IOT PROJECT - 1

NFC Payment and Security

Team Members:
Jaysheel Shah – 131019
Mohana Saraf – 131028
Netra Pathak – 131029
Sharvil Shah - 131050

Definition

NFC (near-field communication) allows two devices placed within a few centimeters of each other to share small bits of data. In order for this to work, both devices must be equipped with an NFC chip. It is a secured and short distance communication procedure and works in the range of 10 cm. NFC utilizes radio frequency of 13.56 MHz with a transmitting range of1Mbps.

Today NFC has been used in multiple range of applications like smart payment system, access control, ticketing at various widely used sectors, etc. The reason for this is that, with advancement in technology, we are able to provide better security. Furthermore, customer convenience also improves because all the data for multiple cards can be stored and handled through one application, in NFC enabled phones.

Once the application is launched in the phone, the phone is tapped on the NFC reader/writer terminal and a connection is made using NFC. At this point, the user is asked to scan either their finger print or enter a passcode to approve the transaction. The transaction is then carried out based on the authentication details. From there, the payment finishes processing the same way it would in a traditional credit card swipe transaction.

There are two ways this process can be carried out –

Two-way communication: This involves two devices that can both read and write to each other. For example, using NFC, you can touch two Android devices together to transfer data like contacts, links, or photos.

One-way communication: Here, a powered device (like a phone, credit card reader, or commuter card terminal) reads and writes to an NFC chip. So, when you tap your commuter card on the NFC terminal, the NFC-powered terminal subtracts money from the balance specified by the user.

Android Application

The android application we aim to develop will be used to send login ID and password to the NFC reader, through the NFC tag. The application uses the built-in NFC tag to send and receive the data. Once the reader authenticates the account details, it sends a signal to the application to proceed for payment. User then enters the payable amount in the text field displayed on the application. Next, the NFC reader reads the details from the application. If the password is incorrect application, it will ask the user to re-enter all the details. For maintaining security, only three attempts will be provided to the user. Failure to all of these will lead to blocking the card for the next 48 hours.

We will be developing the application of MIT App Inventor:

- **Version**: NV150E
- **Use Companion**: 2.38

We will be using this application on Android version 6.0.1

Input will always be either the information that the user enters or the flag value read from the NFC reader. The information received from the user will be sent to the NFC reader, as the output from the application.

Interfacing for the application is done using the inbuilt NFC of an android phone.

Problems Faced

The initial android application, explained as above, was designed, formulated and framed such that

- The developed android application is used to send login ID and password to the NFC reader, through the NFC tag and,
- User enters payable amount in the application.

But, following were the problems that we faced while proceeding further and at each stage how we solved the problem to move further:

- First, we had devised the application to accept login ID and password separately, in two different text fields, with the password text field having hidden characters (for security). The first challenge was to accept two NDEF. After rigorous trials, we ultimately found out that it was not feasible to accept two different messages. For this, we came up with the solution that: we internally combined both the text fields with an identifier '_' and then sent one NDEF message only of this combined string.

- After the implementation of being able to read the login ID and password, we faced another challenge. We were not able to send data to the application (write on the NFC reader). We executed the default library code on some other available android applications in the android market also but it was not working. We searched for several other libraries and tried implementing but eventually figured out that it was unfavourable with the NFC reader we were using. This would not help us if we would want to send any confirmation or 'insufficient funds' message to the application, for the user. For this reason, we decided to enter only the login ID and password through the application. Because of being unable to send two NDEF message from the application and receive message from the NFC reader, we introduced a 4*4 keypad and a LCD in our system. They keypad would enable us to enter the payable amount and this entered amount would be simultaneously displayed on the LCD. Also, on the LCD we would be able to display messages like 'verification successful' (when login ID and password, sent over internet, match from the database; 'insufficient funds' (when balance is less than the payable amount entered), etc.

- Next, after the addition of two actuators (keypad and LCD), with one sensor (NFC reader); we were left with very less pins on Arduino UNO board. This would not facilitate the internet connection of the standalone system with the server because necessary pins for connection were already occupied. The solution to this problem was to use an Arduino MEGA 2560 which had multiple TX and RX, unlike Arduino UNO board.

- The last challenge/problem faced while implementing NFC payment and security was that NFC reader and Ethernet Shield both used SPI protocol. The pins for SPI protocol on Arduino were limited and were used by NFC reader. Due to this, we could not connect the Ethernet shield. Hence, we decided to use the WIFI module ESP8266.

- ESP8266 is very rarely used and thus there were no proper documentation for its connection with the Arduino boards. Thus, we tried various different commands for its connections. While working and reading various material online we realized WIFI module works on 3.3V and if we input 3.3V from the Arduino board directly the WIFI module may work inconsistently. Therefore, we introduced the Voltage regulator LM1117T which converts the 5V from the Arduino to 3.3V, which can be now given to the WIFI module.

- Overcoming all challenges and problems at each stage the only drawback that our project has is while NFC touch to beam is used we need to keep the phone and the reader at the precise location for transfer to take place successfully.

Initial Block Diagram

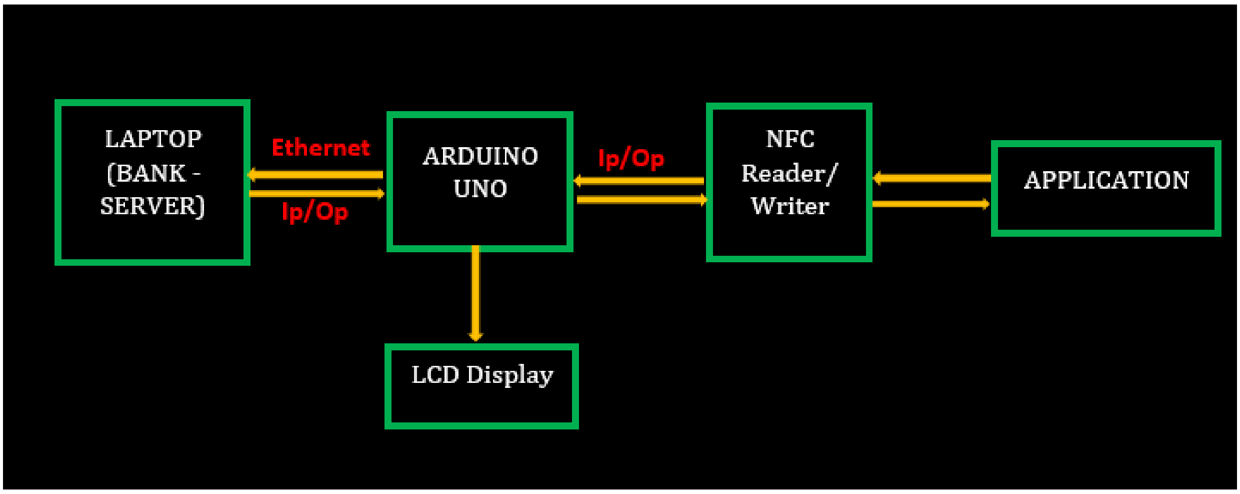


Figure 1: Block Diagram – General Flow of the NFC payment application

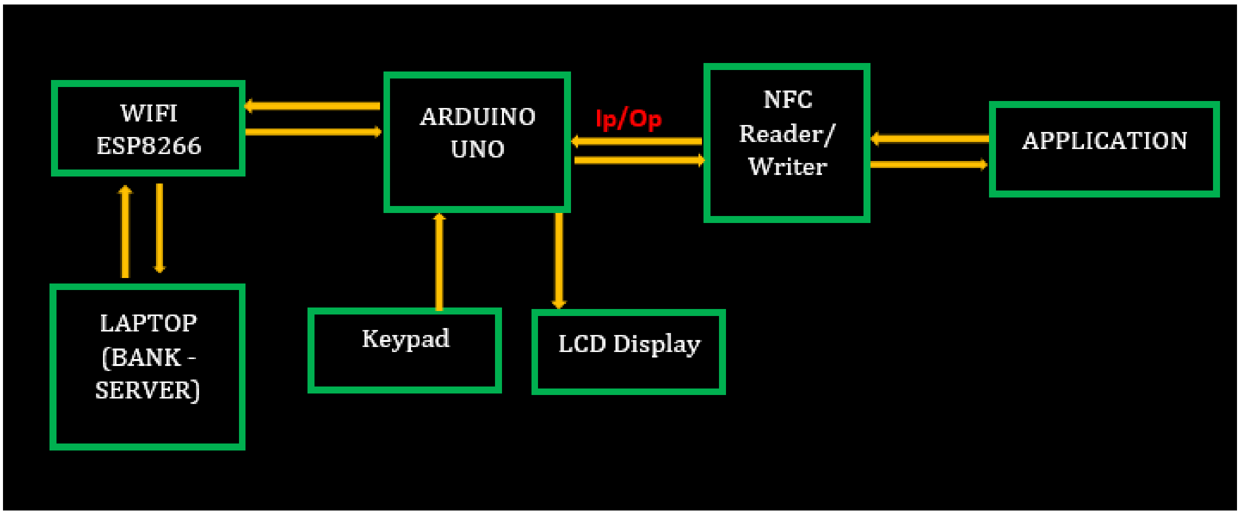


Figure 2: Updated Block Diagram – General Flow of the NFC payment application with keypad and WIFI module

List of Components – (initial definition)

- Arduino Uno
- NFC reader PN532
- LCD Display
- Ethernet Cable

Final List of Components –

- Arduino MEGA 2560
- NFC reader PN532
- LCD Display
- Potentiometer
- 4*4 Keypad
- WIFI Module ESP8266
- Voltage Regulator

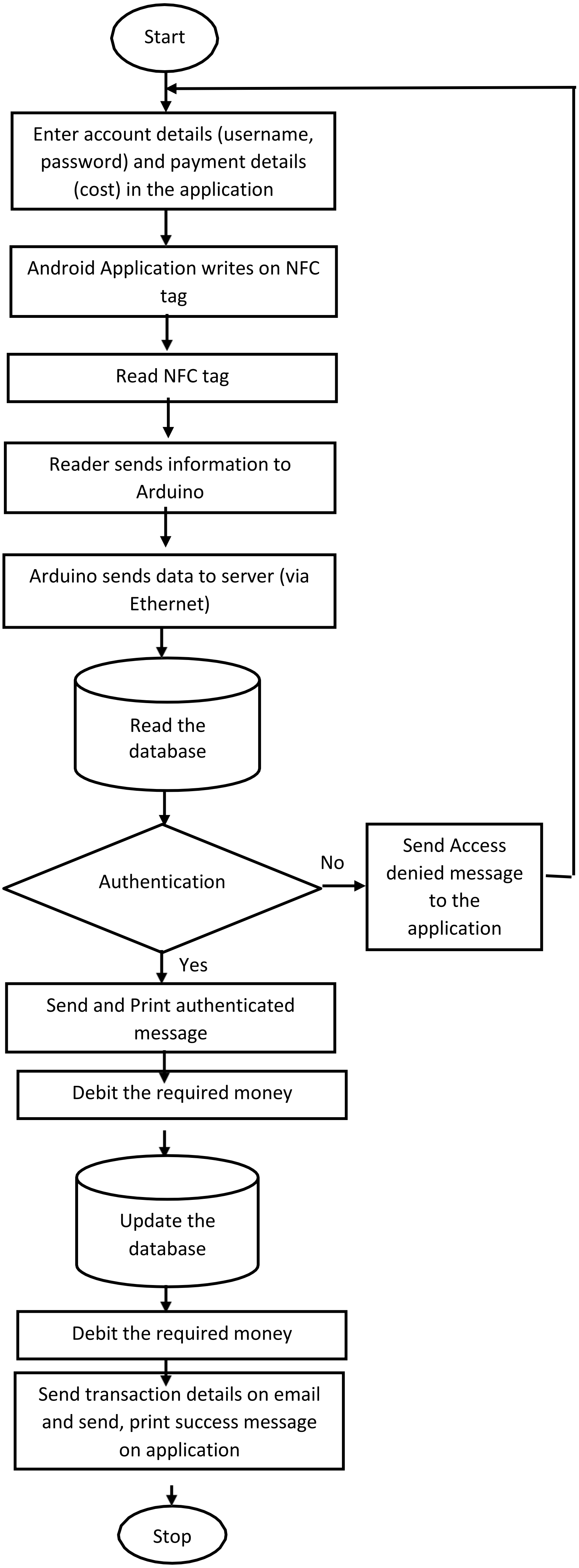
Selection Criteria for each Component

Table 1: Specification of Components used

	Sensor 1: NFC Tag Reader	**Actuator 1 : LCD Display**	**Actuator 2: Keypad**
Company	Robodo	Issued by college	Issued by college
Price	1099 Rs	NIL	NIL
Measurement Range	5cm - 7cm	-	-
Interface Type	Digital	Digital	Digital
Power Supply	5V	5V	5V
Principle operation	Read/Write NFC Tag	Displays Information sent by Arduino	Take numbers as Input
Special Note		16 X 2 Display	4 X 4 Keypad

