Mobile Application Development for IoT Unit-2 Material

Syllabus:

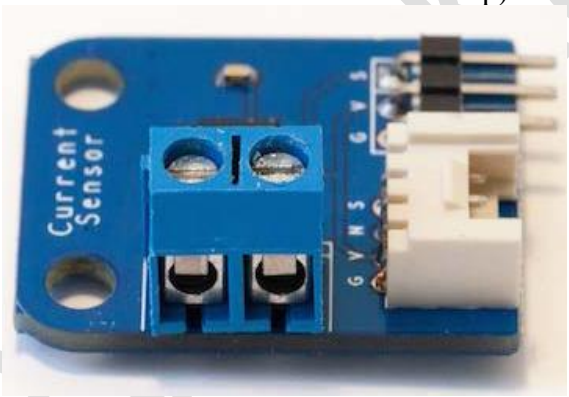
Wi-Fi Smart Power Plug: Hardware and Software requirements, Writing the Arduino sketch **Control an Arduino Board via NFC:** Hardware and Software requirements, Writing the Arduino sketch

Project-1: Wifi Smart Power Plug

Hardware Requirements

- The Arduino Uno board
- The 5V relay module
- The Current sensor to measure instant power consumption
- The Adafruit CC3000 Wi-Fi breakout board to receive commands via the Android device
- The breadboard
- Jumper wires

The Current Sensor (Breakout board based on ACS712 chip)



To connect a lamp or any other device, we will need a pair of power cables as shown below

Software Requirements (Arduino)

- Arduino IDE
- Library for the CC3000 chip found at https://github.com/adafruit/Adafruit_CC3000_Library
- aREST library found at https://github.com/marcoschwartz/aREST

Configuring the hardware

Step-1: Connect the Arduino Uno +5V pin to the red rail on the breadboard and ground pin to the blue rail

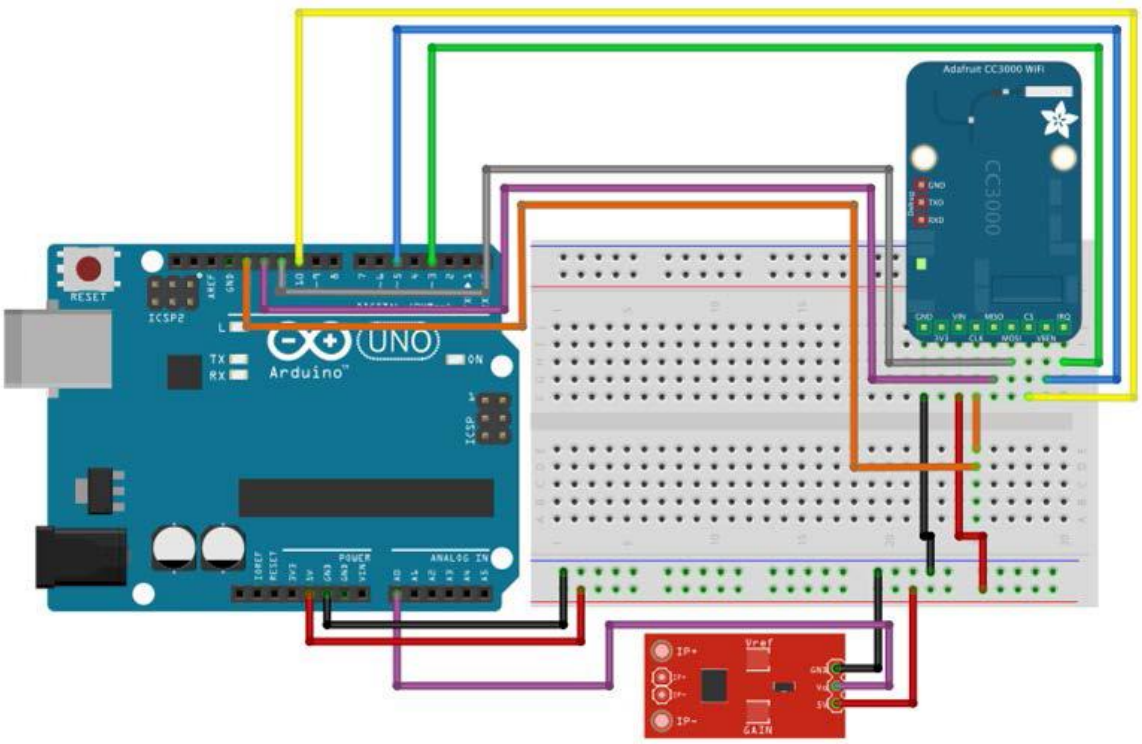
Step-2: Connect the IRQ (Interrupt Request) pin of the CC3000 board to pin number 3 of the Arduino board, VBAT(Voltage Battery) to pin 5, and CS(Chip Select) to pin 10.

