Mobile Application Development for IoT Unit-3 Material

Syllabus

Bluetooth Weather Station: Hardware and Software requirements, Writing the Arduino sketch, Enhancing the user interface
Pulse Rate Sensor: Hardware and Software requirements, Writing the Arduino sketch

Project-1: Bluetooth Weather Station

Hardware Requirements

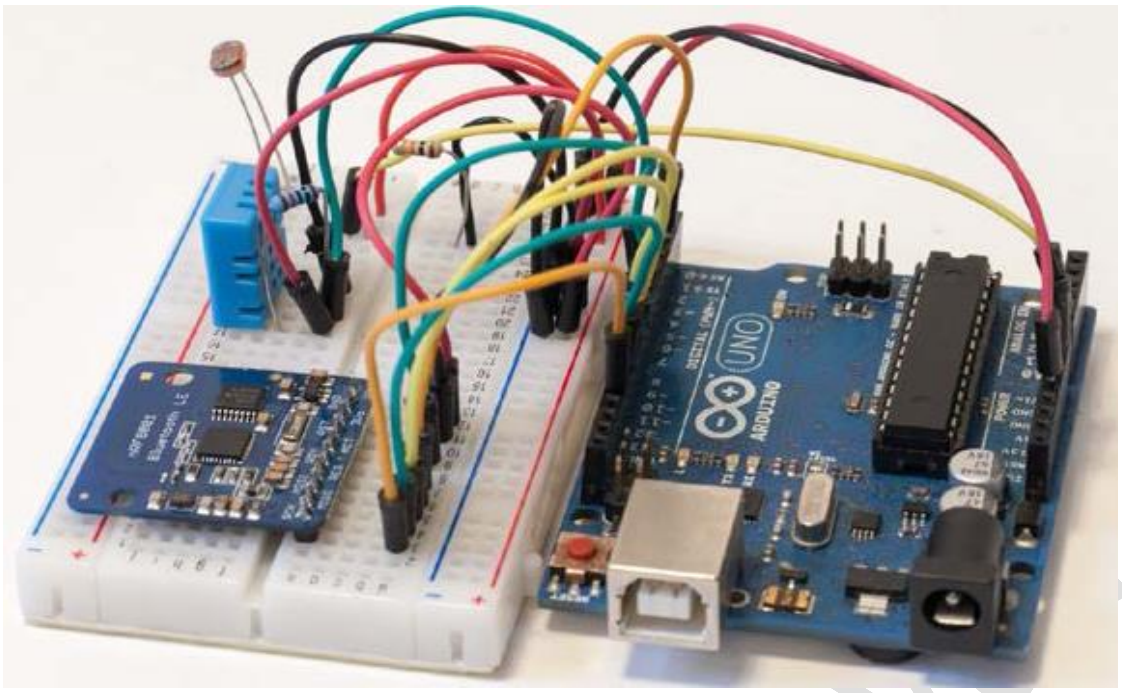
- The Arduino Uno board
- The DHT11 sensor and 4.7K Ohm resistor
- The photocell
- The 10K Ohm resistor
- Adafruit nRF8001 breakout board
- The breadboard
- Jumper wires

Software Requirements

- Arduino IDE
- aREST library
- DHT11 library
- nRF8001 library

Hardware Configuration

1. Place the Bluetooth module, the DHT11 sensor, and the photocell on the breadboard
2. Connect the power supply from the Arduino board to the breadboard: 5V goes to red power rail, and GND goes to the blue power rail
3. Connecting the BLE module: First, connect the power supply of the module: GND goes to the blue power rail, and VIN goes to the red power rail. Connect the different wires responsible for the SPI: SCK to Arduino pin 13, MISO to Arduino pin 12, and MOSI to Arduino pin 11. Connect the REQ pin to Arduino pin 10. Finally, connect the RDY pin to Arduino pin 2, and RST pin to Arduino pin 9.
4. For DHT11 sensor, first connect the power supply: the VCC pin goes to the red power rail on the breadboard, and the GND pin goes to the blue power rail, DATA pin to pin number 7 of the Arduino board.
5. Place the 4.7K Ohm resistor between the VCC and the DATA pin for the sensor
6. For the photocell, connect the 10K Ohm resistor in series with the photocell. This means that one pin of the photocell should be in contact (on the same row of the breadboard) with one pin of the resistor. Connect the other pin of the resistor to the blue power rail, and the other pin of the photocell to the red power rail of the breadboard
7. Finally, connect the common pin between the photocell and resistor to the analog pin A0 of the Arduino board



Testing the Sensors

```
#include "DHT.h"
// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11
// DHT instance
DHT dht(DHTPIN, DHTTYPE);
void setup()
{
// Initialize the Serial port
Serial.begin(9600);
// Init DHT
dht.begin();
}
void loop()
{
// Measure from DHT
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();
// Measure light level
float sensor_reading = analogRead(A0);
float light = sensor_reading/1024*100;
// Display temperature
Serial.print("Temperature: ");
Serial.print((int)temperature);
Serial.println(" C");
// Display humidity
Serial.print("Humidity: ");
Serial.print(humidity);
Serial.println("%");
// Display light level
Serial.print("Light: ");
Serial.print(light);
Serial.println("%");
Serial.println("");
// Wait 500 ms
delay(500);
}
```