Figure 2:

Flow Diagram


Start
Enter account details (username, password) and payment details (cost) in the application
Android Application writes on NFC tag
Read NFC tag
Reader sends information to Arduino
Arduino sends data to server (via Ethernet)
Read the database
Authentication
No
Send Access denied message to the application
Yes
Send and Print authenticated message
Debit the required money
Update the database
Debit the required money
Send transaction details on email and send, print success message on application
Stop

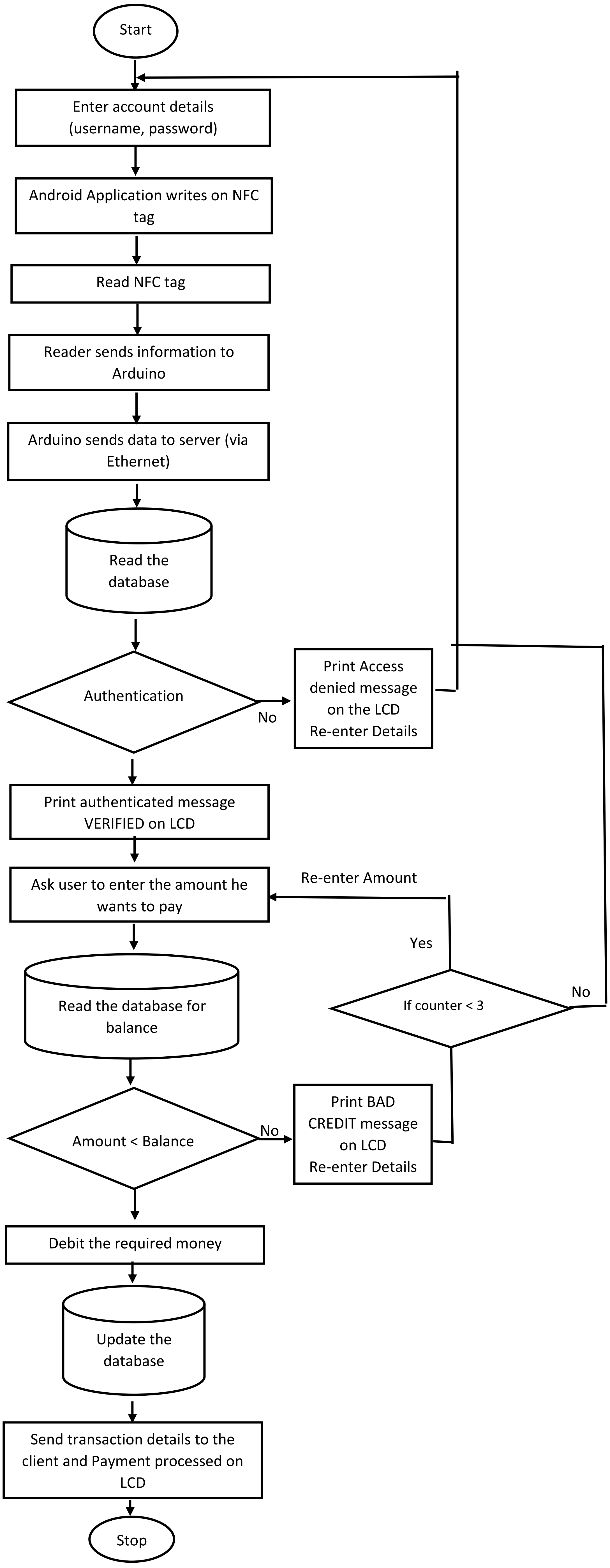

Start
Enter account details (username, password)
Android Application writes on NFC tag
Read NFC tag
Reader sends information to Arduino
Arduino sends data to server (via Ethernet)
Read the database
Authentication
No
Print Access denied message on the LCD Re-enter Details
Print authenticated message VERIFIED on LCD
Ask user to enter the amount he wants to pay
Re-enter Amount
Yes
If counter < 3
No
Read the database for balance
Amount < Balance
No
Print BAD CREDIT message on LCD Re-enter Details
Debit the required money
Update the database
Send transaction details to the client and Payment processed on LCD
Stop

Figure 3: Updated Flow of the NFC payment

ANDROID APPLICATION FLOW CHART (initial idea)

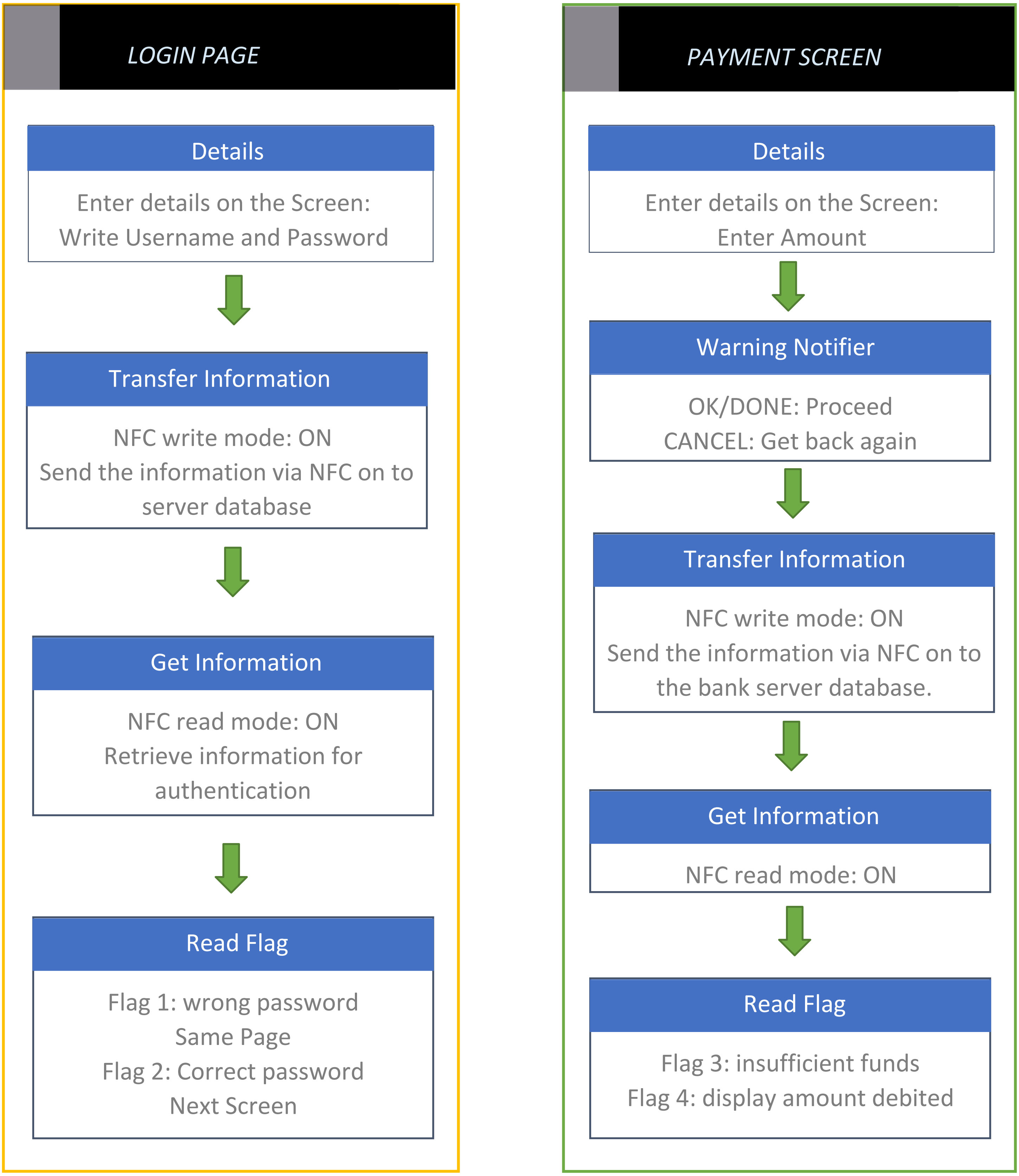


Figure 4: Flow Diagram of the Android Application

Working Principle

Sensor: Near Field Communication Reader/Writer PN532

The PN532 is a highly integrated transmission module for contactless communication at 13.56 MHz including microcontroller functionality based on an 80C51 core with 40 Kbytes of ROM and 1 Kbytes of RAM.

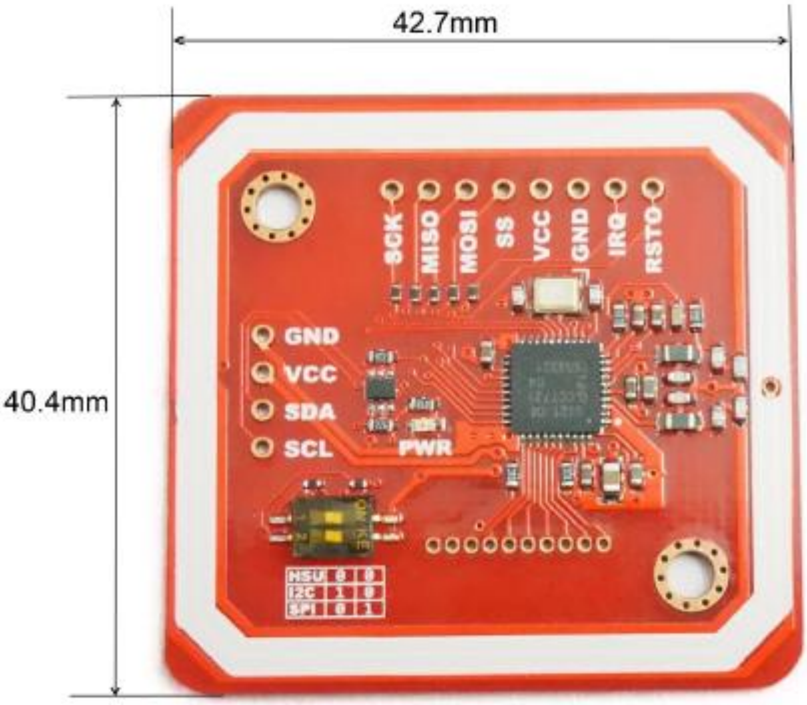


Figure 5: PN532 NFC Chip

The embedded firmware and the internal hardware support the handling of the host controller protocol for the different interfaces –

- I2C,
- SPI
- Serial High Speed UART (HSU).

Table 2: Selecting between different modes in PN532

Modes	Switch 1	Switch 2
HSU	0	0
I2C	1	0
SPI	0	1

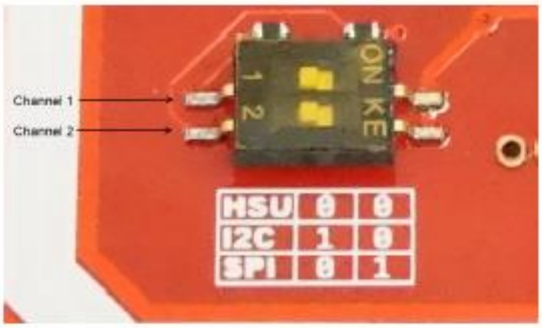


Figure 6: Toggle Switch to change the Modes

Concept of Near Field Communication:

NFC works on the principle of sending information over radio waves. Near Field Communication is another standard for wireless data transitions, meaning that there are specifications which devices must adhere to in order to communicate with each other properly. The technology that is used in NFC is based on older RFID (Radio-frequency identification) idea, which uses electromagnetic induction in order to transmit information.