Step-3: Connect the SPI(Serial Peripheral Interface) pins to the Arduino board: MOSI(Master Out Slave In), MISO(Master In Slave Out), and CLK(Clock) go to pins 11, 12 and 1 respectively.

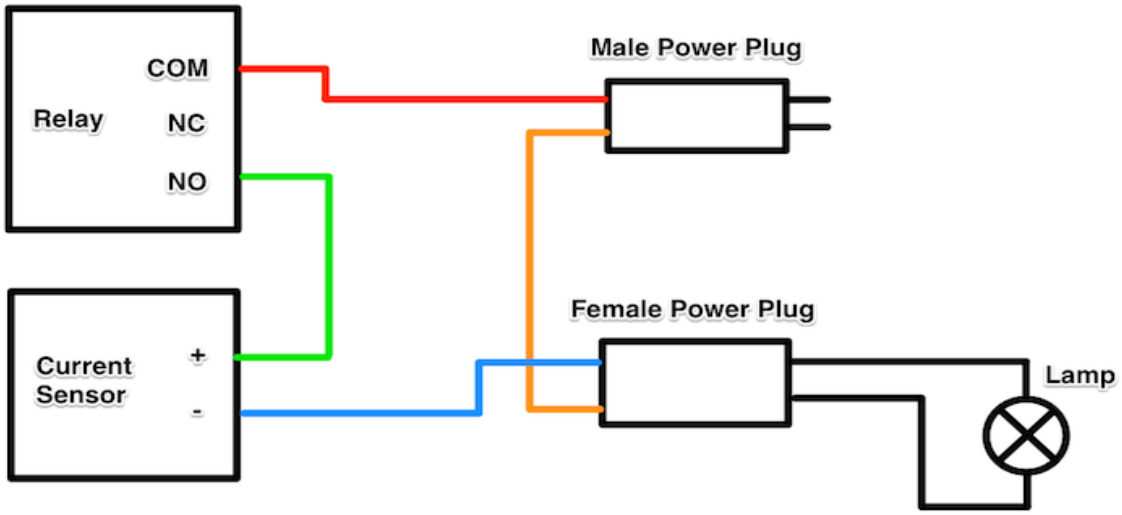
Step-4: VIN goes to the Arduino 5V and GND to GND

Step-5: Connect to the power supply: the VCC pin of the relay goes to the red power rail, and the GND pin goes to the blue power rail, SIG pin to Arduino pin number 8

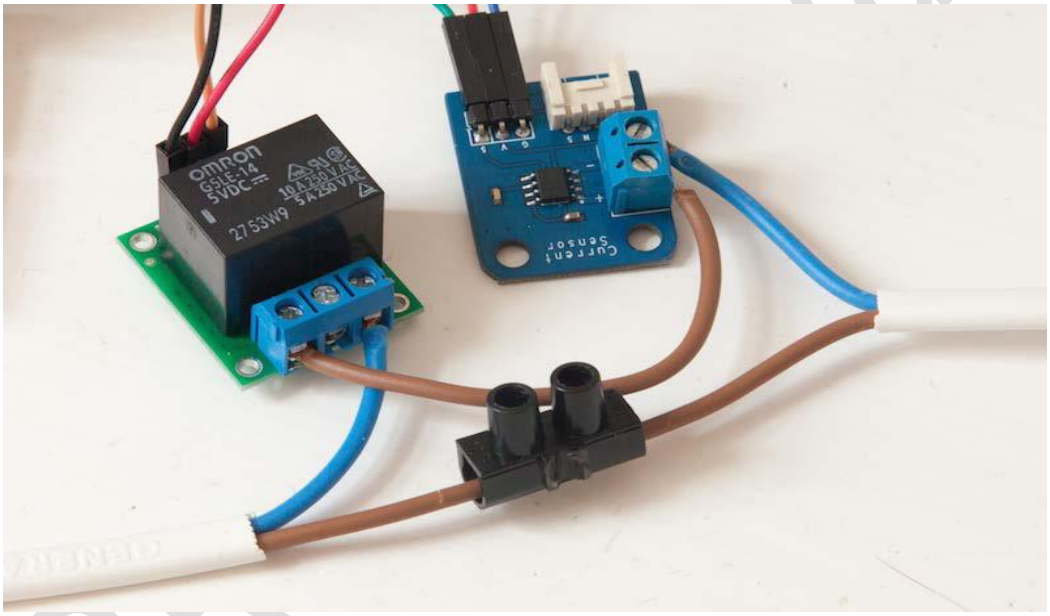
The following is a schematic of the project, without the relay module connected:



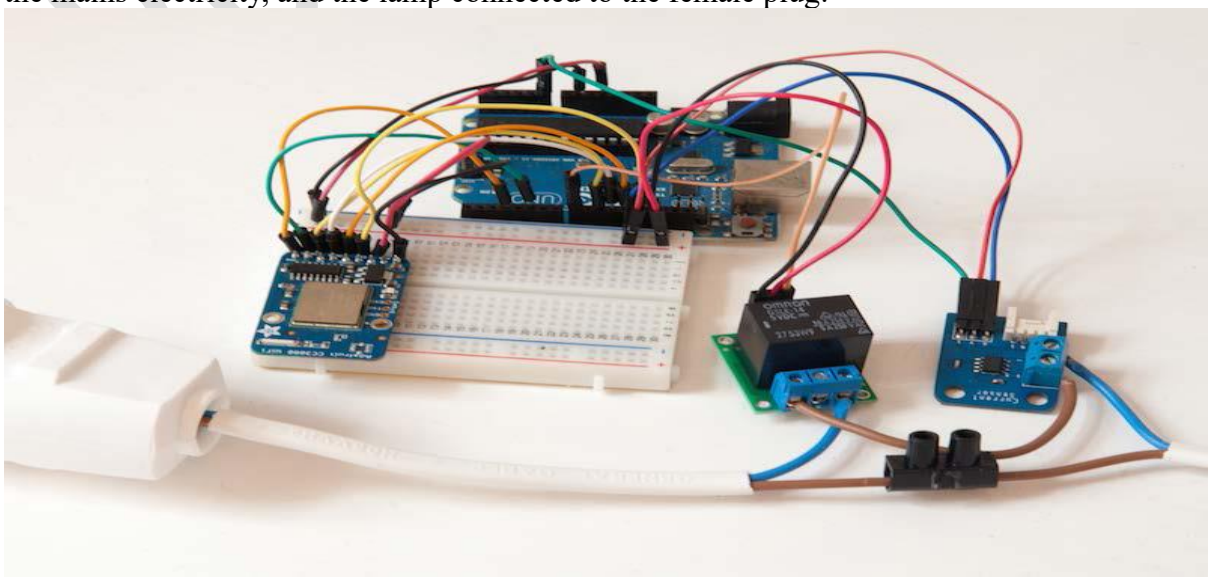
The following is a schematic describes how the different cables are connected to the relay and to the current sensor



The following is an image illustrating the different connections between the cables, the relay, and the current sensor



Finally, the following is an image of the complete project, with the male cable connected to the mains electricity, and the lamp connected to the female plug:



Testing the relay

Testing the relay will ensure that the relay is correctly connected to your Arduino board, and that the power cable connections are correctly done.

Refer the following Arduino sketch

```
// Relay pin
const int relay_pin = 8;
void setup() {
pinMode(relay_pin,OUTPUT);
}
void loop() {
// Activate relay
digitalWrite(relay_pin, HIGH);
// Wait for 5 seconds
delay(5000);
// Deactivate relay
digitalWrite(relay_pin, LOW);
// Wait for 5 seconds
delay(5000);
}
```