Complete Arduino Sketch for Bluetooth Weather Station

```
// Control Arduino board from BLE
// Enable lightweight
#define LIGHTWEIGHT 1
// Libraries
#include <SPI.h>
#include "Adafruit_BLE_UART.h"
#include <aREST.h>
#include "DHT.h"
// Pins
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2
#define ADAFRUITBLE_RST 9
// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11
// DHT instance
DHT dht(DHTPIN, DHTTYPE);
// Create aREST instance
aREST rest = aREST();
// BLE instance
Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ,
ADAFRUITBLE_RDY, ADAFRUITBLE_RST);
// Variables to be exposed to the API
int temperature;
int humidity;
int light;
void setup(void)
{
// Start Serial
Serial.begin(9600);
// Start BLE
BTLEserial.begin();
// Give name and ID to device
rest.set_id("001");
rest.set_name("weather_station");
// Expose variables to API
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
rest.variable("light",&light);
// Init DHT
dht.begin();
// Welcome message
Serial.println("Weather station started");
}
void loop() {
// Measure from DHT
float t = dht.readTemperature();
float h = dht.readHumidity();
temperature = (int)t;
humidity = (int)h;
// Measure light level
float sensor_reading = analogRead(A0);
light = (int)(sensor_reading/1024*100);
// Tell the nRF8001 to do whatever it should be working on.
BTLEserial.pollACI();
// Ask what is our current status
aci_evt_opcode_t status = BTLEserial.getState();
// Handle REST calls
if (status == ACI_EVT_CONNECTED) {
rest.handle(BTLEserial);
}
}
```

<u>**Android code for Bluetooth Weather Station**</u>

Create a new project with the following properties

- Name: Bluetooth Weather Station
- Minimum SDK: 18
- Project: Empty Activity
- Activity Name: MainActivity
- Domain: your choice

This project will require two TextView objects. The upper TextView object will show all the Bluetooth callbacks, state changes, and characteristics written to the BLE module, while the lower TextView object will show the output from the temperature, light, and humidity sensor depending on which button was tapped.

The TextView objects will give them the following IDs:

- connectionStatusView
- dataOutputTextView

We will require three buttons reflecting the three parameters that we will be requesting, that is, temperature, light, and humidity.

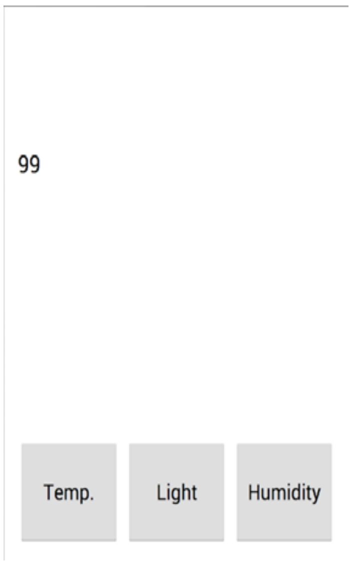
We will name the buttons as follows:

- The temperature button will be named as follows:
  - Text: Temperature
  - ID: temperatureButton
- The humidity button will be named as follows:
  - Text: Humidity
  - ID: humidityButton
- The light button will be named as follows:
  - Text: Light
  - ID: lightButton

Add the following code for AndroidManifest.xml file after <manifest> tag:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

The basic layout:



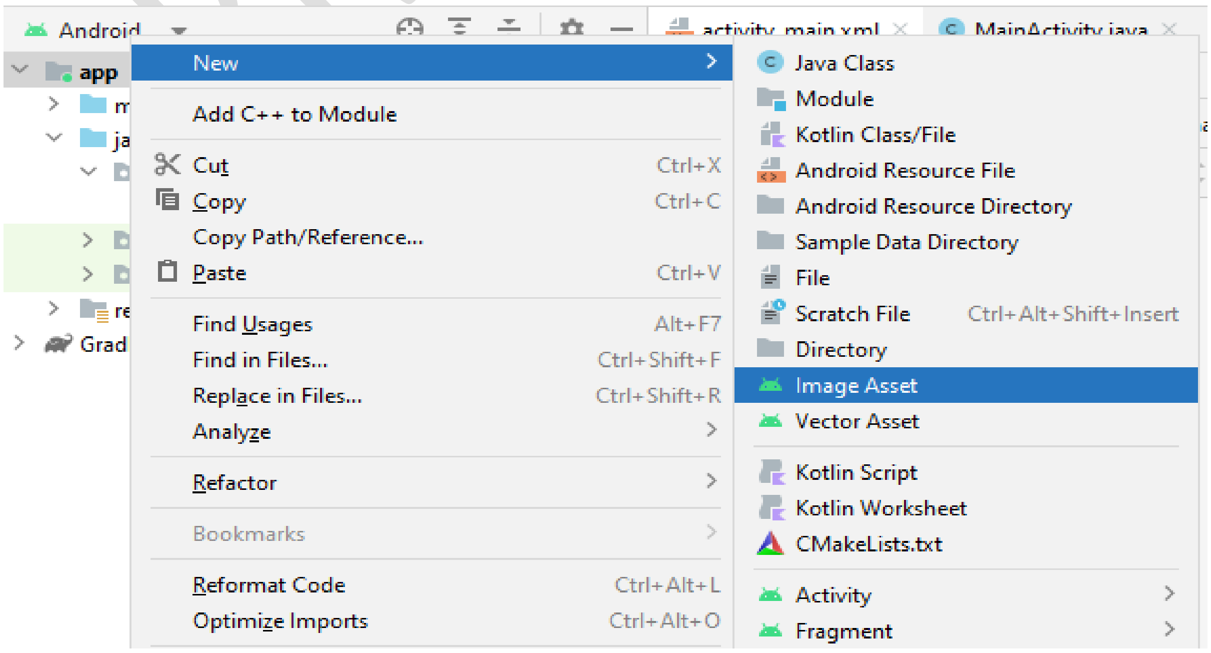
Enhancing the user interface:

We will work on the following main tasks:

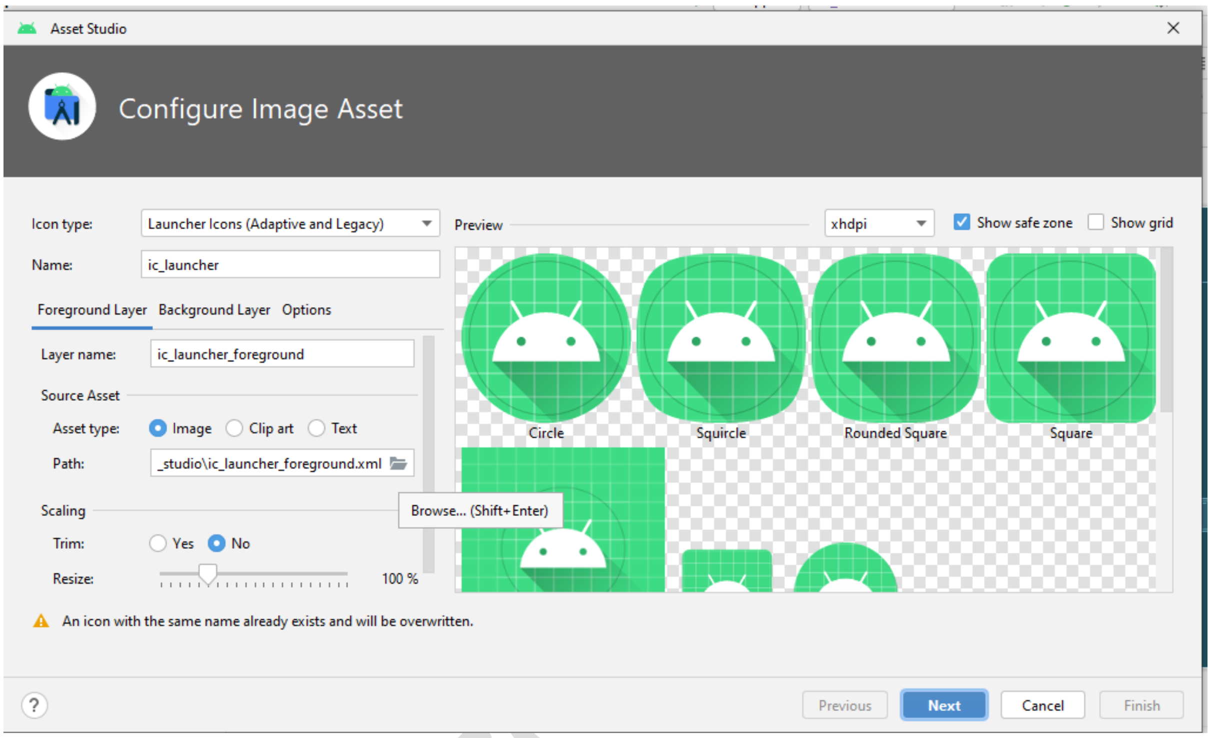
- Creating and adding our very own Android app icon
- Centering and enlarging the data output text
- Modifying the buttons and adding some color to our text

Creating and adding our very own Android app icon:

We should navigate using the project tree, followed by a right-click on the app folder, as shown in the following screenshot:



You will then be shown an Asset Studio pop-up window, which will allow you to choose your very own image file, as shown in the following screenshot. For optimization purposes, we recommend that you go for a .png file with a resolution of 144 pixels by 144 pixels. Android Studio automatically does all the resizing and resource creation to adapt your graphic to different screens:


Asset Studio

Configure Image Asset

Icon type: Launcher Icons (Adaptive and Legacy)

Name: ic_launcher

Foreground Layer | Background Layer | Options

Layer name: ic_launcher_foreground

Source Asset

Asset type: [✓ Image] Clip art Text

Path: _studio\ic_launcher_foreground.xml

Scaling

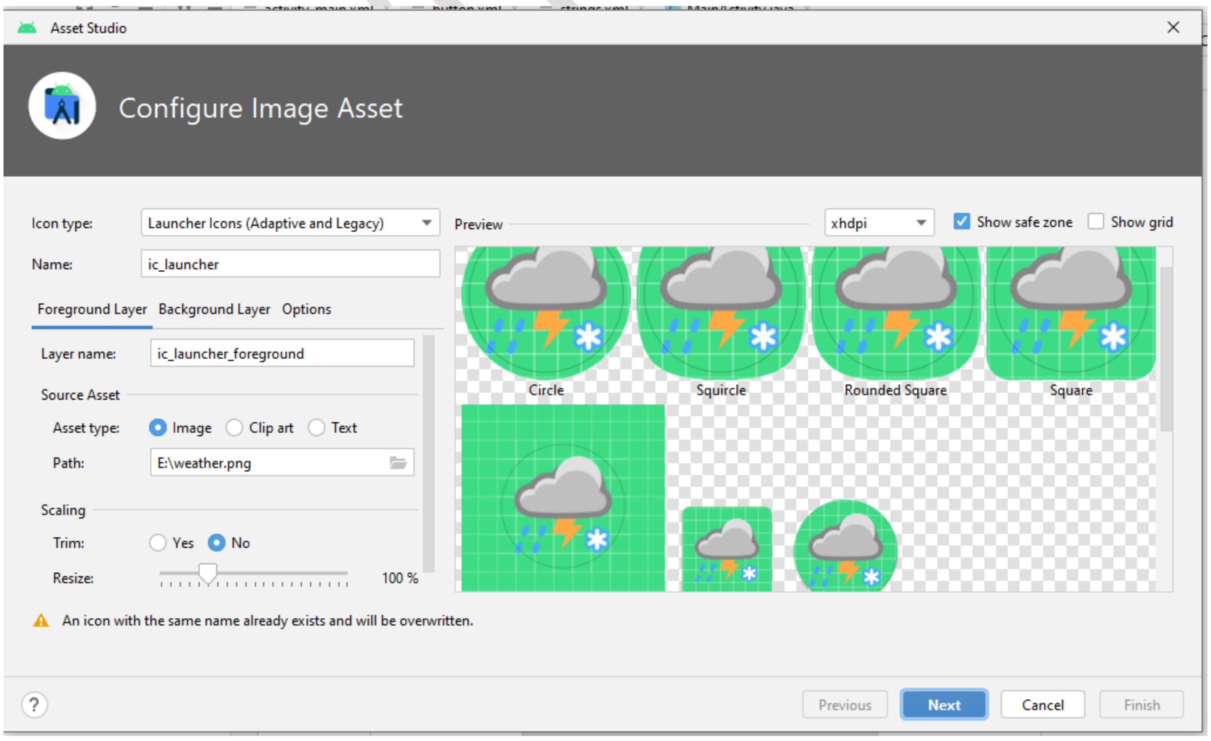
Trim: Yes [✓ No]

