Campus Automation Using RFID Technology

By Charles Beam

Spring 2021

Sanjeetha Peters, Senior Lecturer of Mathematics and Computer Science

---

Professor Randy Key, Associate Lecturer of Mathematics

Dr. Emily Allen, Associate Lecturer of English

Dr. Kristi Key, Director of Academic Services

Contents

0 General Project Overview

0.1 Purpose

The purpose of this project is to have students be able to declare their current location. Students do this by having RFID tags that they scan whenever they want to declare their location. Students scan their tags on boxes that are placed around campus where students would need to declare their locations. The RFID box sends these RFID tag scans to a MySQL database for later parsing.

0.2 Use Cases

Some use cases of this project are:

- Automating Classroom Attendance
- Automating SISO
- Automating Speed Bump
- Automating Community Meeting Sign In
- Automating Activity Attendance (Dance Recitals, Food Servings, Blue and Gold Week Presentations)

0.3 Importance of Project

Many required activities at LSMSA are either inefficient or not Covid friendly.

- Classroom attendance takes too long. LSMSA, having 262 classes meeting 2.5 times a week for 32 weeks a year means that attendance is taken 20,960 times a year.

  262 classes * 2.5 meetings/week * 32 weeks = 20,960 attendances taken

  Assuming that taking attendance takes one minute per class period, teachers spend 350 hours or 14.5 days per school year taking attendance. Attendance on a year-long scale takes far too much time and needs to be automated so teachers can do what they do best, teach.

  20,960 attendances/year * 1 minute = 20,960 minutes / 349.33 hours / 14.56 days

- The current solution for student locations (SISO using Reach) is a clunky, expensive mess. Students rarely update it with their actual locations, there is no method to ensure students are being honest, and Reach does not consistently work among many other issues.

- Currently, speed bumping students requires the students to stand at the front desks of the dorms and make sure that the front desk worker sees them. If the front desk worker misses them, the student has to go down at 7 pm to speed bump again. This means multiple students are all walking to the desk at the same time, which is not good for social distancing.
- For community meetings, students wait outside the doors and slowly walk in getting their IDs scanned. This is not a good system since it is not only slow, but makes students group together allowing for Covid to travel between them.
- For food events or events happening in the Recital Hall, students have to sign a sheet of paper stating that they are there. To do this, students all use the same pen and paper, which is not good for preventing germs from spreading.

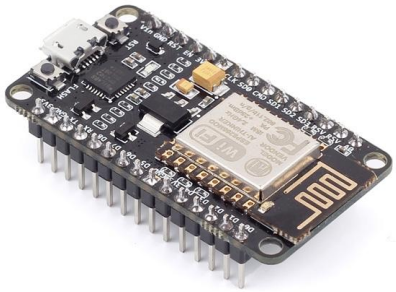
1 Hardware Overview

This project has three main pieces of hardware, all communicating with each other to successfully scan student's RFID IDs.

1.1 RFID Box

RFID Boxes are physically placed at locations where students need to declare their locations. Inside the boxes are two boards communicating with each other: the NodeMCU ESP-12E and the RC522.

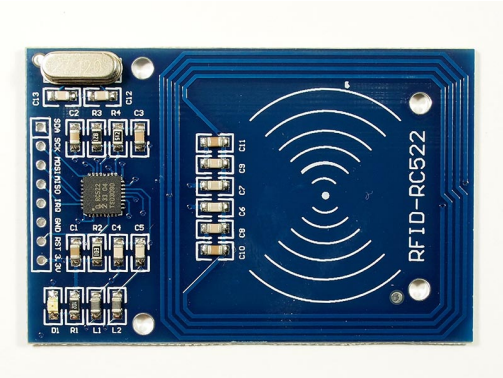
1.1.1 NodeMCU ESP-12E



The NodeMCU ESP-12E is the brain of the RFID box. Essentially, a NodeMCU ESP-12E is an Arduino with a WiFi chip built-in. This allows it to run Arduino code to interact with the RC522 but also be able to communicate with the MySQL database through the school's WiFi.

The NodeMCU ESP-12E also has a blue LED (like a small light bulb) built-in. This LED provides the user with confirmation whenever they scan their RFID ID.

1.1.2 RFID Reader RC522



The RC522 is the board that reads the RFID tags. Whenever a RFID tag is scanned, the contents of the tag are sent to the NodeMCU ESP-12E.