Complete Arduino sketch for Wi-Fi Smart Power Plug

```
// Import required libraries
#include “Adafruit_CC3000.h”
#include “SPI.h”
#include “aREST.h”
// Relay state
const int relay_pin = 8;
// Define measurement variables
float amplitude_current;
float effective_value;
float effective_voltage = 230.; // Set voltage to 230V
float zero_sensor;
// These are the pins for the CC3000 chip if you are using a breakout
board
#define ADAFRUIT_CC3000_IRQ 3
#define ADAFRUIT_CC3000_VBAT 5
#define ADAFRUIT_CC3000_CS 10
// Create CC3000 instance
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT,
SPI_CLOCK_DIV2);
// Create aREST instance
aREST rest = aREST();
// Your WiFi SSID and password
#define WLAN_SSID "yourWiFiNetworkName"
#define WLAN_PASS "yourPassword"
#define WLAN_SECURITY WLAN_SEC_WPA2
// The port to listen for incoming TCP connections
#define LISTEN_PORT 80
```

```
// Server instance
Adafruit_CC3000_Server restServer(LISTEN_PORT);
// Variables to be exposed to the API
int power;
void setup(void)
{
// Start Serial
Serial.begin(9600);
// Init variables and expose them to REST API
rest.variable("power",&power);
// Set relay pin to output
pinMode(relay_pin,OUTPUT);
// Calibrate sensor with null current
zero_sensor = getSensorValue(A0);
// Give name and ID to device
rest.set_id("001");
rest.set_name((char*)"smart_lamp");
// Set up CC3000 and get connected to the wireless network.
if (!cc3000.begin())
{
while(1);
}
if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
while(1);
}
while (!cc3000.checkDHCP())
{
delay(100);
}
// Display connection details
displayConnectionDetails();
// Start server
restServer.begin();
Serial.println(F("Listening for connections..."));
}
void loop() {
// Perform power measurement
float sensor_value = getSensorValue(A0);
// Convert to current
amplitude_current = (float)(sensor_value-zero_sensor)/1024*5/185*1000000;
effective_value = amplitude_current/1.414;
power = (int)(abs(effective_value*effective_voltage/1000));
// Handle REST calls
Adafruit_CC3000_ClientRef client = restServer.available();
rest.handle(client);
}
// Function to display connection details
bool displayConnectionDetails(void)
{
uint32_t ipAddress, netmask, gateway, dhcpserv, dnsserv;
if(!cc3000.getIPAddress(&ipAddress, &netmask, &gateway, &dhcpserv,
&dnsserv))
```

```
{
Serial.println(F("Unable to retrieve the IP Address!\r\n"));
return false;
}
else
{
Serial.print(F("\nIP Addr: ")); cc3000.printIPdotsRev(ipAddress);
Serial.print(F("\nNetmask: ")); cc3000.printIPdotsRev(netmask);
Serial.print(F("\nGateway: ")); cc3000.printIPdotsRev(gateway);
Serial.print(F("\nDHCPsrv: ")); cc3000.printIPdotsRev(dhcpserv);
Serial.print(F("\nDNSserv: ")); cc3000.printIPdotsRev(dnsserv);
Serial.println();
return true;
}
}
// Get the reading from the current sensor
float getSensorValue(uint8_t pin)
{
uint16_t sensorValue;
float avgSensor = 0;
uint8_t nb_measurements = 100;
for (uint8_t i = 0; i < nb_measurements; i++) {
sensorValue = analogRead(pin);
avgSensor = avgSensor + float(sensorValue);
}
avgSensor = avgSensor/float(nb_measurements);
return avgSensor;
}
```

Implementation of Wifi Smart Power Plug

Step-1: Create a New Project with the following setup:

- Name: Minimum SDK: 15
- Project: Empty Activity
- Activity Name: MainActivity

Step-2: Edit the activity_main.xml and change the layout as ‘relative layout’ by editing the code as

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

Step-3: Drag-and-drop four buttons together with aTextView, which will look as shown in the following figure

Step-4: Identify each user interface item as follows:

- The Open the Gate button: openGateButton
- The Switch On button: switchOnButton
- The Switch Off button: switchOffButton
- The Check Power button: checkPowerButton
- The Power Output text view: powerOutput

Step-5: Add the following code to AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

Android Code for MainActivity.java

```
import android.app.AlertDialog;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.widget.Button;
import android.widget.TextView;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.BufferedReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
public class MainScreen extends Activity {
   public static final String TAG = MainScreen.class.getSimpleName();
   final Activity activity = this;
   public static final String URL = "192.168.1.3";
   //Main Thread
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main_screen);
```

```
//Declare our View Variables and assign them to the layout elements
        Button checkPowerButton = (Button) findViewById(R.id.checkPowerButton);
        Button openTheGateButton = (Button) findViewById(R.id.openGateButton);
        Button switchOnButton = (Button) findViewById(R.id.switchOnButton);
        Button switchOffButton = (Button) findViewById(R.id.switchOffButton);
        checkPowerButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (isNetworkAvailable()) {
                    checkPowerTask getPowerTask = new checkPowerTask();
                    getPowerTask.execute();
                }
            }       });

openTheGateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (isNetworkAvailable()) {
                    SwitchOpenTask switchOpenTask = new SwitchOpenTask();
                    switchOpenTask.execute();
                }
            }
        });
switchOnButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (isNetworkAvailable()) {
                    SwitchOnTask switchOnTask = new SwitchOnTask();
                    switchOnTask.execute();
                }
            }
        });
switchOffButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (isNetworkAvailable()) {
                    SwitchOffTask switchOffTask = new SwitchOffTask();
                    switchOffTask.execute();
                }
            }
        });

    }
private class SwitchOpenTask extends AsyncTask<Object,Void,String> {
    }
    private class SwitchOnTask extends AsyncTask<Object,Void,String> {
    }
    private class SwitchOffTask extends AsyncTask<Object,Void,String> {
    }
    private class checkPowerTask extends AsyncTask<Object,Void,String> {
    }
}
```