Resize: 100 %

Preview: xhdpi [✓ Show safe zone] Show grid

An icon with the same name already exists and will be overwritten.

Once you choose the ic_launcher image file, you will be shown a screen with the icon in different sizes. Click on Next, where you will see the following screen:


Asset Studio

Configure Image Asset

Icon type: Launcher Icons (Adaptive and Legacy)

Name: ic_launcher

Foreground Layer | Background Layer | Options

Layer name: ic_launcher_foreground

Source Asset

Asset type: [✓ Image] Clip art Text

Path: E:\weather.png

Scaling

Trim: Yes [✓ No]

Resize: 100 %

Preview: xhdpi [✓ Show safe zone] Show grid

An icon with the same name already exists and will be overwritten.

Centering and enlarging the data output text

In order to edit the layout for the main text output where the sensor data will be shown, we will need to open the project tree and navigate towards the layout file, which is available at app > src > main > res > layout > activity_main.xml.

Add the following code to the TextView in activity_main.xml

```
android:textSize="200dp"
android:gravity="center"
```

The whole block of code responsible for the Sensor Data Output TextView will now look as follows:

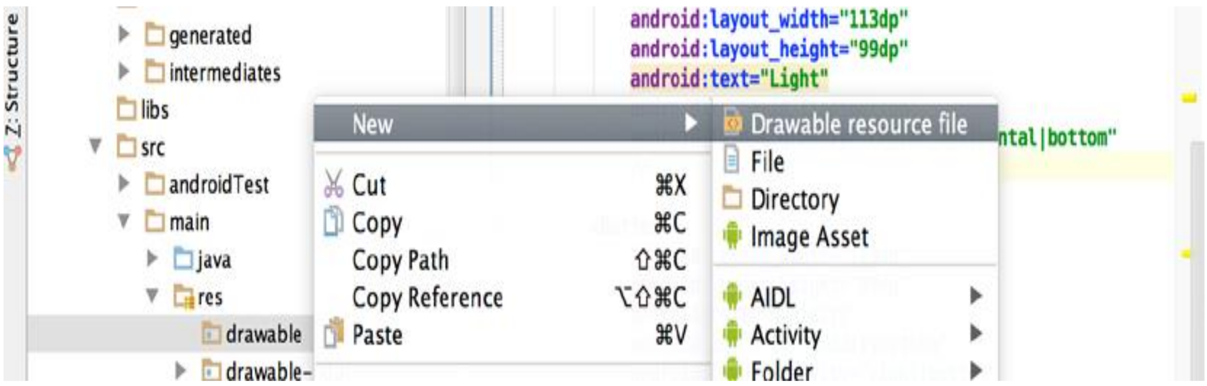
```
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceLarge"
android:id="@+id/dataOutputTextView"
android:layout_gravity="center_vertical"
android:textSize="200dp"
android:gravity="center"
android:text="99" />
```

In this block of code, we have temporarily used the placeholder text 99 so that we can approximate how it will look with the Android layout designer. With this modification, the sensor data output is now big enough to be seen by the user, thus enhancing the user experience.

Modifying the buttons and adding some color to our text

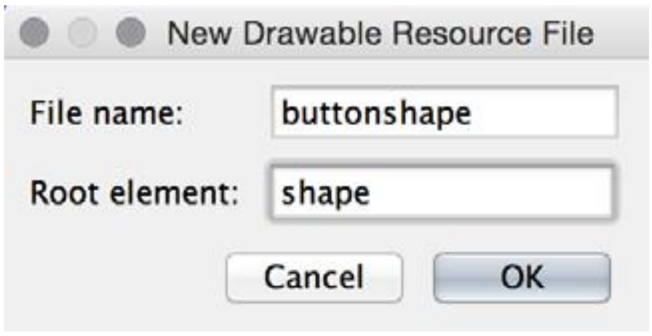
Finally, we will modify our buttons and add some color to the text by performing the following steps:

1. Create a drawable folder with a new XML drawable file known as buttonshape.xml.
2. We will then connect the drawable resource file to the main Android layout file.
3. Create the drawable folder by right-clicking on the res folder, which is available by navigating to app > src > main > res.
4. After creating the drawable folder within the res folder, we need to once again right-click on the new drawable folder and click on New and choose Drawable resource file, as shown in the following screenshot:



```
android:layout_width="113dp"
android:layout_height="99dp"
android:text="Light"
ntal|bottom"
```

5. Name the file buttonshape and type down shape as the Root element followed by clicking on OK, as shown in the following screenshot:



File name: buttonshape

Root element: shape

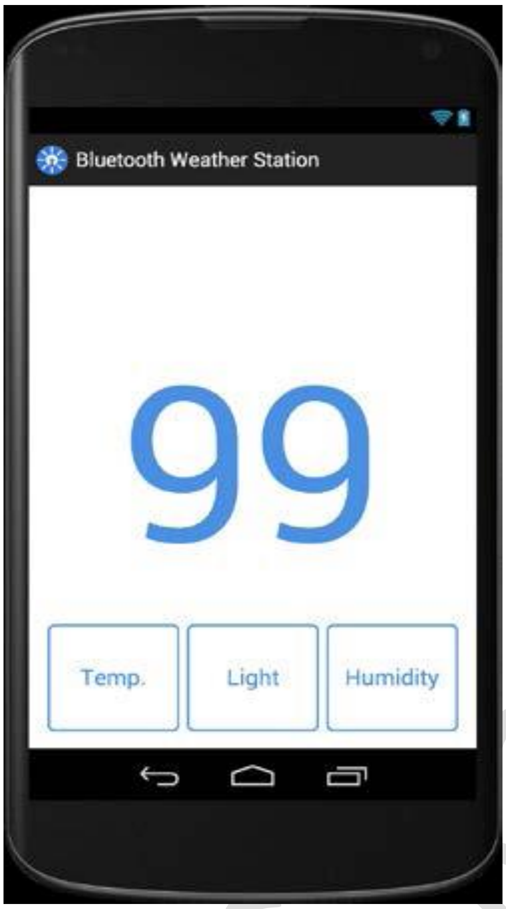
Within the buttonshape.xml file, we will add the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
android:shape="rectangle" >
<corners android:radius="10dp"/>
<solid android:color="#FFFFFF" />
<padding
android:left="0dp"
android:top="0dp"
android:right="0dp"
android:bottom="0dp"
/>
<size
android:width="85dp"
android:height="99dp"
/>
<stroke
android:width="2dp"
android:color="#4A90E2"
/>
</shape>
```

6. Then, we go towards the activity_main_screen.xml file and refer to this drawable by including the following line of code within the button modules:
android:background="@drawable/buttonshape"

7. We will also add some flavor by adding the following line of code to the button and TextView modules within the activity_main_screen.xml file:
android:textColor="#4A90E2"

In the preceding code, #4A90E2 refers to the hex code of the main color used in the app icon so that we maintain some consistency with the main user interface.
The final layout will look as follows on a Nexus 5 smartphone:



Android Code for MainScreen.java:

```
package com.arduinoandroid.ble_weather_station;


import android.app.Activity;
import android.bluetooth.*;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.util.*;
public class MainScreen extends Activity {

    private final String LOG_TAG = MainScreen.class.getSimpleName();

    // UUIDs for UAT service and associated characteristics.
    public static UUID UART_UUID = UUID.fromString("6E400001-B5A3-F393-E0A9-E50E24DCCA9E");
    public static UUID TX_UUID = UUID.fromString("6E400002-B5A3-F393-E0A9-E50E24DCCA9E");
    public static UUID RX_UUID = UUID.fromString("6E400003-B5A3-F393-E0A9-E50E24DCCA9E");

    // UUID for the BTLE client characteristic which is necessary for notifications.
    public static UUID CLIENT_UUID = UUID.fromString("00002902-0000-1000-8000-00805f9b34fb");

    // UI elements
    private TextView dataOutput;
    private TextView connectionOutput;
    private Button temperature;
    private Button light;
    private Button humidity;
    private String output;
    private String output2;

    // BTLE state
    private BluetoothAdapter adapter;
    private BluetoothGatt gatt;
    private BluetoothGattCharacteristic tx;
    private BluetoothGattCharacteristic rx;

    // Main BTLE device callback where much of the logic occurs.
    private BluetoothGattCallback callback = new BluetoothGattCallback() {
      // Called whenever the device connection state changes, i.e. from disconnected to connected.
      @Override
      public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
        super.onConnectionStateChange(gatt, status, newState);
        if (newState == BluetoothGatt.STATE_CONNECTED) {
          writeLine("Connected!");
          // Discover services.
          if (!gatt.discoverServices()) {
            writeLine("Failed to start discovering services!");
```

```
            }
        }
        else if (newState == BluetoothGatt.STATE_DISCONNECTED) {
            writeLine("Disconnected!");
        }
        else {
            writeLine("Connection state changed.  New state: " + newState);
        }
    }

// OnCreate, called once to initialize the activity.
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main_screen);

    // Grab references to UI elements.
    dataOutput = (TextView) findViewById(R.id.dataOutputTextView);
    connectionOutput = (TextView) findViewById(R.id.connectionStatusView);

    adapter = BluetoothAdapter.getDefaultAdapter();
    temperature = (Button) findViewById(R.id.temperatureButton);
    light = (Button) findViewById(R.id.lightButton);
    humidity = (Button) findViewById(R.id.humidityButton);

    // Set Commands to be send to Bluetooth Module

    temperature.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            String setTempMessage = "/temperature /";
            tx.setValue(setTempMessage.getBytes(Charset.forName("UTF-8")));
            if (gatt.writeCharacteristic(tx)) {
```

```
            writeLine("Sent: " + setTempMessage);
        } else {
            writeLine("Couldn't write TX characteristic!");
        }

    }
});

light.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String setLightMessage = "/light /";
        tx.setValue(setLightMessage.getBytes(Charset.forName("UTF-8")));
        if (gatt.writeCharacteristic(tx)) {
            writeLine("Sent: " + setLightMessage);
        }
        else {
            writeLine("Couldn't write TX characteristic!");
        }
    }
});

humidity.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String setHumidityMessage = "/humidity /";
        tx.setValue(setHumidityMessage.getBytes(Charset.forName("UTF-8")));
        if (gatt.writeCharacteristic(tx)) {
            writeLine("Sent: " + setHumidityMessage);
        }
        else {
            writeLine("Couldn't write TX characteristic!");
        }
    }
```

