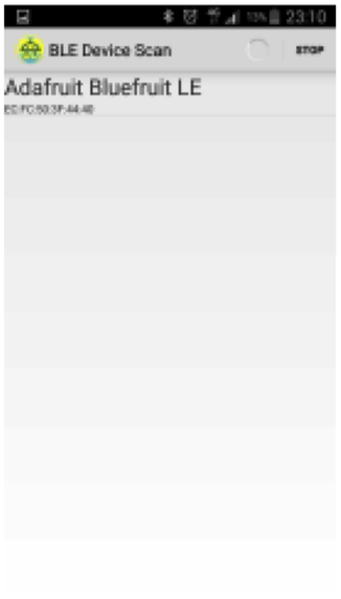
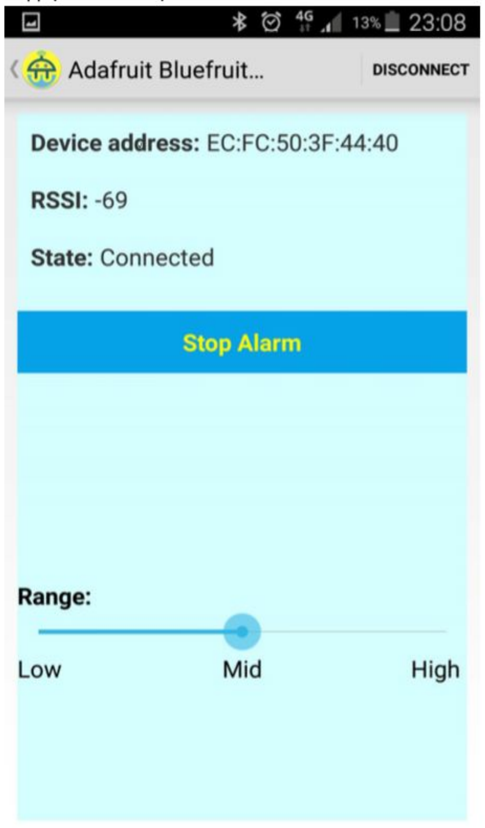
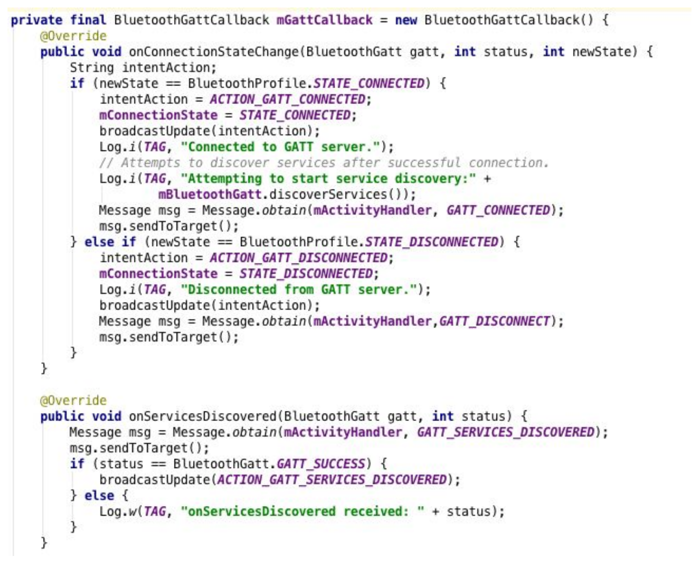
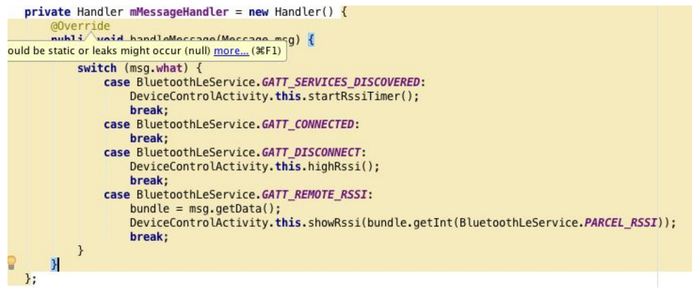
The main parts of the code for the Android App consist of two Activities and one Service. Activities provide the user interface where users interact with the app directly, while the Service allows components to run in the background to perform long-running operations.

 Adafruit Bluefruit LE EC:FC:50:3F:44:40	With the first Activity, let's call it the ScanActivity, we can scan, stop, and refresh the scanning process for BLE-enabled devices, and then select and connect to the BLE-enabled Sticker* that is attached to our item. *Sticker used for this prototype is nrf51822-based Adafruit Flora
 23:08 Adafruit Bluefruit... DISCONNECT Stop Alarm Low Mid High **Device address:** EC:FC:50:3F:44:40 **RSSI:** -69 **State:** Connected **Range:** Mid	Another Activity, let's call it the ControlActivity, displays device information and real-time updates of RSSI values. We can also use the slider here to adjust the threshold distance--how far the item is separated from our phone before both the phone and the Sticker ring.