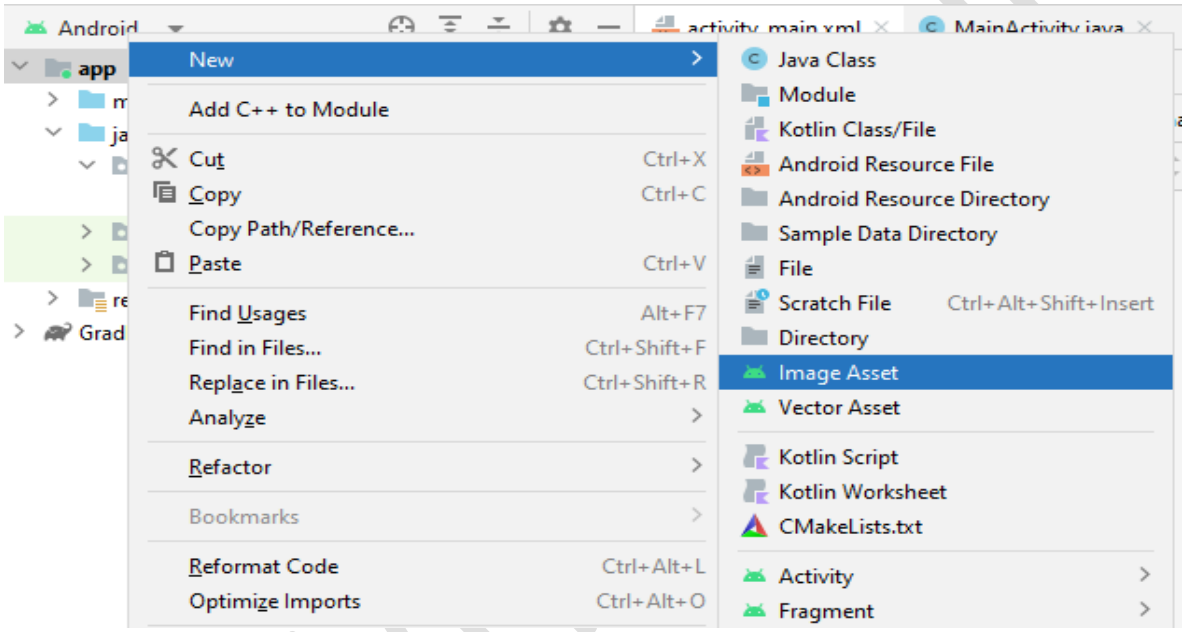
Polishing the user interface and experience

We will improve the user interface with the following main actions:

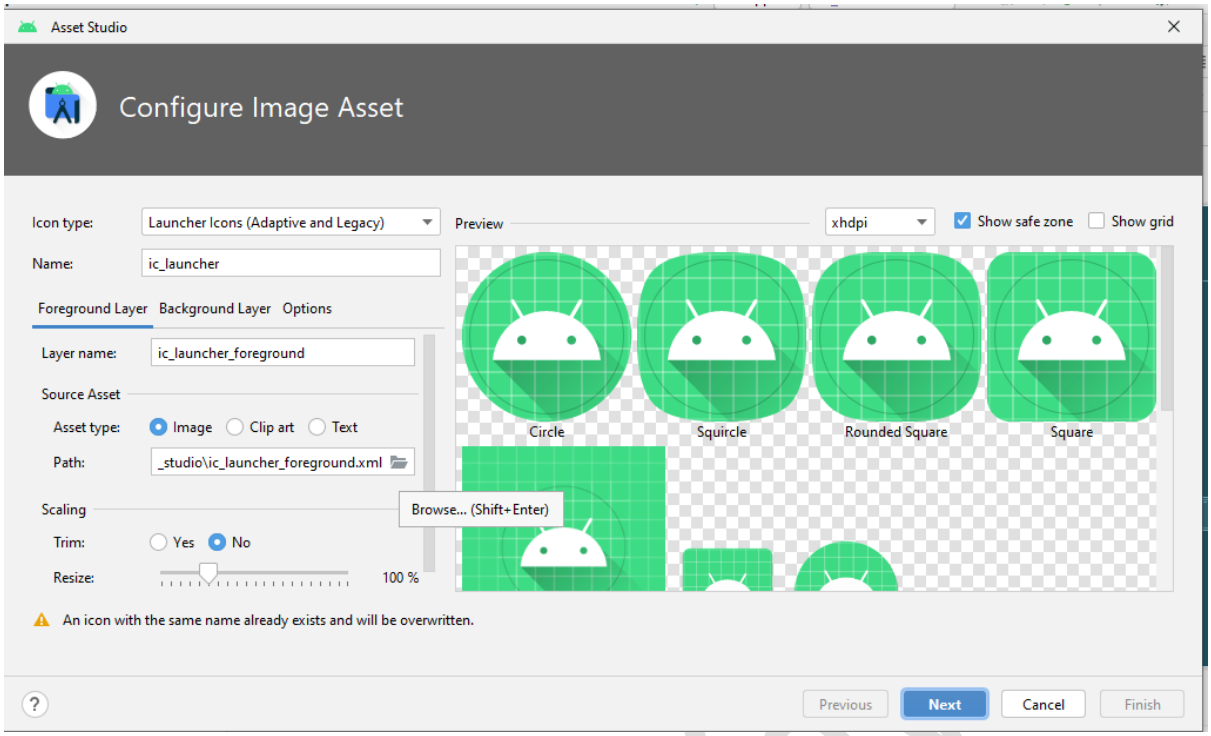
- Adding a new app icon
- Enlarging the power output text
- Aligning and styling the buttons
- Changing the application name in the action bar