```
  });
}

// OnResume, called right before UI is displayed.  Start the BTLE connection.
@Override
protected void onResume() {
  super.onResume();
  // Scan for all BTLE devices.
  // The first one with the UART service will be chosen--see the code in the scanCallback.
  writeLine("Scanning for devices...");
  adapter.startLeScan(scanCallback);
}

// OnStop, called right before the activity loses foreground focus.  Close the BTLE connection.
@Override
protected void onStop() {
  super.onStop();
  if (gatt != null) {
    // For better reliability be careful to disconnect and close the connection.
    gatt.disconnect();
    gatt.close();
    gatt = null;
    tx = null;
    rx = null;
  }
}
```

Proejct-2 Pulse Rate Sensor:

Hardware Requirements:

- Arduino Uno board
- nRF8001 breakout board
- Heard rate sensor

- Breadboard
- Jumper wires

Software Requirements:

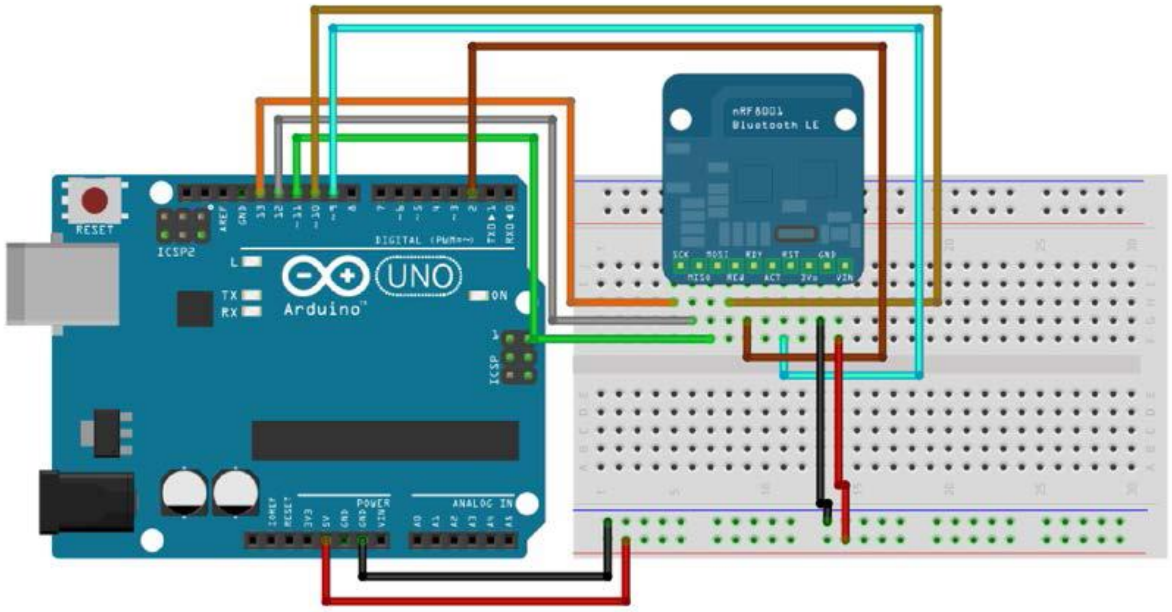
- The library for the nRF8001 chip, available at https://github.com/adafruit/Adafruit_nRF8001
- The aREST library to send commands to the robot, available at https://github.com/marcoschwartz/aREST

Configuring our hardware:

We are now going to build the project by performing the following steps:

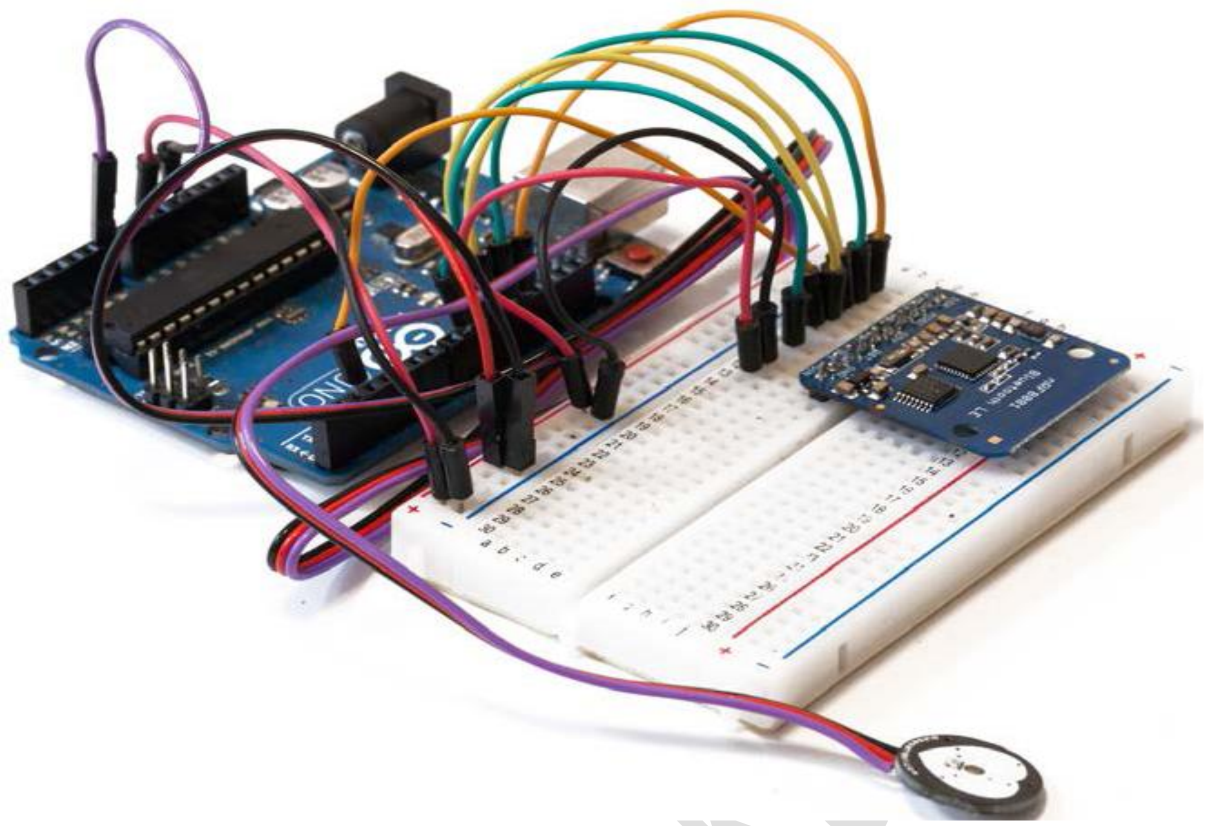
1. First, connect the BLE breakout board to the Arduino Uno board.
2. Place the module on the breadboard.
3. Connect the power supply of the module: **GND** goes to prototyping shield **GND** and **VIN** goes to the prototyping shield +5V.
4. Connect the different wires responsible for the SPI interface: **SCK** to Arduino pin **13**, **MISO** to Arduino pin **12**, and **MOSI** to Arduino pin **11**.
5. Then connect the **REQ** pin to Arduino pin 10.
6. Finally, connect the **RDY** pin to Arduino pin **2** and the **RST** pin to Arduino pin **9**.

The following is a schematic diagram to help you out for this part:



7. Now, connecting the pulse rate sensor is actually very simple. You simply need to connect the red wire to the Arduino +5V pin, the black cable to the Arduino GND pin, and the remaining pin to the Arduino A0 pin.

This is an image of the fully assembled project:



Testing the sensor