As NFC enabled devices induces current within the other passive components, thus such passive devices do not need their own power supply, and instead are powered by the electromagnetic field produced by an active NFC component when it comes into range.

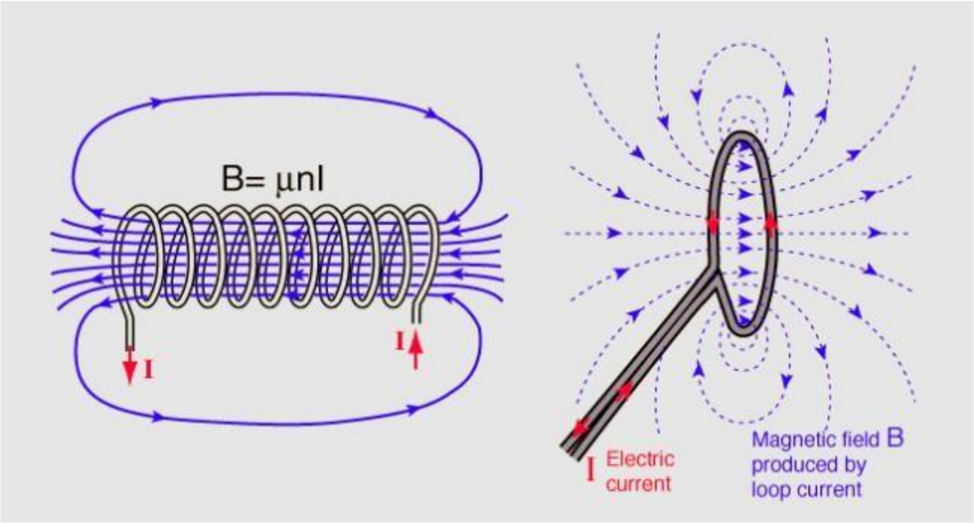


Figure 7: Field produced by active devices

Actuator: Liquid Crystal Display (LCD)

An LCD screen uses the polarizing sunglasses trick to switch its coloured pixels on or off.

Concept of polarized light:

When sunlight streams come down from the sky, the light waves are all mixed up and is vibrating in all possible direction. But if a filter is put in its way, with a row of lines vertically arranged, all light waves except the ones vibrating vertically are blocked. Thus our filter effectively dims the light as we block off much of the original sunlight.

This is how polarizing sunglasses work: they cut out all but the sunlight vibrating in one direction or plane. Light filtered in this way is called **polarized** or **plane-polarized** light (because it can travel in only one plane).

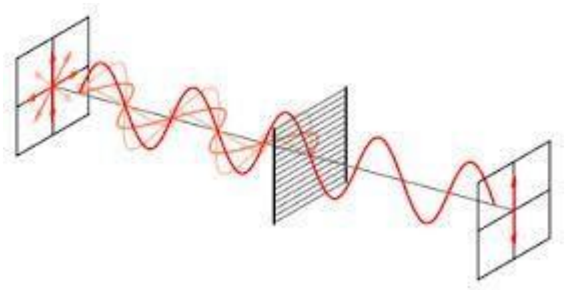


Figure 8: Shows blocking of light

LCD uses liquid crystals and polarized light:

Now, similarly in LCD display at the back of the screen, there's a large bright light that shines out toward the viewer. In front of this light, there are the millions of pixels, each pixel is made up of smaller areas called sub-pixels. Now, each such sub-pixels are coloured red, blue, or green. Behind each pixel there is a polarizing glass filter and another one in front of it at 90 degrees. That means the pixel normally looks dark.

A tiny twisted, nematic liquid crystal that can be switched on or off electronically, is present between the two glasses. It rotates the light passing through it at 90 degrees, when it’s switched off, effectively allowing light to flow through the two polarizing filters and making the pixel look bright. When it's switched on, it doesn't rotate the light, which is blocked by one of the polarizers, and the pixel looks dark. Each pixel is controlled by a separate transistor that can switch it on or off many times each second.

Showing how pixels are switched off.

1. From a large bright light, light travels from the back of the TV toward the front.
2. The front horizontal polarizing filter blocks all light except the ones vibrating horizontally.
3. Thus, only the light waves vibrating horizontally can get through.

4. A transistor switches off this pixel by switching *on* the electricity flowing through its liquid crystal which makes the crystal straighten out (so it's completely untwisted), and the light travels straight through it unchanged.

5. Now, the light waves emerging from this liquid crystal is still vibrating horizontally.

6. The front vertical polarizing filter of the liquid crystal blocks all light except the ones vibrating vertically. The horizontally vibrating light that travelled through the liquid crystal cannot get through the vertical filter.

7. No light reaches the screen at this point. In other words, this pixel is dark.

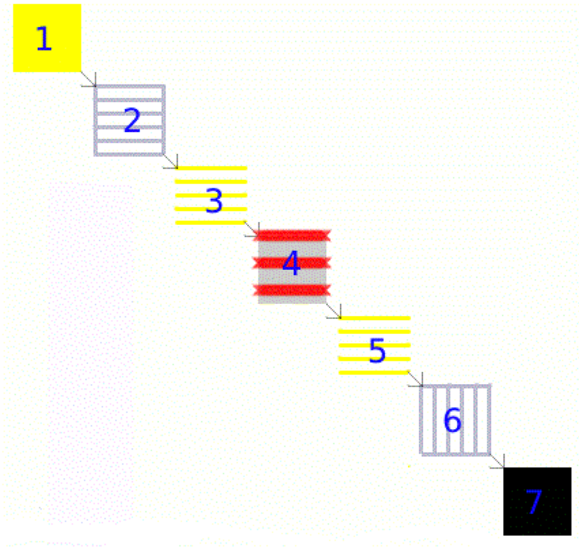


Figure 9: LCD pixels switched off

Showing how pixels are switched on.

1. The bright light at the back of the screen shines as before.

2. The front horizontal polarizing filter blocks all light except the ones vibrating horizontally.

3. Thus, only the light waves vibrating horizontally can get through.

4. A transistor switches on this pixel by switching *off* the electricity flowing through its liquid crystal. That makes the crystal twist. The twisted crystal rotates light waves by 90° as they travel through it.

5. Light waves that enters the liquid crystal vibrating horizontally emerge from it vibrating vertically.

6. The front vertical polarizing filter of the liquid crystal blocks all light except the ones vibrating vertically. The vertically vibrating light that emerged from the liquid crystal can now get through the vertical filter.

7. The pixel is lit up. A red, blue, or green filter gives the pixel its color.

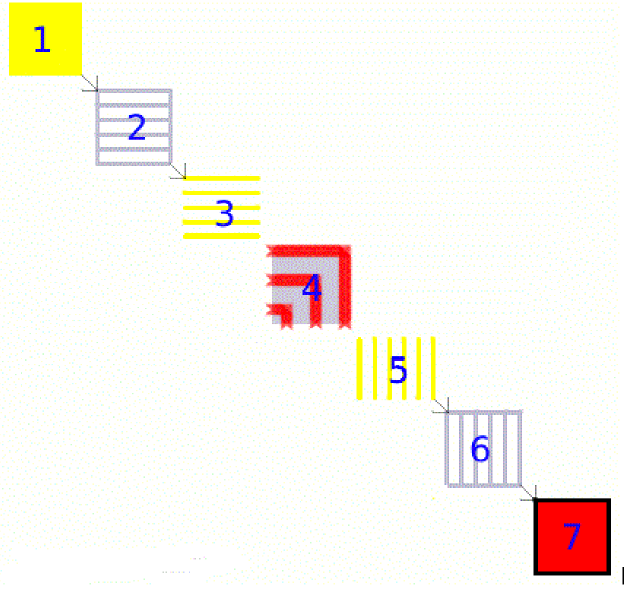


Figure 10: LCD pixels switched on

Actuator: 4 X 4 Keypad

Keypad 4x4 is used for taking numerical input to the microcontroller. It is arranged in a form of an array containing four lines and four columns, thus total 16 buttons.

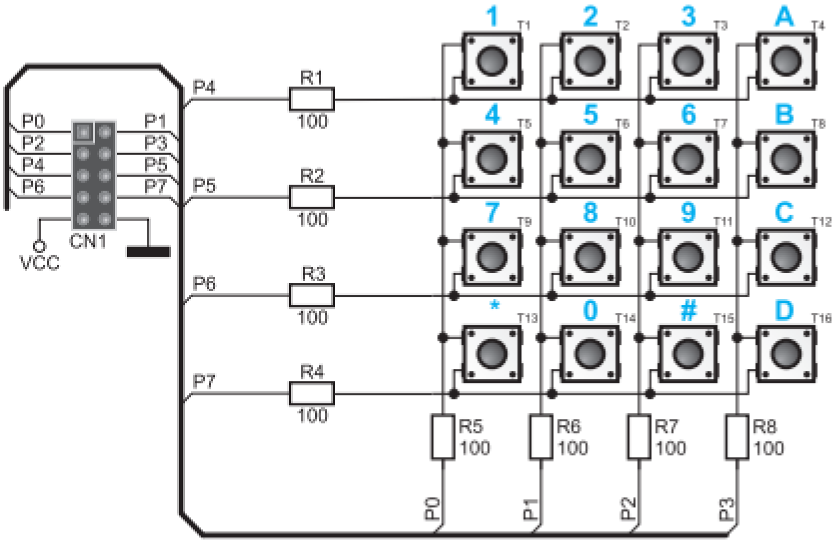


Figure 11: Keypad Pins and Connections

Matrix Keypad Interface Logic

Initially there is no connection between the rows and columns as all switches are assumed to be released. When any one of the switches is pressed the corresponding rows and columns are connected (short circuited). A press of any switch drives the respective column pin (initially high) low.

Initially logic 0 is given to the rows and logic 1 to the columns. When any key is pressed, the columns are scanned first and if it detects a logic 0, then the program knows a key press was made in that particular column. This is achieved because when the switch in nth column is pressed it shorts the Cn line with Rn. Hence, Cn is driven low.

Now, once the column is detected in which the key was pressed, we sequentially write logic 1 to each row and check for the Cn column becomes high again. The logic is that if the switch in a row Rn was pressed then it will reflect logic 1 in column Cn because they are short circuited.

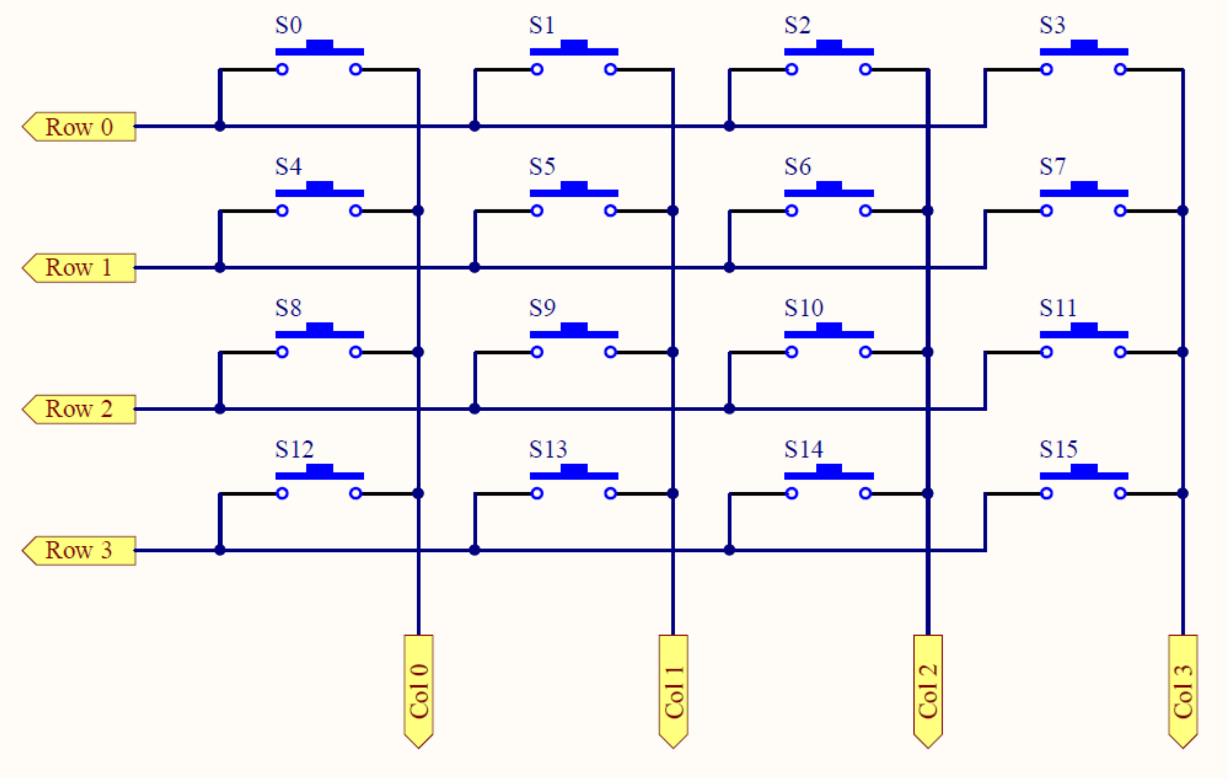


Figure 12: Shows Internal Connections of PINS

<u>CODE for Interfacing</u> (initial code)