Adding a new app icon

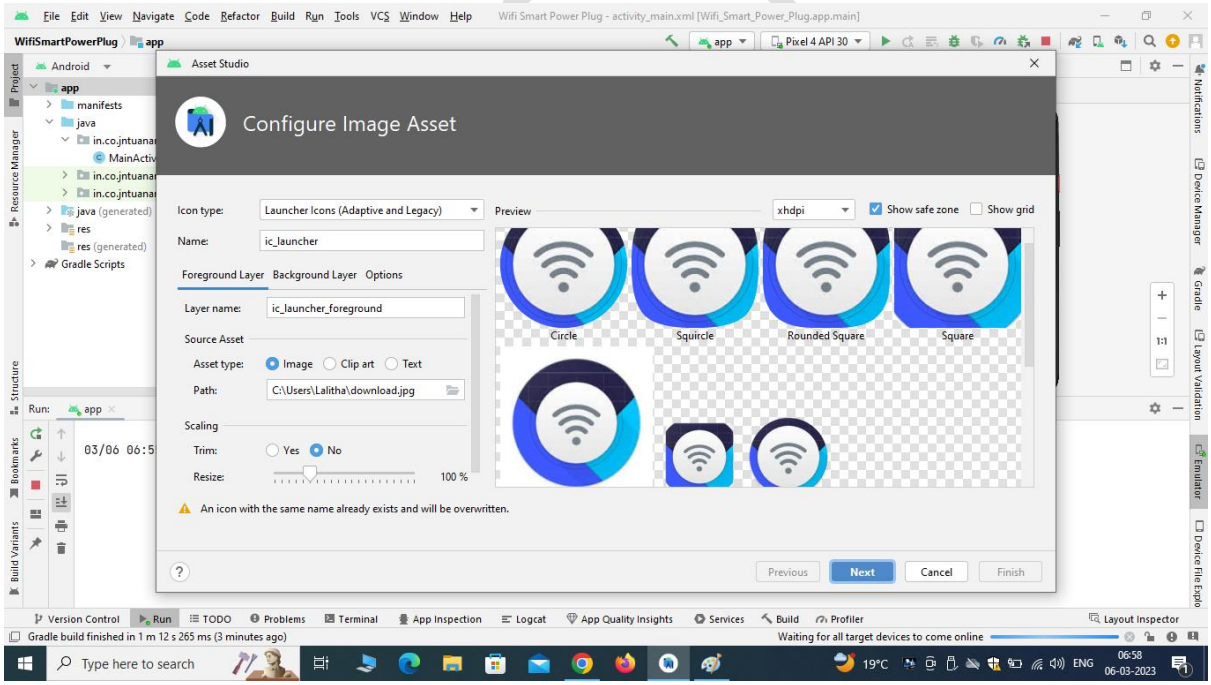
We should navigate using the project tree, followed by a right-click on the app folder, as shown in the following screenshot:



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:



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



This screen warns you that the previous files will be overwritten and shows you the image launcher file in a number of different resolutions once again. Click on Finish, compile the app, launch it on your physical device, and you should see something pleasant in your app tray and in the app's action bar, which is shown as follows:

Enlarging the power 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 > res > layout > activity_main.xml.
- Once in this view, we recommend that you modify the text using the text view. This will allow you finer control and get you used to the different conventions used when editing Android layout files programmatically.
- When opening the activity_main.xml file, we will see the different XML codes for the buttons and Text Views. At this point, look out for the code that takes care of the Power Data Output TextView and add the following code:

```
android:textSize="100sp"
android:textAlignment="center"
```

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

```
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="100W"
android:textSize="100sp"
android:id="@+id/powerOutput"
android:textAlignment="center"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"
android:layout_marginTop="78dp"
/>
```

In this block of code, we temporarily used the placeholder text 100W so that we can approximate how it will look with the Android layout designer. With this modification, the sensor data is now big enough to show to the user and will be part of the enhancement within the user experience.

Aligning and styling the buttons

For our final steps, we will modify our buttons and add some color to the text.
There will be two steps when creating the new buttons:

1. Create a new XML file button.xml (Right click on ‘drawable’ folder in ‘app’ folder of project explorer and select new->drawable resource file
2. We will then connect the drawable resource file to the main Android layout file.