```
// Sensor and pins variables
int pulsePin = 0;
int blinkPin = 13;
int fadePin = 5;
int fadeRate = 0;

// Pulse rate variable
volatile int BPM;

// Raw signal
volatile int Signal;

// Interval between beats
volatile int IBI = 600;

// Becomes true when the pulse is high
volatile boolean Pulse = false;

// Becomes true when Arduino finds a pulse
volatile boolean QS = false;

void setup(){

  // Update pin that will blink to your heartbeat
  pinMode(blinkPin,OUTPUT);

  // Update pin that will fade to your heartbeat!
  pinMode(fadePin,OUTPUT);

  // Start Serial
  Serial.begin(115200);

  // Sets up to read Pulse Sensor signal every 2mS
  interruptSetup();
}

void loop(){

  // If heart beat is found
  if (QS == true) {

      // Set 'fadeRate' Variable to 255 to fade LED with pulse
      fadeRate = 255;

      // Print heart rate with a 'B' prefix
      sendDataToProcessing('B',BPM);

      // Reset the Quantified Self flag for next time
      QS = false;
    }

  // Fade LED
  ledFadeToBeat();

  // Wait 20 ms
  delay(20);
}
```

Complete Arduino Sketch for Pulse Rate Sensor

```
#include <SPI.h>
#include "Adafruit_BLE_UART.h"
#include <aREST.h>

// Sensor and pins variables
int pulsePin = 0;
int blinkPin = 13;
int fadePin = 5;
int fadeRate = 0;

// Pins
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2     // This should be an interrupt pin, on Uno thats #2 or #3
#define ADAFRUITBLE_RST 9

// Create aREST instance
aREST rest = aREST();

// BLE instance
```

```
Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ,
ADAFRUITBLE_RDY, ADAFRUITBLE_RST);

// Pulse rate variable
volatile int BPM;

// Raw signal
volatile int Signal;

// Interval between beats
volatile int IBI = 600;

// Becomes true when the pulse is high
volatile boolean Pulse = false;

// Becomes true when Arduino finds a pulse
volatile boolean QS = false;

// Variable to be returned by Bluetooth LE
int bpm = 0;


void setup(){

  // Start Serial
  Serial.begin(115200);

  // Start BLE
  BTLEserial.begin();

  // Give name and ID to device
  rest.set_id("1");
  rest.set_name("pulse_sensor");

  // Expose variables to API
  rest.variable("bpm",&bpm);

  // Sets up to read Pulse Sensor signal every 2mS
  interruptSetup();
}

void loop(){

  // Assign BPM variable
  bpm = BPM;

  // Tell the nRF8001 to do whatever it should be working on.
  BTLEserial.pollACI();