This code is the initial code that we used for receiving message from the application and then displaying on the serial monitor. For the stand alone prototype.

```
#include <SPI.h>
#include <PN532.h>
#include <NFCLinkLayer.h>
#include <SNEP.h>
#include <avr/power.h>
#include <avr/sleep.h>

#include <NdefMessage.h>

#define SS 10

PN532 nfc(SS);


NFCLinkLayer linkLayer(&nfc);
SNEP snep(&linkLayer);

// This message shall be used to rx or tx
// NDEF messages it shall never be released
#define MAX_PKT_HEADER_SIZE 50
#define MAX_PKT_PAYLOAD_SIZE 100

void phoneInRange()
{
  //sleep_disable(); // Prevents the arduino from going to sleep if it was about too.
}


void setup(void) {
    Serial.begin(960
    0);
    Serial.println(F("----------------- NFC pull NDEF message --------------------"));

nfc.initializeReader();
```

```
uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
  }
  // Got ok data, print it out!
  Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);
  Serial.print("Supports "); Serial.println(versiondata & 0xFF, HEX);

#ifdef ENABLE_SLEEP
  // set power sleep mode
  set_sleep_mode(SLEEP_MODE_ADC);
  // interrupt to wake MCU
  attachInterrupt(0, phoneInRange, FALLING);
#endif

  nfc.SAMConfig();
}

uint8_t buf[180];

void loop(void)
{
  Serial.println();
  Serial.println(F("---------------- LOOP ----------------------"));
  Serial.println();
  uint8_t rxNDEFMessage[MAX_PKT_HEADER_SIZE + MAX_PKT_PAYLOAD_SIZE];
  uint8_t *rxNDEFMessagePtr;
  uint32_t rxResult = GEN_ERROR;
  rxNDEFMessagePtr = &rxNDEFMessage[0];

#ifdef ENABLE_SLEEP
   if (IS_ERROR(nfc.configurePeerAsTarget(SNEP_SERVER))) {
    sleepMCU();

    extern uint8_t pn532_packetbuffer[];
    nfc.readspicommand(PN532_TGINITASTARGET, (PN532_CMD_RESPONSE
*)pn532_packetbuffer);
   }
#else
   if (IS_ERROR(nfc.configurePeerAsTarget(SNEP_SERVER))) {
    extern uint8_t pn532_packetbuffer[];

    while (!nfc.isReady()) {
    }
    nfc.readspicommand(PN532_TGINITASTARGET, (PN532_CMD_RESPONSE
*)pn532_packetbuffer);
    }
#endif
```

```
do {
       rxResult = snep.rxNDEFPayload(rxNDEFMessagePtr);

       if (rxResult == SEND_COMMAND_RX_TIMEOUT_ERROR)
       {
         Serial.println("rxNDEFPayload() timeout");
          break;
       } else if (IS_ERROR(rxResult)) {
         Serial.println("rxNDEFPlayload() failed");
         break;
       }


       if (RESULT_OK(rxResult))
       {
          NdefMessage *message = new NdefMessage(rxNDEFMessagePtr, rxResult);
          Serial.print("NDEF record: ");
          Serial.println(message->getRecordCount());
          NdefRecord record = message->getRecord(0);
          record.print();
          //uint8_t *pl = record.getPayload();
          //Serial.print("asasas:");Serial.println((char)pl[3]);
          delay(3000);
       }
    } while(0);
}

void sleepMCU()
{
   delay(100); // delay so that debug message can be printed before the MCU goes to
sleep

   sleep_enable();          // Enable sleep mode
   power_adc_disable();
   power_spi_disable();
   power_timer0_disable();
   power_timer1_disable();
   power_timer2_disable();
   power_twi_disable();

   //Serial.println("Going to Sleep\n");
   //delay(1000);

   // Puts the device to sleep.
   sleep_mode();
   Serial.println("Woke up");

   // Program continues execution HERE
   // when an interrupt is recieved.
   // Disable sleep mode
   sleep_disable();

   power_all_enable();
}
```

Final Code for Interfacing the Project

```
#include <SPI.h>
#include <PN532.h>
#include <NFCLinkLayer.h>
#include <SNEP.h>
#include <avr/power.h>
#include <avr/sleep.h>

#include <NdefMessage.h>
#define DEBUG true
#define SS 10

PN532 nfc(SS);
NFCLinkLayer linkLayer(&nfc);
SNEP snep(&linkLayer);

uint32_t createNDEFShortRecord(uint8_t *message, uint8_t payloadLen, uint8_t *&NDEFMessage);

// This message shall be used to rx or tx
// NDEF messages it shall never be released
#define MAX_PKT_HEADER_SIZE  50
#define MAX_PKT_PAYLOAD_SIZE 100
uint8_t rxNDEFMessage[MAX_PKT_HEADER_SIZE + MAX_PKT_PAYLOAD_SIZE];
uint8_t txNDEFMessage[MAX_PKT_HEADER_SIZE + MAX_PKT_PAYLOAD_SIZE];
uint8_t *txNDEFMessagePtr;
uint8_t *rxNDEFMessagePtr;
uint8_t txLen;

uint8_t flag=0;
uint8_t *msg;

#define SHORT_RECORD_TYPE_LEN   0x0A
#define NDEF_SHORT_RECORD_MESSAGE_HDR_LEN   0x03 + SHORT_RECORD_TYPE_LEN
#define TYPE_STR "text/plain"

void phoneInRange()
{
  //sleep_disable(); // Prevents the arduino from going to sleep if it was about too.
}
void(* resetFunc) (void) = 0;
/***********************KEYPAD******************************** /

#include <Keypad.h>

const byte numRows= 4; //number of rows on the keypad
const byte numCols= 4; //number of columns on the keypad

//keymap defines the key pressed according to the row and columns just as appears on the keypad
```

```
char keymap[numRows][numCols]=
{
{'1', '2', '3', 'A'},
{'4', '5', '6', 'B'},
{'7', '8', '9', 'C'},
{'*', '0', '#', 'D'}
};

String Users[4]={"Moh_123","Net_456","Sha_789","Jay_555"};
int bal[4]={50,2000,667,1000};
//Code that shows the the keypad connections to the arduino terminals
byte colPins[numRows] = {A1,2,A2,4}; //Rows 0 to 3
byte rowPins[numCols]= {A4,7,A5,8}; //Columns 0 to 3

//initializes an instance of the Keypad class
Keypad myKeypad= Keypad(makeKeymap(keymap), rowPins, colPins, numRows, numCols);


String keypad_input="";
int flag_break = 1;

/*******************************LCD*******************************/

#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(A0, 3, A3, 5, 6, 9);


void setup(void) {
  Serial.begin(115200);
  Serial1.begin(115200);
  lcd.begin(16, 2);
  Serial.println("----------------- nfc ndef demo --------------------");


    sendData("AT+RST\r\n",2000,DEBUG); // reset module
    sendData("AT+CWMODE=2\r\n",1000,DEBUG); // configure as access point
   sendData("AT+CIFSR\r\n",1000,DEBUG); // get ip address
   sendData("AT+CIPMUX=1\r\n",1000,DEBUG); // configure for multiple connections
  sendData("AT+CIPSERVER=1,80\r\n",1000,DEBUG); // turn on server on port 80



  uint8_t message[] = "1";
  txNDEFMessagePtr = &txNDEFMessage[MAX_PKT_HEADER_SIZE];
  rxNDEFMessagePtr = &rxNDEFMessage[0];
  txLen = createNDEFShortRecord(message, sizeof(message), txNDEFMessagePtr);

  if (!txLen)
  {
    Serial.println("Failed to create NDEF Message.");
    while(true); //halt
  }
```

```
   nfc.initializeReader();

   uint32_t versiondata = nfc.getFirmwareVersion();
   if (! versiondata) {
      Serial.print("Didn't find PN53x board");
      while (1); // halt
   }
   // Got ok data, print it out!
   Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
   Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
   Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);
   Serial.print("Supports "); Serial.println(versiondata & 0xFF, HEX);

#ifdef ENABLE_SLEEP
   // set power sleep mode
   set_sleep_mode(SLEEP_MODE_ADC);
   // interrupt to wake MCU
   attachInterrupt(0, phoneInRange, FALLING);
#endif

   nfc.SAMConfig();
}

uint8_t buf[180];

void loop(void)
{
  Serial.println();
  Serial.println(F("---------------- LOOP ----------------------"));
  Serial.println();

   uint32_t rxResult = GEN_ERROR;
   uint32_t txResult = GEN_ERROR;
   rxNDEFMessagePtr = &rxNDEFMessage[0];


#ifdef ENABLE_SLEEP
    if (IS_ERROR(nfc.configurePeerAsTarget(SNEP_SERVER))) {
      sleepMCU();

      extern uint8_t pn532_packetbuffer[];
      nfc.readspicommand(PN532_TGINITASTARGET, (PN532_CMD_RESPONSE
*)pn532_packetbuffer);
    }

#else
    if (IS_ERROR(nfc.configurePeerAsTarget(SNEP_SERVER))) {
      extern uint8_t pn532_packetbuffer[];

      while (!nfc.isReady()) {
      }
      nfc.readspicommand(PN532_TGINITASTARGET, (PN532_CMD_RESPONSE
*)pn532_packetbuffer);
     }
#endif
```