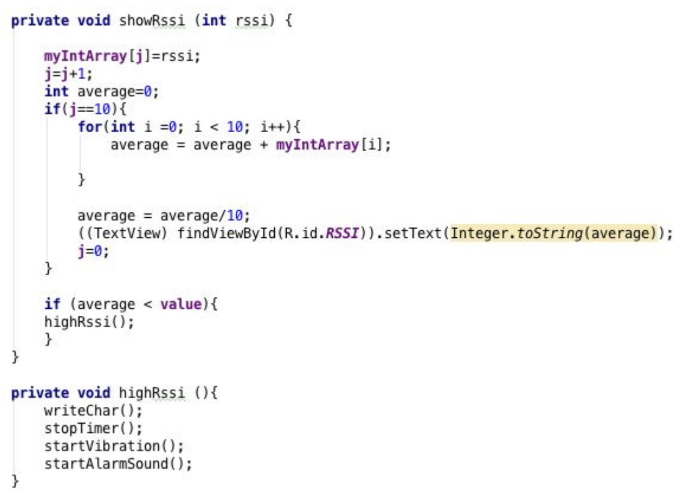
```java
private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
        String intentAction;
        if (newState == BluetoothProfile.STATE_CONNECTED) {
            intentAction = ACTION_GATT_CONNECTED;
            mConnectionState = STATE_CONNECTED;
            broadcastUpdate(intentAction);
            Log.i(TAG, "Connected to GATT server.");
            // Attempts to discover services after successful connection.
            Log.i(TAG, "Attempting to start service discovery:" +
                    mBluetoothGatt.discoverServices());
            Message msg = Message.obtain(mActivityHandler, GATT_CONNECTED);
            msg.sendToTarget();
        } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
            intentAction = ACTION_GATT_DISCONNECTED;
            mConnectionState = STATE_DISCONNECTED;
            Log.i(TAG, "Disconnected from GATT server.");
            broadcastUpdate(intentAction);
            Message msg = Message.obtain(mActivityHandler,GATT_DISCONNECT);
            msg.sendToTarget();
        }
    }

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        Message msg = Message.obtain(mActivityHandler, GATT_SERVICES_DISCOVERED);
        msg.sendToTarget();
        if (status == BluetoothGatt.GATT_SUCCESS) {
            broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
        } else {
            Log.w(TAG, "onServicesDiscovered received: " + status);
        }
    }
```

The Service, BluetoothLeService, implements callback methods for GATT events, including checking whether or not it is connected to the Sticker, and which services are discovered.



```java
  private Handler mMessageHandler = new Handler() {
      @Override
      public void handleMessage(Message msg) {
          switch (msg.what) {
              case BluetoothLeService.GATT_SERVICES_DISCOVERED:
                  DeviceControlActivity.this.startRssiTimer();
                  break;
              case BluetoothLeService.GATT_CONNECTED:
                  break;
              case BluetoothLeService.GATT_DISCONNECT:
                  DeviceControlActivity.this.highRssi();
                  break;
              case BluetoothLeService.GATT_REMOTE_RSSI:
                  bundle = msg.getData();
                  DeviceControlActivity.this.showRssi(bundle.getInt(BluetoothLeService.PARCEL_RSSI));
                  break;
          }
      }
  };
```

Once connected to the Sticker, a timer will be called to read RSSI values via the BLE Service every 5 milliseconds. When RSSI values are obtained, a function in BLE Service will call the message handler in ControlActivity to show and update the RSSI field as the value changes.



```java
private void showRssi (int rssi) {

    myIntArray[j]=rssi;
    j=j+1;
    int average=0;
    if(j==10){
        for(int i =0; i < 10; i++){
            average = average + myIntArray[i];

        }

        average = average/10;
        ((TextView) findViewById(R.id.RSSI)).setText(Integer.toString(average));
        j=0;
    }

    if (average < value){
    highRssi();
    }
}

private void highRssi (){
    writeChar();
    stopTimer();
    startVibration();
    startAlarmSound();
}
```

To avoid false alarm triggered by outlier RSSI values, we implemented the **moving average filter**.
When the threshold is breached, we call several other functions that are designed to allow the phone to vibrate and ring indefinitely. Both the Sticker and the Phone will ring until the user presses the Stop Alarm button on the app.

As a security measure, the Sticker will read RSSI values itself independent of the operation of the phone app and the buzzer on the Sticker will ring whenever the threshold is breached

//Our pickpocket prevention tool also ensures that, whether the phone or the item is stolen, both items will ring and alert the user.