  // Ask what is our current status
  aci_evt_opcode_t status = BTLEserial.getState();
  // Handle REST calls
  if (status == ACI_EVT_CONNECTED) {
    rest.handle(BTLEserial);
  }
}
```

Setting up the Android app

Create a new project with the following properties

- Name: Bluetooth Weather Station
- Minimum SDK: 18
- Project: Empty Activity
- Activity Name: MainActivity
- Domain: your choice

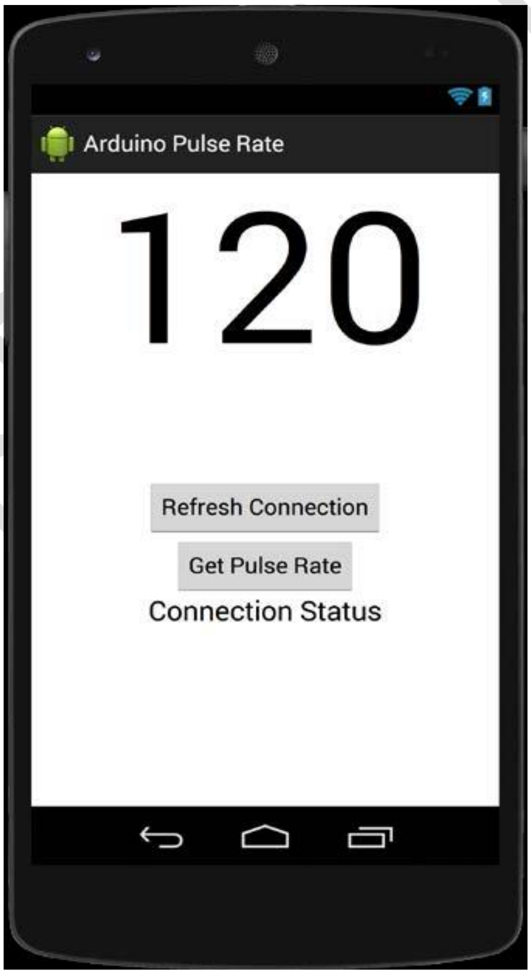
Add the following code to AndroidManifest.xml file

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Replace the activity_main.xml file with the following code:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <LinearLayout
        android:id="@+id/rest"
        android:layout_width="fill_parent"
        android:layout_height="250dip"
        android:orientation="vertical"
        android:weightSum="1">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:id="@+id/pulseValueView"
            android:layout_gravity="center_horizontal"
            android:textSize="150dp"
            android:gravity="center"
            android:text="120"/>
    </LinearLayout>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Refresh Connection"
        android:id="@+id/refreshBtn"
        android:layout_gravity="center_horizontal" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Get Pulse Rate"
        android:id="@+id/heartRateBtn"
        android:layout_gravity="center_horizontal" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Connection Status"
        android:id="@+id/connectionStsView"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

The end result will look as follows within the IDE:



The text 120 is meant to be a placeholder text to ensure that there is enough place within the user interface to accommodate the pulse rate readings. In the final implementation, you have the option of removing the placeholder text and leaving it blank.

Android Code for MainActivity.java

```
package com.example.arduinopulserate;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import java.nio.charset.Charset;

public class PulseActivity extends Activity {

    //Logging Variables
    private final String LOG_TAG = PulseActivity.class.getSimpleName();

    //User Interface Variables
    Button getPulseRate;
    Button refreshButton;
    TextView pulseRateView;
    TextView connectionStsView;

    private BluetoothAdapter adapter;
    private BluetoothGatt gatt;
    private BluetoothGattCharacteristic tx;
    private BluetoothGattCharacteristic rx;

    private boolean areServicesAccessible = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pulse);

        //Connect U.I Elements
        getPulseRate = (Button) findViewById(R.id.heartRateBtn);
        pulseRateView = (TextView) findViewById(R.id.pulseValueView);
        connectionStsView = (TextView) findViewById(R.id.connectionStsView);
        refreshButton = (Button) findViewById(R.id.refreshBtn);


        getPulseRate.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String setOutputMessage = "/bpm /";
                tx.setValue(setOutputMessage.getBytes(Charset.forName("UTF-8")));
                if (gatt.writeCharacteristic(tx)) {
                    writeConnectionData("Sent: " + setOutputMessage);
                } else {
                    writeConnectionData("Couldn't write TX characteristic!");
                }
            }
        });

        refreshButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                restartScan();
            }
        });
    }

    private void writeConnectionData(final CharSequence text) {
        Log.e(LOG_TAG, text.toString());
        connectionStsView.setText(text.toString());
    }

//Implement Method Below to output pulse rate sensor readings to pulseRateView TextView
    private void writeSensorData(final CharSequence text) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Log.e(LOG_TAG,text.toString());
                output=text.toString().trim();

                if (output.length() > 0 && output.length() <=3) {
                    pulseRateView.setText(output);
                }
                else {
                    return;
                }
            }
        });
    }

    // BTLE device scanning bluetoothGattCallback.

    // Main BTLE device bluetoothGattCallback where much of the logic occurs.
    private BluetoothGattCallback bluetoothGattCallback = new BluetoothGattCallback() {


        // Called when services have been discovered on the remote device.
        // It seems to be necessary to wait for this discovery to occur before
        // manipulating any services or characteristics.
        public void onServicesDiscovered(BluetoothGatt gatt, int status) {
            super.onServicesDiscovered(gatt, status);
            if (status == BluetoothGatt.GATT_SUCCESS) {
               writeConnectionData("Service discovery completed!");
            } else {
               writeConnectionData("Service discovery failed with status: " + status);
            }
            }

   protected void onStart() {
      Log.d(LOG_TAG,"onStart has been called");
      super.onStart();
      // / Scan for all BTLE devices.
      // The first one with the UART service will be chosen--see the code in the scanCallback.
      adapter = BluetoothAdapter.getDefaultAdapter();
      startScan();
   }

   //BLUETOOTH METHODS
   private void startScan() {
      if (!adapter.isEnabled()) {
         adapter.enable();
      }
      if (!adapter.isDiscovering()) {
         adapter.startDiscovery();
      }
      writeConnectionData("Scanning for devices...");
      adapter.startLeScan(scanCallback);
   }

   private void stopScan() {
      if (adapter.isDiscovering()) {
         adapter.cancelDiscovery();
      }
      writeConnectionData("Stopping scan");
      adapter.stopLeScan(scanCallback);
   }

   private void restartScan() {
      stopScan();
      startScan();
   }

  }
}

}
```