1.2 RFID Tag

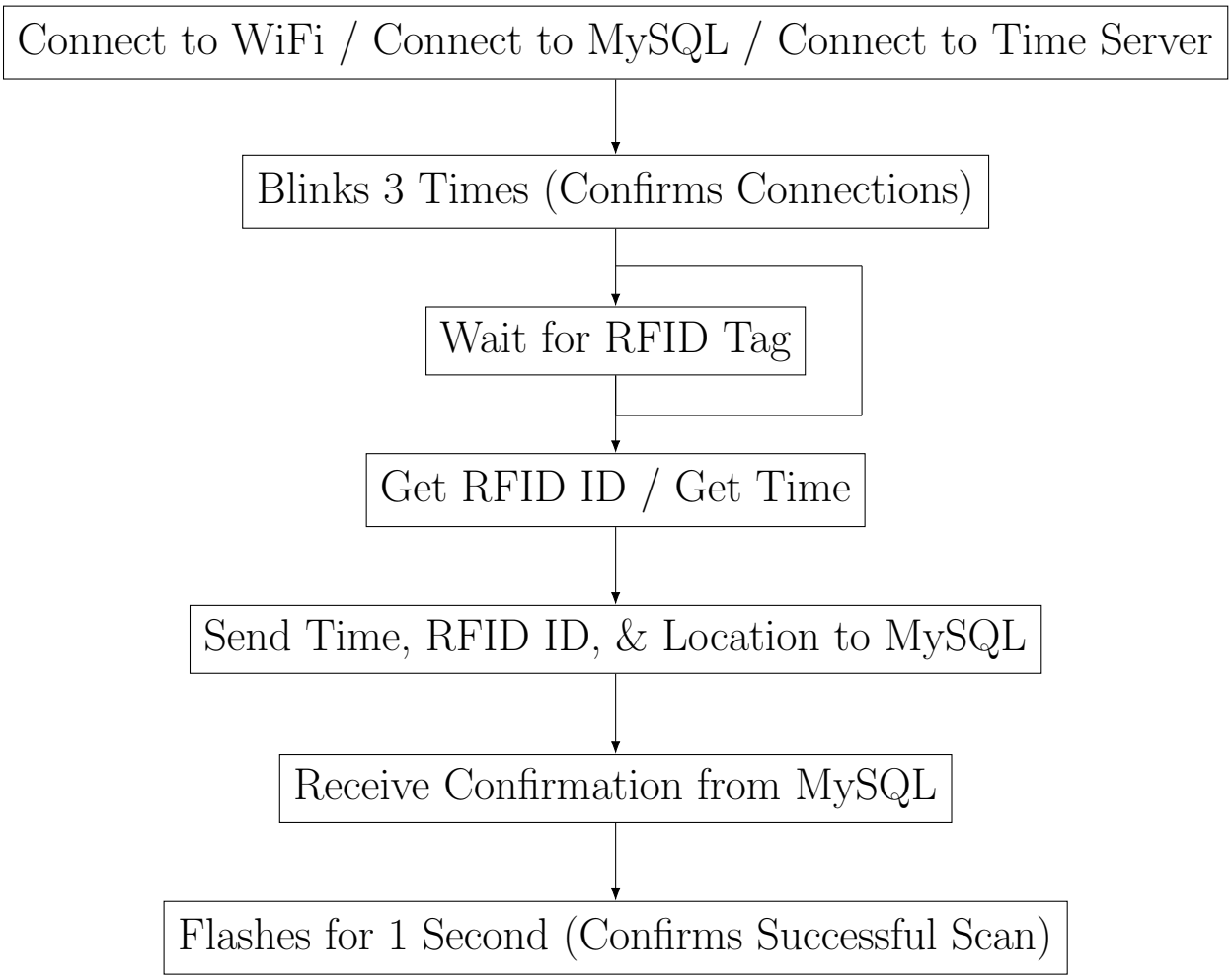
A RFID tag contains an ID that is transmitted to the RC522 when in close proximity. Each student is issued one RFID tag.

2 Software Overview

This project has two main pieces of software: The program on the NodeMCU ESP-12E and the MySQL Database. These two pieces of software communicate with each other over the school's WiFi.