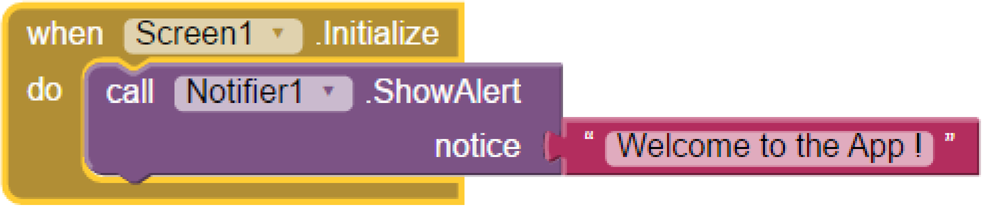
```
do {
    //Serial.println("Begin Rx Loop");
    rxResult = snep.rxNDEFPayload(rxNDEFMessagePtr);

    if (rxResult == SEND_COMMAND_RX_TIMEOUT_ERROR)
    {
     Serial.println("rxNDEFPayload() timeout");
      break;
    } else if (IS_ERROR(rxResult)) {
     Serial.println("rxNDEFPlayload() failed");
     break;
    }


    if (RESULT_OK(rxResult) && flag==0)
    {
      NdefMessage *message = new NdefMessage(rxNDEFMessagePtr, rxResult);
      Serial.print("NDEF record: ");
      Serial.println(message->getRecordCount());
      NdefRecord record = message->getRecord(0);
      Serial.println(" ");
      msg=record.getPayload();
      //Serial.println((char*)msg);
      if((char)(msg[1]=='e') && ((char)msg[2]=='n') )
      {
        //int tflag=0;
        int cmpr=0;
        int q=0;
        int l=0;
        for(q=0;q<4;q++)
        {
         for(l=0;l<7;l++)
         {
           if(msg[l+3]!=Users[q].charAt(l))
           {
             break;
           }
         }
         if(l==7)
         {
           cmpr=q;
           break;
         }
        }
        if(q==4)
        {
         Serial.println((char*)msg);
         resetFunc();
        }
```

```
Serial.println((char*)msg);
        Serial.print("User=");
        Serial.println(q);
        //txResult = snep.pushPayload(txNDEFMessagePtr, txLen);
        flag=1;
        //from internet database verify pin, yet to code
        lcd.setCursor(0, 0);
        lcd.print("Verified");
        //LCD display remaining - verified:
        delay(10000);
        lcd.setCursor(0, 0);
        lcd.print("Amount entered:");
         /******************Keypad input*************/
         String bal_in="";
         while(flag_break){
         char keypressed = myKeypad.getKey();
         lcd.setCursor(0,1);
          if (keypressed != NO_KEY)
          {
          Serial.print(keypressed);

          if(keypressed == '*')
          {
           lcd.setCursor(0, 1);
           if(bal[q]<bal_in.toInt())
           {
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Bad credit");
            lcd.setCursor(0, 1);
            lcd.print("Resetting");
            delay(5000);
            resetFunc();
           }
           lcd.print("Payment processed");

           while(true)
           {
              if(Serial1.available()) // check if the esp is sending a message
              {

               if(Serial1.find("+IPD,"))
               {
                delay(1000);

                int connectionId = Serial1.read()-48; // subtract 48 because the read() function
returns
                                                   // the ASCII decimal value and 0 (the first decimal number)
starts at 48
String webpage = "Successful Transaction of ";
            webpage+=bal_in;
            webpage+=" rupees";
            webpage+=" Curr bal=";
            webpage+=String(bal[q]-bal_in.toInt());
            //webpage+="</h2>";
            String cipSend = "AT+CIPSEND=";
            cipSend += connectionId;
            cipSend += ",";
            cipSend +=webpage.length();
            cipSend +="\r\n";

            sendData(cipSend,1000,DEBUG);
            sendData(webpage,1000,DEBUG);

            /*webpage="<button>LED2</button>";

            cipSend = "AT+CIPSEND=";
            cipSend += connectionId;
            cipSend += ",";
            cipSend +=webpage.length();
            cipSend +="\r\n";

            sendData(cipSend,1000,DEBUG);
            sendData(webpage,1000,DEBUG);*/

            String closeCommand = "AT+CIPCLOSE=";
            closeCommand+=connectionId; // append connection id
            closeCommand+="\r\n";

            sendData(closeCommand,3000,DEBUG);
           }
          }
         }


flag_break = 0;
          break;
         }
         bal_in+=keypressed;
         keypad_input+=keypressed;
         }
         lcd.print(keypad_input);
        }
      }

/* yet to check validity of amount entered - only numbers and then check balance and print
successful*/
```

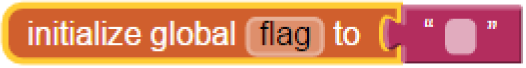
```
else
      {
        /*if((char)(msg[1]=='e') && ((char)msg[2]=='n'))
        {

        }*/
      }
      //record.print();
    }



    delay(3000);
  } while(0);


}

uint32_t createNDEFShortRecord(uint8_t *message, uint8_t payloadLen, uint8_t
*&NDEFMessage)
{
  //Serial.print("Message: ");
  //Serial.println((char *)message);
  uint8_t * NDEFMessageHdr = ALLOCATE_HEADER_SPACE(NDEFMessage,
NDEF_SHORT_RECORD_MESSAGE_HDR_LEN);

  NDEFMessageHdr[0] =  NDEF_MESSAGE_BEGIN_FLAG | NDEF_MESSAGE_END_FLAG |
NDEF_MESSAGE_SHORT_RECORD | TYPE_FORMAT_MEDIA_TYPE;
  NDEFMessageHdr[1] =  SHORT_RECORD_TYPE_LEN;
  NDEFMessageHdr[2] =  payloadLen;
  memcpy(&NDEFMessageHdr[3], TYPE_STR, SHORT_RECORD_TYPE_LEN);
  memcpy(NDEFMessage, message, payloadLen);
  //Serial.print("NDEF Message: ");
  //Serial.println((char *)NDEFMessage);
  NDEFMessage = NDEFMessageHdr;
  return (payloadLen + NDEF_SHORT_RECORD_MESSAGE_HDR_LEN);
}
int mycmp(uint8_t *s1,uint8_t *s2)
{
 int flag1=0;
 for(int i=3;i<6;i++)
 {
  if(s1[i]!=s2[i-3])
  {
   flag1=1;
   break;
  }
 }
 return flag1;
}
```

```
void sleepMCU()
{
  delay(100);  // delay so that debug message can be printed before the MCU goes to sleep

  // Enable sleep mode
  sleep_enable();

  power_adc_disable();
  power_spi_disable();
  power_timer0_disable();
  power_timer1_disable();
  power_timer2_disable();
  power_twi_disable();

  //Serial.println("Going to Sleep\n");
  //delay(1000);

  // Puts the device to sleep.
  sleep_mode();
  Serial.println("Woke up");

  // Program continues execution HERE
  // when an interrupt is recieved.

  // Disable sleep mode
  sleep_disable();

  power_all_enable();
}

String sendData(String command, const int timeout, boolean debug)
{
  String response = "";

  Serial1.print(command); // send the read character to the esp8266

  long int time = millis();

  while( (time+timeout) > millis())
  {
   while(Serial1.available())
   {

     // The esp has data so display its output to the serial window
     char c = Serial1.read(); // read the next character.
     response+=c;
   }
  }
if(debug)
  {
   Serial.print(response);
  }

  return response;
}
```

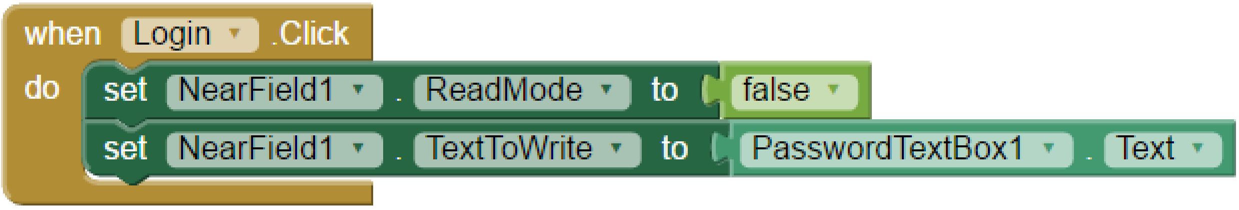
Android Application



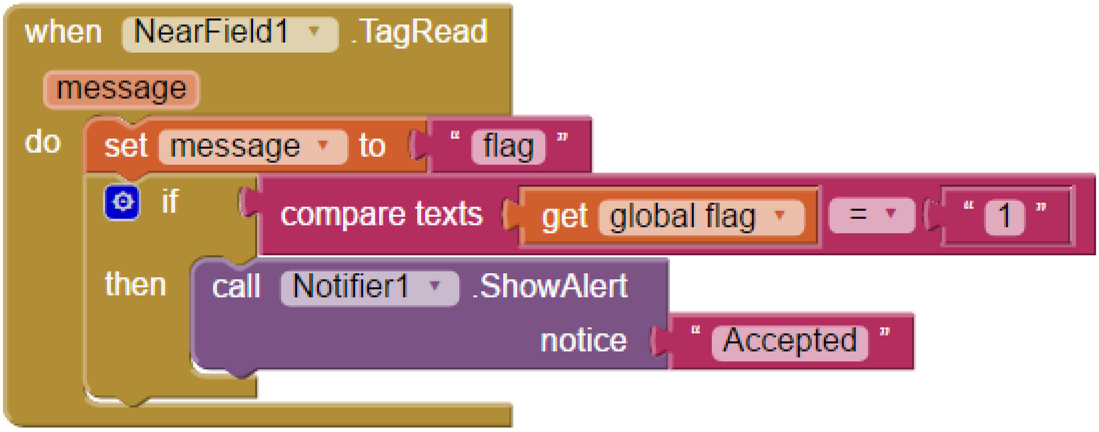
```
when Screen1 .Initialize
do  call Notifier1 .ShowAlert
                   notice  " Welcome to the App ! "
```



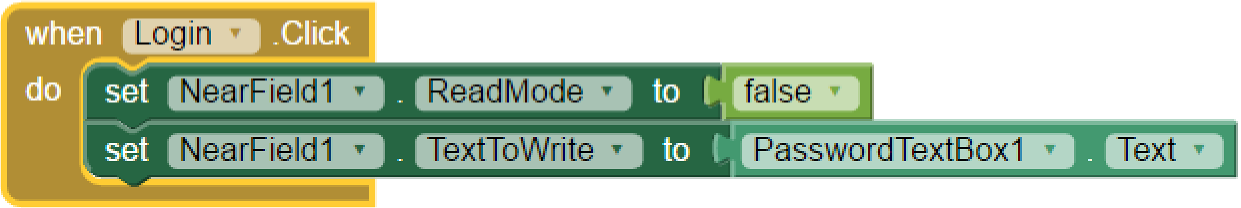
```
initialize global flag to " "
```



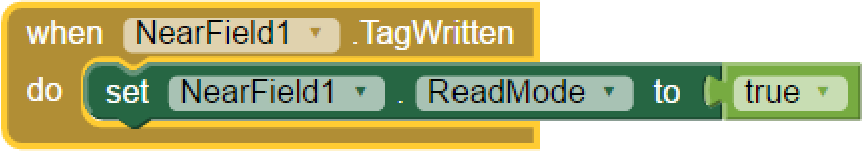
```
when Login .Click
do  set NearField1 . ReadMode to false
    set NearField1 . TextToWrite to PasswordTextBox1 . Text
```



```
when NearField1 .TagRead
  message
do  set message to " flag "
    if    compare texts  get global flag  =  " 1 "
    then  call Notifier1 .ShowAlert
                         notice  " Accepted "
```



```
when Login .Click
do  set NearField1 . ReadMode to false
    set NearField1 . TextToWrite to PasswordTextBox1 . Text
```



```
when NearField1 .TagWritten
do  set NearField1 . ReadMode to true
```

Schematic Diagram

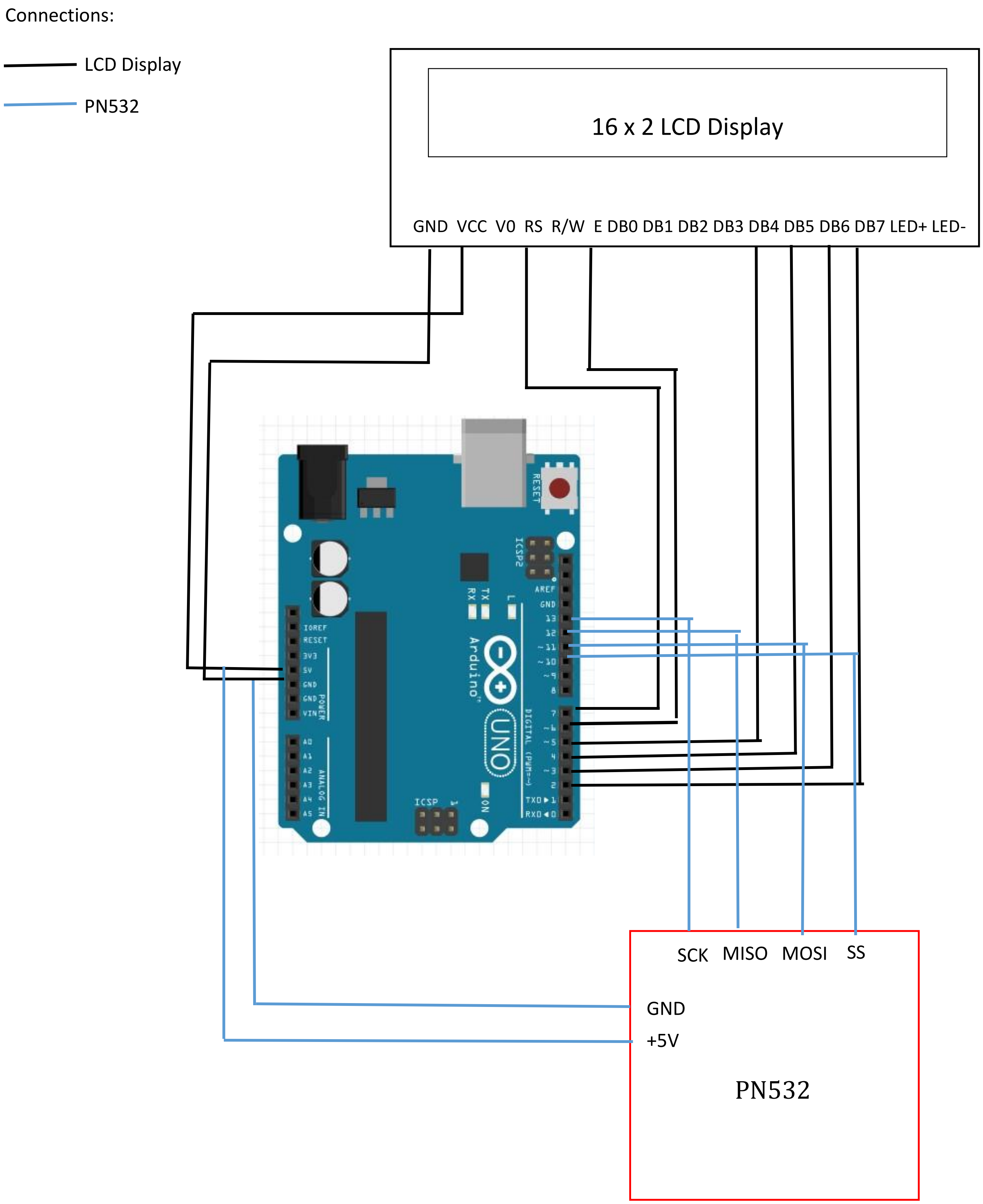


Figure 13: Schematic Diagram – Interfacing Arduino Board with PN532 and LCD Display