Within the button.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="30dp"/>
<solid
android:color="#FFFFFF"/>
<padding
android:left="0dp"
android:top="0dp"
android:right="0dp"
android:bottom="0dp"/>
<size
android:width="120dp"
android:height="60dp"/>
<stroke
android:width="2dp"
android:color="#4A90E2"/>
</shape>
```

Then, we go towards the activity_main.xml file and refer to this drawable by including the following line of code within the button modules:

```
android:background="@drawable/button"
```

We will add some flavor by adding the following line of code to the Button and TextView modules within the activity_main.xml file:

```
android:textColor="#4A90E2"
```

The #4A90E2 term 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.

Changing the application name within the action bar

We all would like to customize the name of the app to one of our own likings and that will be the easiest thing within our project! We will just go over to the strings.xml file where we have all our constant text values within the project. This is available at

app > res > values > string.xml.

Then, you can change the text of arduinoWifi to any name of your liking. In this case, we will stick to WiFi Lamp Switch:

```
<string name="app_name">WiFi Lamp Switch</string>
```

Our final project should now look as follows (device used in this case is a Nexus 4):

Project-2: Control an Arduino Board via NFC:

Hardware Requirements

- The Arduino Uno board
- The 5V relay module
- The Arduino NFC shield
- The breadboard
- Jumper wires

Software Requirements (Arduino)

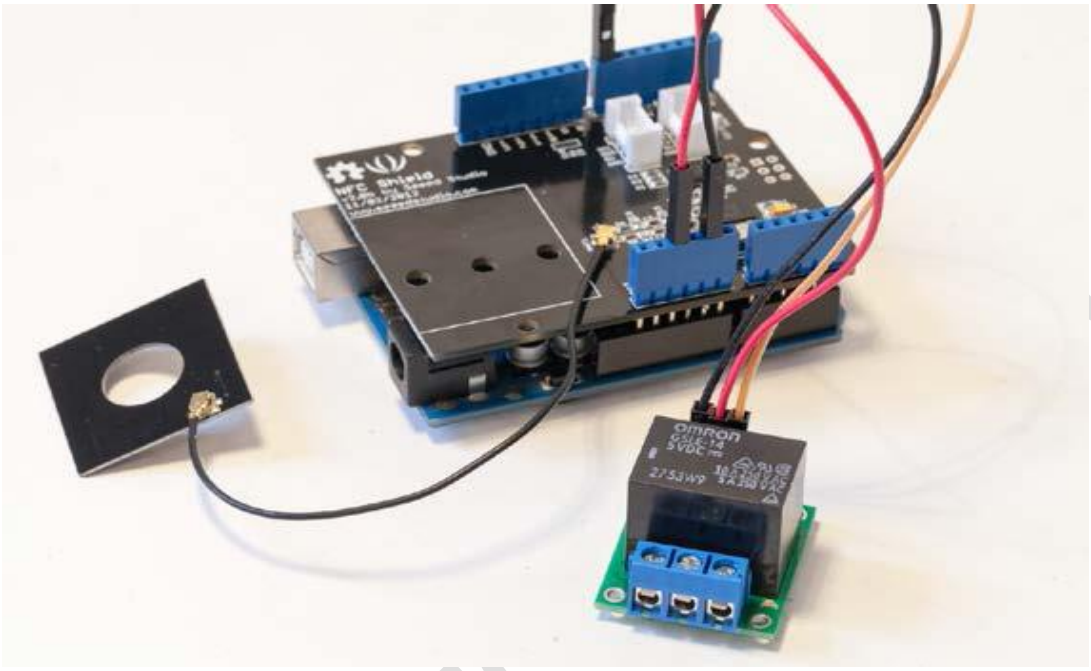
- Download the PN532 library from https://github.com/Seeed-Studio/PN532 and put all folders into your Arduino's libraries folder
- Download the NDEF(NFC Data Exchange Format) library from https://github.com/don/NDEF and put it in your Arduino's libraries folder and rename the folder to NDEF

Configuring the Hardware

1. Put the NFC shield on top of the Arduino Uno board, and connect the NFC reader to the shield. (If the header is not soldered then we need to solder the headers)

2. To assemble the NFC reader to the shield, simply connect the reader via the antenna connector on the shield
3. Connect the relay module VCC pin to the 5V pin of the Arduino board, and the GND pin to the GND pin of the board. Connect the SIG pin of the relay to the pin number 8 of the Arduino board

The following figure shows the Hardware configuration



Testing the NFC shield

Arduino code to test the functionality of the shield:

```
#include <SPI.h>
#include <PN532_SPI.h>  // SPI- Serial Peripheral Interface
#include <PN532.h>
#include <NfcAdapter.h>
// NFC instances
PN532_SPI pn532spi(SPI, 10);
NfcAdapter nfc = NfcAdapter(pn532spi);
void setup(void) {
// Start Serial
Serial.begin(9600);
// Start NFC chip
Serial.println("NFC shield started");
nfc.begin();
}
void loop(void) {
// Start scan
Serial.println("\nScan a NFC tag\n");
if (nfc.tagPresent())
{
NfcTag tag = nfc.read();
tag.print();
}
```

