIFORNIMENTO SCORTE", scorte rimangono a 0

Debugging:

```
// 1. Verifica comando inviato
android.util.Log.d("VendingMonitor", "writeCommand chiamato con cmd=11")

// 2. Controlla firmware riceva comando
// printf("[BLE] Comando ricevuto: %d\n", cmd);

// 3. Verifica firmware invii notifica STATUS aggiornata dopo rifornimento
// vendingServicePtr->updateStatus(credito, statoCorrente);

// 4. Log notifica STATUS ricevuta
android.util.Log.d("VendingMonitor", "STATUS ricevuto: scorte=
[$scorteAcqua,$scorteSnack,...]")
```

7. App crasha all'avvio su Android < 12

Sintomo: App si chiude immediatamente su dispositivi Android 11 o precedenti

Causa: Codice usa API Android 12+ senza controllo `Build.VERSION.SDK_INT`

Soluzione:

```
// Usa API condizionale per writeCharacteristic/writeDescriptor
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
    // Android 13+ API
    gatt.writeCharacteristic(char, payload, WRITE_TYPE_NO_RESPONSE)
} else {
    // Android 12- API (deprecata ma funzionante)
    @Suppress("DEPRECATION")
    char.value = payload
    @Suppress("DEPRECATION")
    gatt.writeCharacteristic(char)
}
```

📝 CHANGELOG VERSIONI

v2.0 (2025-01-06) - UI Migliorata + Documentazione Italiana

Modifiche UI:

- ☑ Font aumentati per leggibilità:
  - Header: 18sp → 22sp
  - Display credito: 36sp → 42sp
  - Nome prodotto: 16sp → 18sp
  - Prezzo prodotto: 13sp → 15sp
  - Pulsanti azione: 11sp → 13sp
  - Stato macchina: 13sp → 15sp
  - Sensori: 16sp → 18sp

Documentazione:

- ☑ Aggiunta documentazione italiana completa (500+ righe commenti)
- ☑ KDoc per ogni funzione e variabile
- ☑ Commenti inline per logica complessa
- ☑ Spiegazione architettura e pattern

File modificato: `app/src/main/java/com/example/vendingmonitor/MainActivity.kt`

---

v1.0 (2025-01-05) - Release Iniziale

Funzionalità implementate:

- ☑ Connessione BLE automatica dispositivo "VendingM"
- ☑ Selezione 4 prodotti (Acqua, Snack, Caffè, The)
- ☑ Display credito inserito
- ☑ Conferma acquisto e annullamento
- ☑ Monitoraggio sensori temperatura/umidità
- ☑ Visualizzazione stato macchina FSM
- ☑ Gestione scorte prodotti
- ☑ Rifornimento scorte remoto
- ☑ UI Jetpack Compose Material Design 3 dark theme

Tecnologie:

- Android 6.0+ (API 23+)
- Kotlin 1.9.0
- Jetpack Compose BOM 2024.01.00
- Bluetooth Low Energy GATT

📚 RIFERIMENTI

Documentazione Correlata

- **Firmware**: `/firmware/main.cpp` - Codice STM32 con protocollo BLE
- **Hardware**: `/WIRING.md` - Schema collegamenti e alimentazione
- **Bug Fix**: `/BUGFIXES.md` - Storico fix applicati
- **Architettura**: `/ARCHITECTURE.md` - Overview sistema completo

Link Utili

Documentazione Ufficiale:

- Jetpack Compose Documentation
- Material Design 3
- Android BLE Guide
- Bluetooth GATT Specifications

Demo e Media:

- 📸 Screenshot e Schermate App - Galleria Google Photos

Contatti Sviluppo

- **Repository**: https://github.com/santoromarco74/VendingMachine
- **Issues**: https://github.com/santoromarco74/VendingMachine/issues

---

Ultimo aggiornamento: 2026-01-06
Versione documento: 1.0
Versione app: v2.0