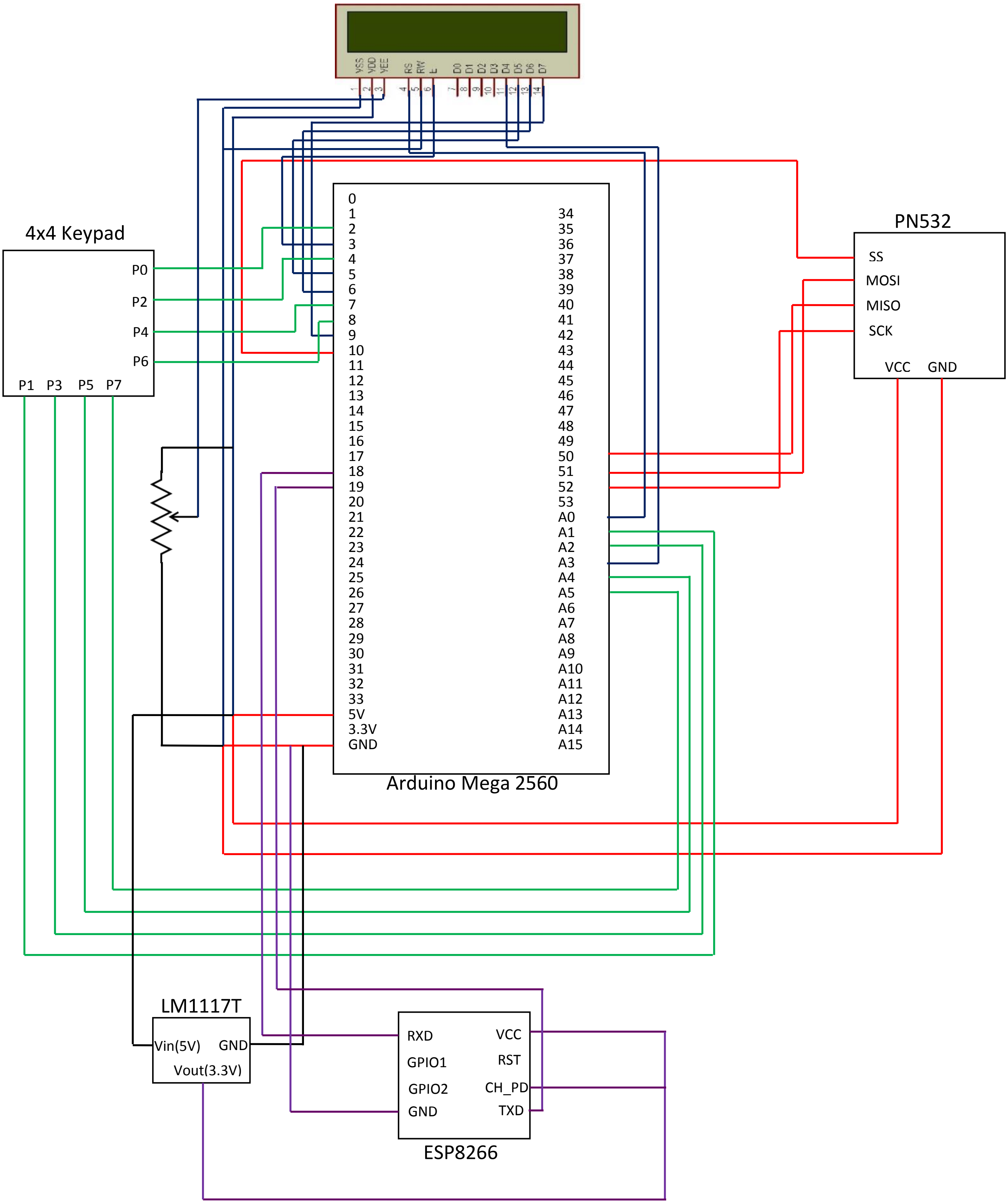
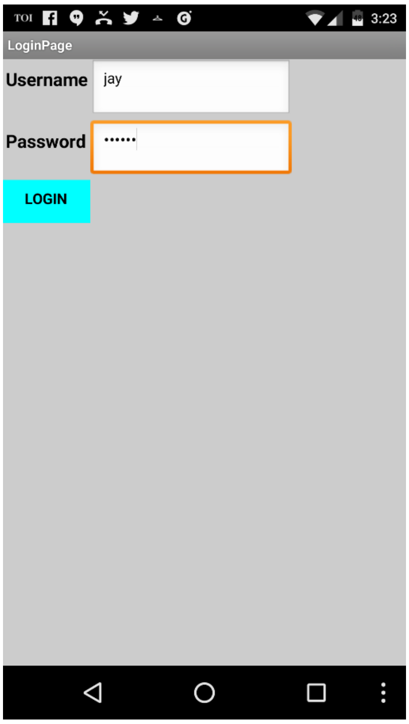


Figure 14: Updated Schematic Diagram

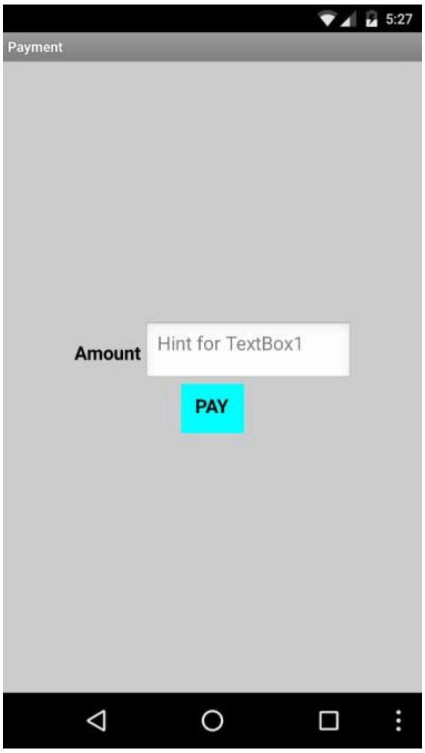
Screenshots of the Android Application (initially built)



Username jay

Password ••••••

Figure 15: Login Screen of Application – Page 1



Amount Hint for TextBox1

Figure 16: Login Screen of Application – Page 2

<u>Screenshots of the Android Application</u> (which supports the final project)