```
delay(5000);
}
```

Complete Arduino Sketch for controlling an Arduino Board via NFC

```
// Include libraries
#include "SPI.h"
#include "PN532_SPI.h"
#include "snep.h"
#include "NdefMessage.h"

// Relay pin
#define RELAY_PIN 8

// Code for the On states
#define RELAY_ON "oWnHV6uXre"
// Create NFC object instance
PN532_SPI pn532spi(SPI, 10);
SNEP nfc(pn532spi);

// NFC buffer
uint8_t ndefBuf[128];

void setup() {

   // Start Serial communications
   Serial.begin(9600);
   Serial.println("NFC Peer to Peer Light Switch");

   // Declare relay pin as output
   pinMode(RELAY_PIN, OUTPUT);
}
void loop(void) {

  // Wait for NFC message
  Serial.println("Waiting for message from Peer");
  int msgSize = nfc.read(ndefBuf, sizeof(ndefBuf));
  if (msgSize > 0) {

   // Read message
   NdefMessage message  = NdefMessage(ndefBuf, msgSize);

   // Make sure there is at least one NDEF Record
   if (message.getRecordCount() > 0) {

     NdefRecord record = message.getRecord(0);
     Serial.println("Got first record");
// Check the TNF and Record Type
     if (record.getTnf() == TNF_MIME_MEDIA && record.getType() ==
"application/com.arduinoandroid.arduinonfc") {
       Serial.println("Type is OK");
```

```
      // Get the bytes from the payload
      int payloadLength = record.getPayloadLength();
      byte payload[payloadLength];
      record.getPayload(payload);
// Convert the payload to a String
      String payloadAsString = "";
      for (int c = 0; c < payloadLength; c++) {
        payloadAsString += (char)payload[c];
      }

      // Print out the data on the Serial monitor
      Serial.print("Payload is ");Serial.println(payloadAsString);
// Modify the state of the light, based on the tag contents
      if (payloadAsString == RELAY_ON) {
         digitalWrite(RELAY_PIN, HIGH);
      } else {
         digitalWrite(RELAY_PIN, LOW);
      }
    } else {
Serial.print("Expecting TNF 'Mime Media' (0x02) with type
'application/com.arduinoandroid.arduinonfc' but found TNF ");
      Serial.print(record.getTnf(), HEX);
      Serial.print(" type ");
      Serial.println(record.getType());
    }
  }
 }
}
```

Setting up the Android app

In this project, we will be implementing an Android app that leverages the use of the NFC API and hardware allowing us to send a MIME-type message to switch on and switch off the relay.

Create a New Project as

- Name: Arduino NFC
- Minimum SDK: 18
- Project: Empty Activity
- Activity Name: MainScreen
- Domain: your choice

Add the following two lines of code to AndroidManifest.xml

```
<uses-permission android:name="android.permission.NFC" />
<uses-feature android:name="android.hardware.nfc" android:required="true" />
```

Design the following screen with two TextView and two Button components

Code for NFCActivity.java

```
package com.arduinoandroid.arduinonfc;

import android.content.Intent;
import android.nfc.NdefMessage;
import android.nfc.NdefRecord;

public class NFCActivity extends Activity {

    //Declaring the User Interface Variables for mStatusText as a TextView
    private TextView mStatusText;
    private TextView messageToBeam;
    private Button switchOn;
    private Button switchOff;
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_nfc);

        mStatusText = (TextView) findViewById(R.id.nfcTextStatus);
        messageToBeam = (TextView) findViewById(R.id.messageToBeam);
        switchOn = (Button) findViewById(R.id.switchOnBtn);
```

```
        switchOff = (Button) findViewById(R.id.switchOffBtn);
// Adding OnClick Listeners to the Buttons
        switchOn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                messageToBeam.setText(open_key);
            }
        });
switchOff.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                messageToBeam.setText(close_key);
            }
        });
// Check for available NFC Adapter
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        if (mNfcAdapter == null) {
            mStatusText.setText("NFC is not available on this device.");
        }

        // Register to create and NDEF(NFC Data Exchange Format) message when another
device is in range
        mNfcAdapter.setNdefPushMessageCallback(new
NfcAdapter.CreateNdefMessageCallback() this);
// And handle the send status
        mNfcAdapter.setOnNdefPushCompleteCallback(new
NfcAdapter.OnNdefPushCompleteCallback() this);
    }

    @Override
    public void onNewIntent(Intent intent) {
        // handle singleTop so we don't launch a number of instances...
        setIntent(intent);
    }
}
```