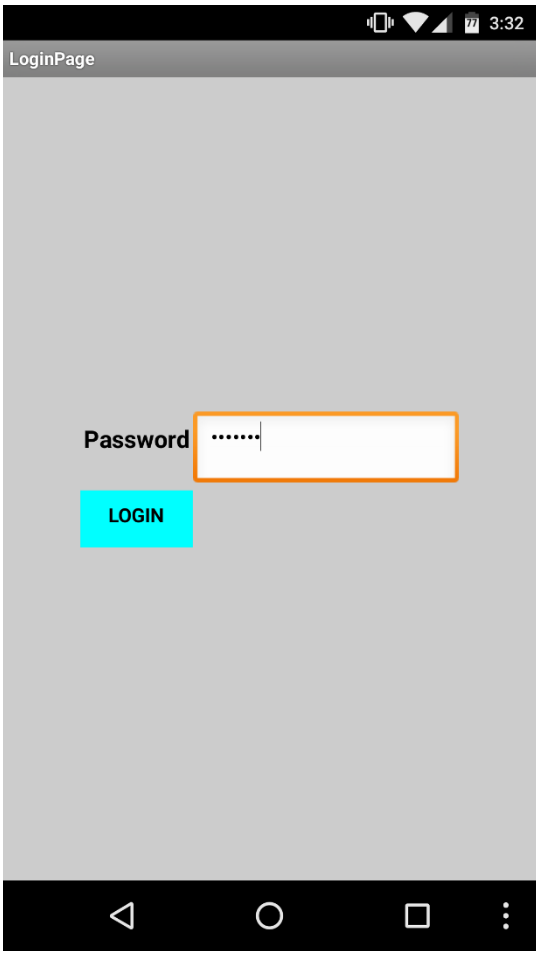


Figure 17: Screen that takes username and password together as input

Circuit Images

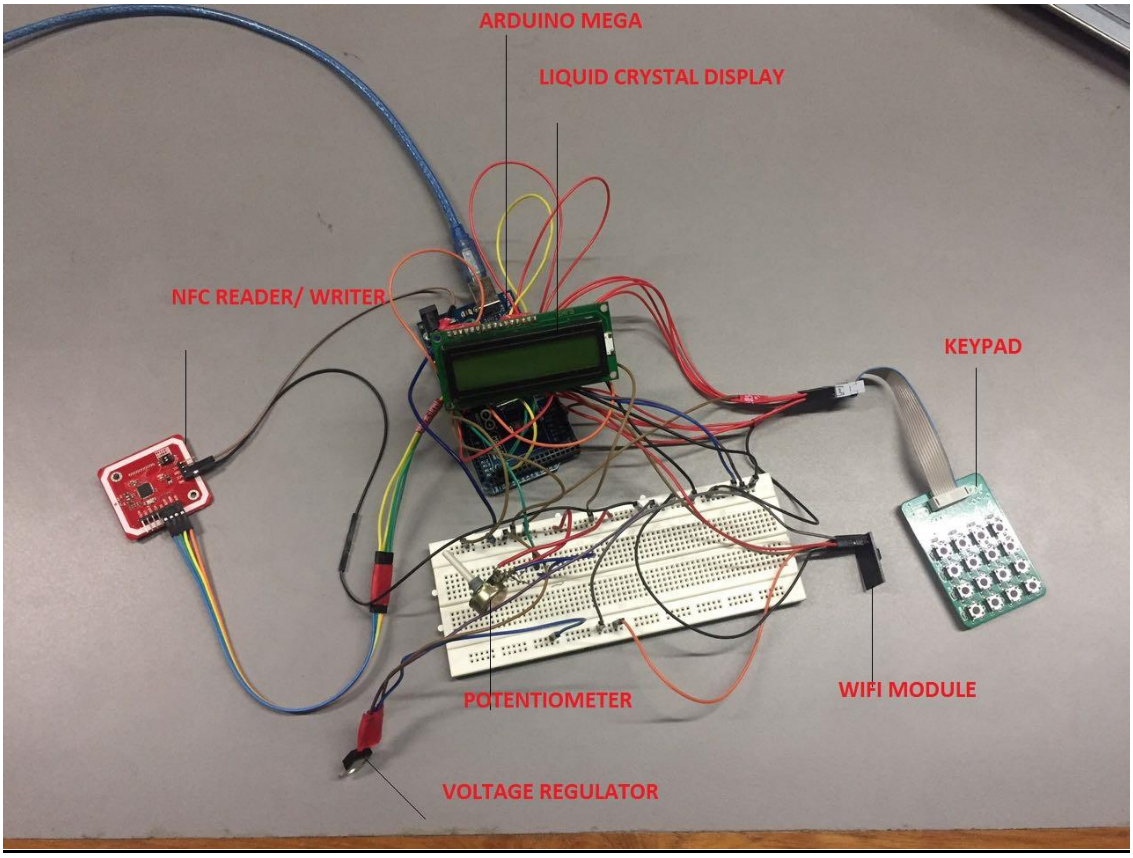


Figure 18: Complete Circuit

Figure 19: Bringing the NFC chip and the Android phone near each other

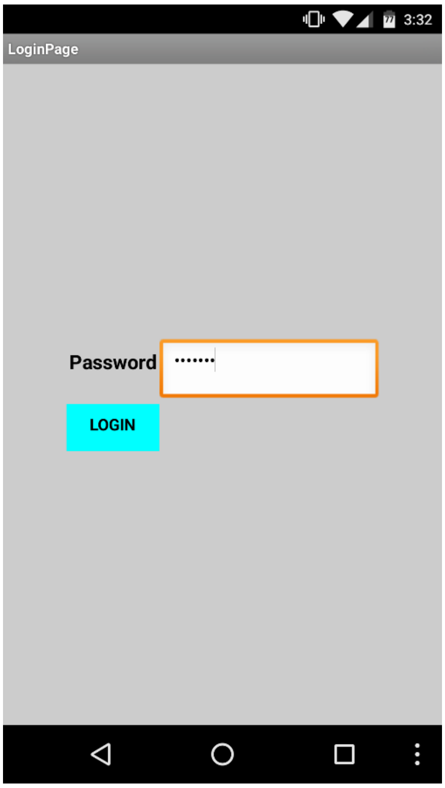
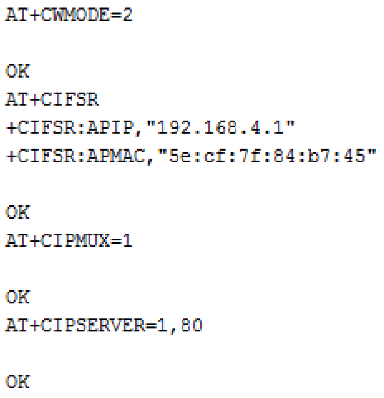


Figure 20: Keypad Pins and Connections



```
AT+CWMODE=2

OK
AT+CIFSR
+CIFSR:APIP,"192.168.4.1"
+CIFSR:APMAC,"5e:cf:7f:84:b7:45"

OK
AT+CIPMUX=1

OK
AT+CIPSERVER=1,80

OK
```

Figure 21: WIFI Hotspot connection made (Serial Monitor)

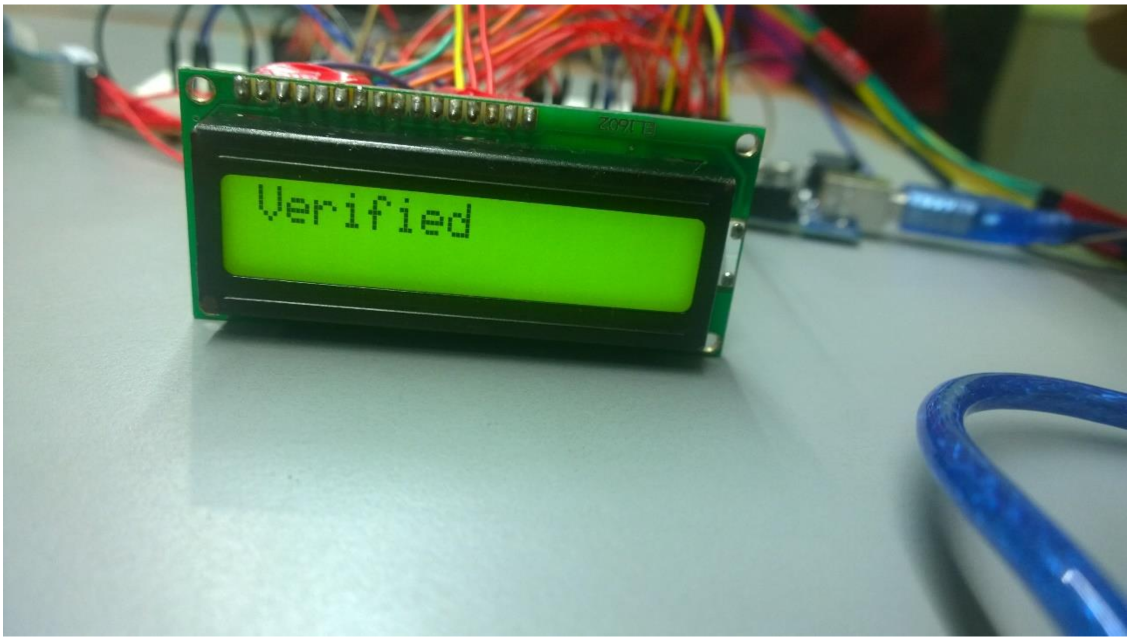


Figure 22: When correct user details are entered

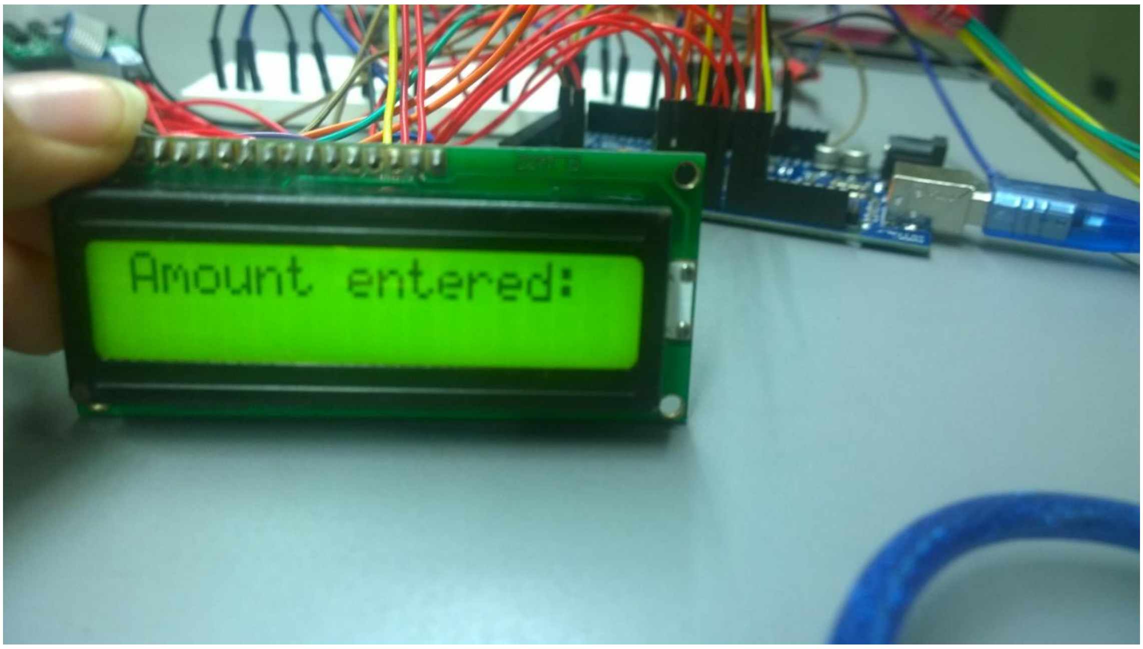
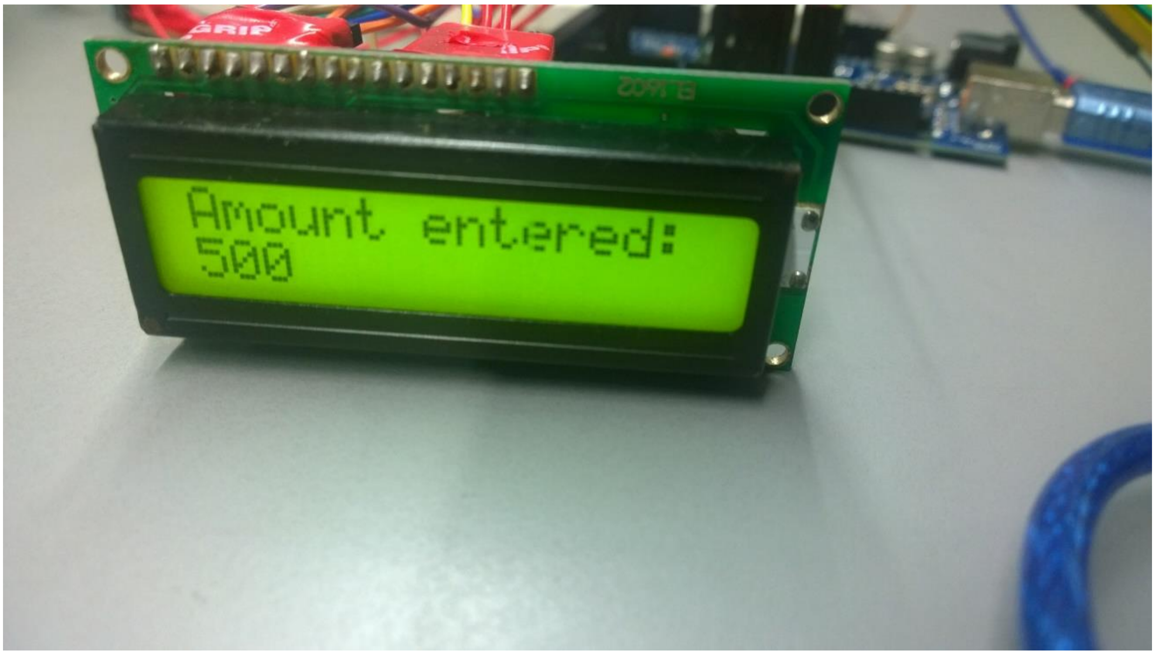
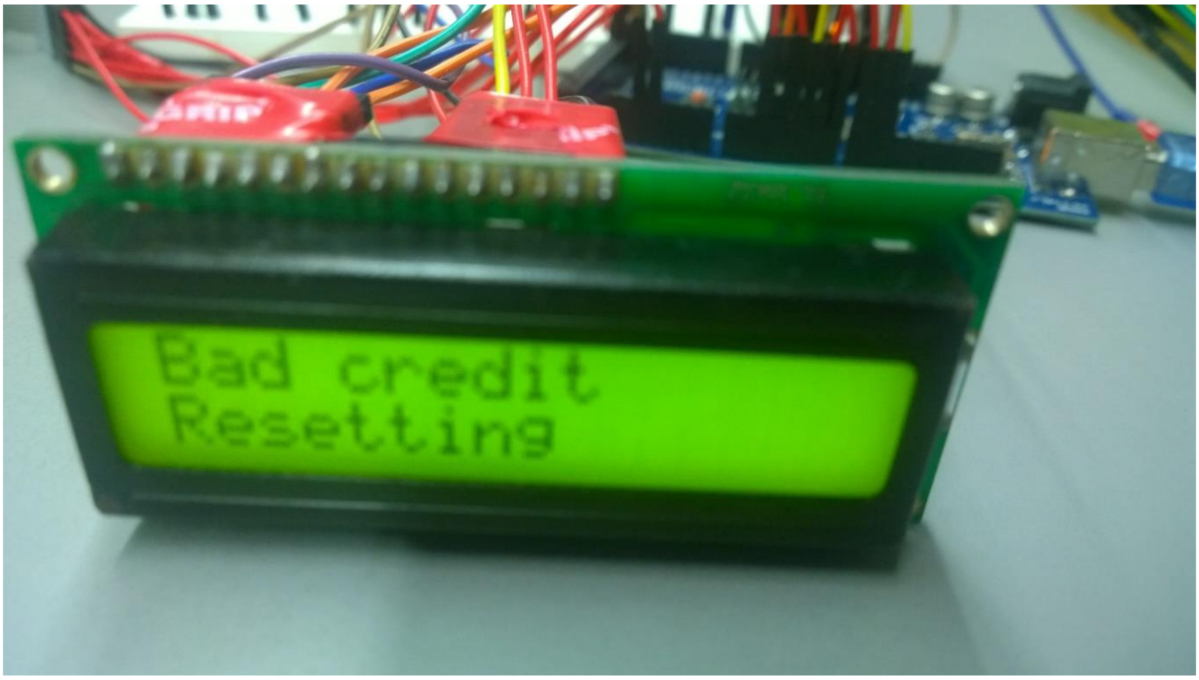


Figure 23: LCD Displays to Enter the Amount to Pay



```
Amount entered:
500
```

Figure 24: When Amount Entered is More than the Balance Amount



```
Bad credit
Resetting
```

Figure 25: LCD Displays BAD CREDIT Message

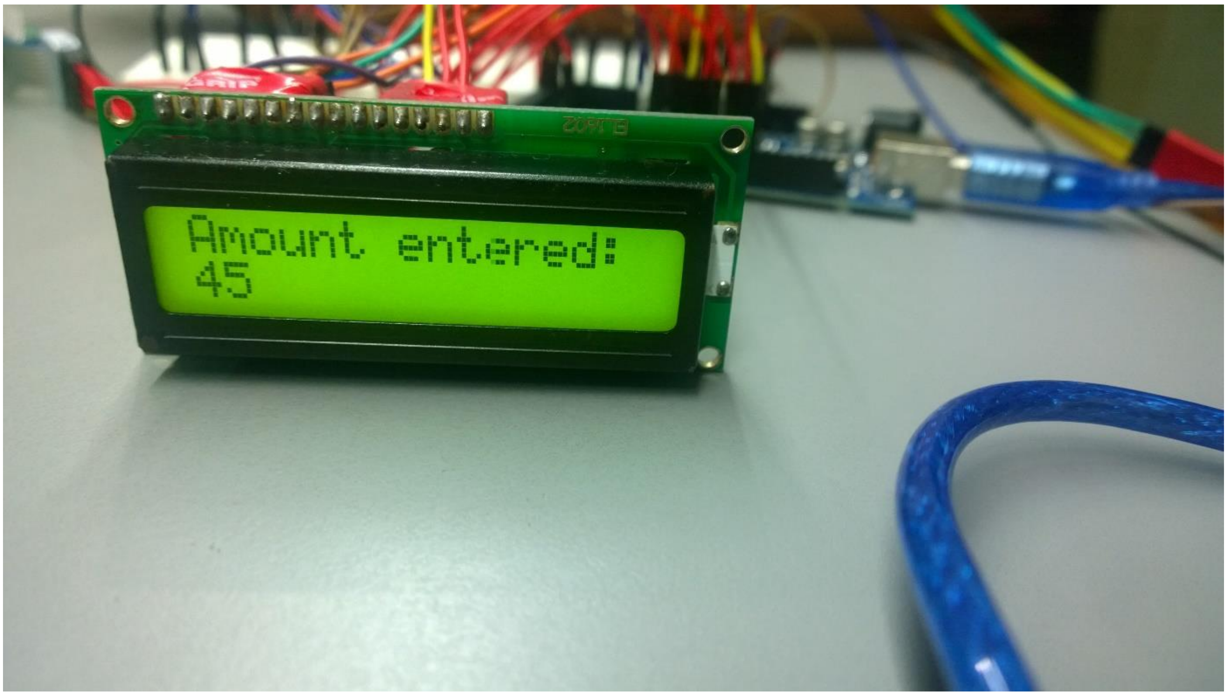


Figure 26: When Amount Entered is less than the balance

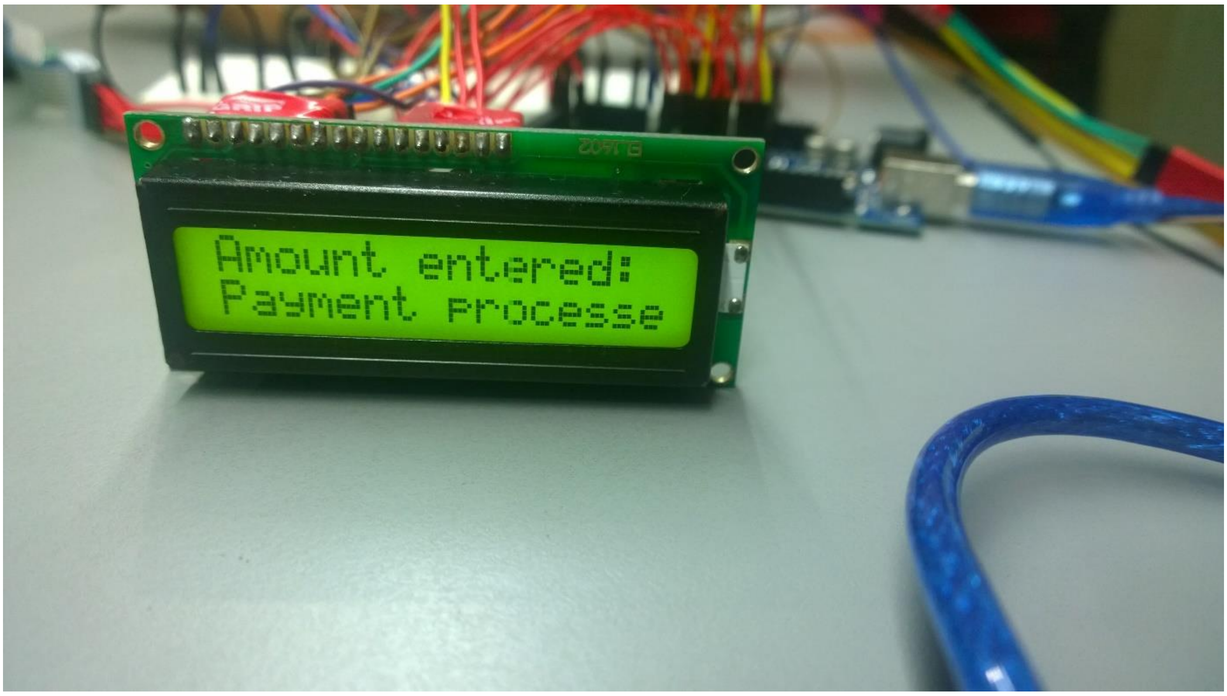
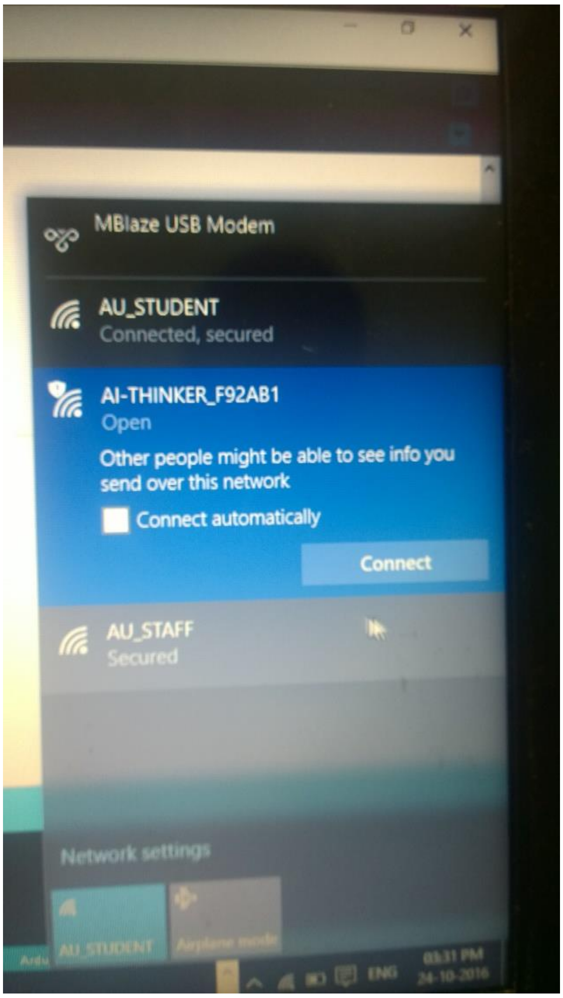


Figure 27: Payment Processed Message is displayed on the LCD



MBlaze USB Modem

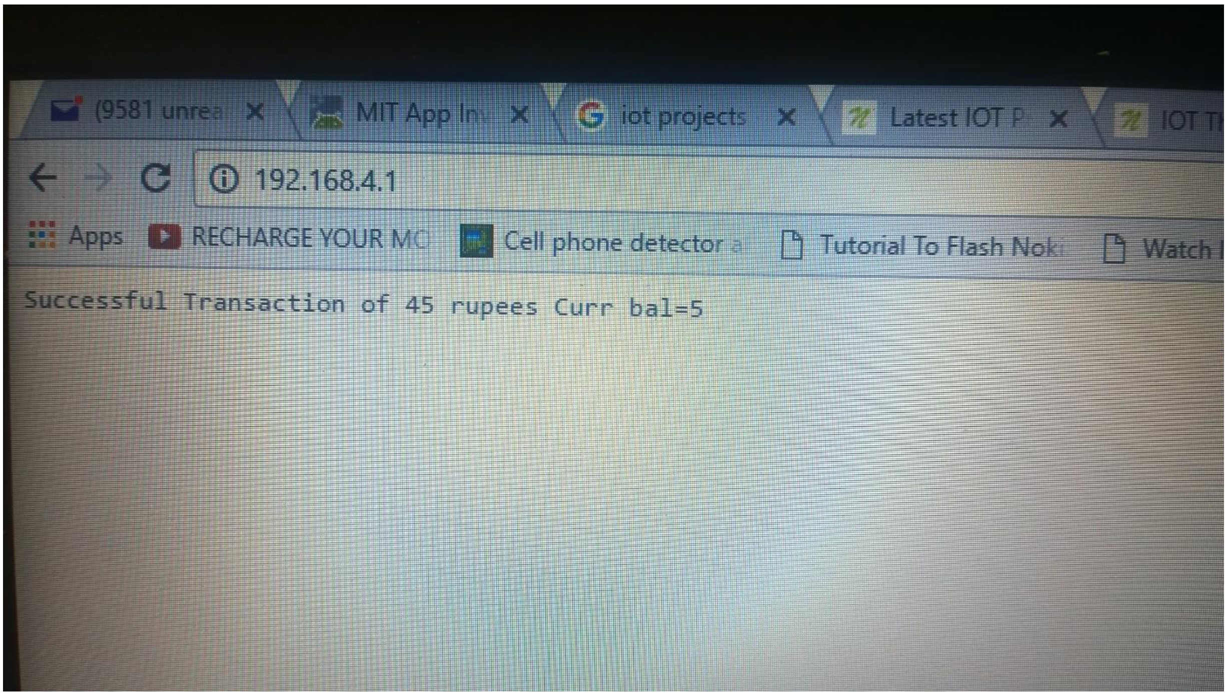
AU_STUDENT
Connected, secured

AI-THINKER_F92AB1
Open

Other people might be able to see info you send over this network

□ Connect automatically

Figure 28: WIFI Module connects to the hotspot created



```
Successful Transaction of 45 rupees Curr bal=5
```

Figure 29: The details of the transaction are displayed on the Client’s screen

References

[1] Andreas Jakl, “How much data can I store on a tag”. Available: https://www.nfcinteractor.com/question/how-much-data-can-i-store-on-a-tag/

[2] Licensed under Android Open Source Project. Available: https://developer.android.com/guide/topics/connectivity/nfc/nfc.html

[3] Woodford, Chris, “LCDs (liquid crystal displays”. Available: http://www.explainthatstuff.com/lcdtv.html

[4] Version 3, “PN532 NFC RFID Module User Guide”. Available: https://dangerousthings.com/wp-content/uploads/2013/12/PN532_-Manual_V3.pdf

[5] PN532 Library. Available: https://github.com/Seeed-Shield/NFC_Shield_DEV

[6] PN532 User Manual. Available: http://www.nxp.com/documents/user_manual/141520.pdf

[7] Figure 4: Showing electromagnetic field. Available: http://www.androidauthority.com/what-is-nfc-270730/

[8] Figure 5: Showing polarization of light. Available: http://physics.stackexchange.com/questions/10794/intrigued-about-a-polarizer-effect

[9] Figure 6, Figure 7: Woodford, Chris, “LCDs (liquid crystal displays”. Available: http://www.explainthatstuff.com/lcdtv.html