2.1 NodeMCU ESP-12E Program Overview

2.1.1 Program Flowchart



2.1.2 Entire Program

Descriptions of each part of this program is found in the next section.

```
1 #include <SPI.h>
2 #include <MFRC522.h>
3 #include <ESP8266WiFi.h>
4 #include <MySQL_Connection.h>
5 #include <MySQL_Cursor.h>
6 #include <NTPClient.h>
7 #include <WiFiUdp.h>
8
9 #define RST_PIN         D3
```

```
10 #define SS_PIN          D8
11 IPAddress server_addr(MySQL IP Address (Replace periods with
       commas));  // IP of the MySQL database
12 char user[] = "MySQL Username";              // MySQL Username
13 char password[] = "MySQL Password";        // MySQL Password
14 char ssid[] = "WiFi Username";             // WiFi Username
15 char pass[] = "WiFi Password";             // WiFi Password
16 WiFiClient client;
17 MySQL_Connection conn(&client);
18 MySQL_Cursor* cursor;
19
20 const long utcOffsetInSeconds = -18000;
21
22 WiFiUDP ntpUDP;
23 NTPClient timeClient(ntpUDP, utcOffsetInSeconds);
24 String timer;
25 char timerarray[9];
26 char location[] = "ClassT231";
27 MFRC522 mfrc522(SS_PIN, RST_PIN);
28 String uidstring;
29 char uid[12];
30 void setup() {
31
32   Serial.begin(115200);
33   SPI.begin();
34   pinMode(LED_BUILTIN, OUTPUT);
35   Serial.printf("\nConnecting to %s", ssid);
36   WiFi.begin(ssid, pass);
37   while (WiFi.status() != WL_CONNECTED) {
38     delay(500);
39     Serial.print(".");
40   }
41   Serial.println("\nConnected to network");
42   Serial.print("My IP address is: ");
43   Serial.println(WiFi.localIP());
44   Serial.print("Connecting to SQL...  ");
```

```
45   if (conn.connect(server_addr, 3306, user, password/*,
        default_db*/))
46     Serial.println("MySQL Connected");
47   else
48     Serial.println("MySQL Failed");
49
50
51   cursor = new MySQL_Cursor(&conn);
52   mfrc522.PCD_Init();
53   timeClient.begin();
54   Serial.println(F("Read personal data on a MIFARE PICC:"));
55   if(WL_CONNECTED && conn.connected())
56         digitalWrite(LED_BUILTIN, HIGH);
57         delay(100);
58         digitalWrite(LED_BUILTIN, LOW);
59         delay(100);
60         digitalWrite(LED_BUILTIN, HIGH);
61         delay(100);
62         digitalWrite(LED_BUILTIN, LOW);
63         delay(100);
64         digitalWrite(LED_BUILTIN, HIGH);
65         delay(100);
66         digitalWrite(LED_BUILTIN, LOW);
67         delay(100);
68         digitalWrite(LED_BUILTIN, HIGH);
69 }
70
71 void loop() {
72
73   if ( ! mfrc522.PICC_IsNewCardPresent()) {
74     return;
75   }
76
77   if ( ! mfrc522.PICC_ReadCardSerial()) {
78     return;
79   }
80   timeClient.update();
```

```
81   uidstring = String(mfrc522.uid.uidByte[0], HEX) + "␣" + String(
       mfrc522.uid.uidByte[1], HEX) + "␣" + String(mfrc522.uid.
       uidByte[2], HEX) + "␣" + String(mfrc522.uid.uidByte[3], HEX)
       ;
82   uidstring.toUpperCase();
83   uidstring.toCharArray(uid, 12);
84   timer = (timeClient.getFormattedTime());
85   timer.toCharArray(timerarray, 9);
86   char INSERT_SQL[50];
87   sprintf(INSERT_SQL, "INSERT␣INTO␣Classrooms_Prod.%s␣(timedate,␣
       rfid_id,␣location)␣VALUES␣('%s',␣'%s',␣'%s')", location,
       timerarray, uid, location);
88   Serial.println(INSERT_SQL);
89   if (conn.connected())
90     cursor->execute(INSERT_SQL);
91   digitalWrite(LED_BUILTIN, LOW);
92   delay(1000);
93   digitalWrite(LED_BUILTIN, HIGH);
94 }
```

2.1.3 Libraries Used

```
1 #include <SPI.h>
2 #include <MFRC522.h>
3 #include <ESP8266WiFi.h>
4 #include <MySQL_Connection.h>
5 #include <MySQL_Cursor.h>
6 #include <NTPClient.h>
7 #include <WiFiUdp.h>
```

The purpose of importing these libraries is to access functions needed for the RFID boxes to work.

Line descriptions (purpose of each library):

1. Used as interface method for RC522.
2. Used to access methods needed to interface with RC522.
3. Used to interface with ESP-12E and access the internet.
4. Used to connect to MySQL database.

5. Used to interface with MySQL database.
6. Used to interface with NTP (time server).
7. Used to connect with NTP (time server).

2.1.4 RFID Reader RC522 Setup

```
1 #define RST_PIN         D3
2 #define SS_PIN          D8
3 MFRC522 mfrc522(SS_PIN, RST_PIN);
4 String uidstring;
5 char uid[12];
```

Line descriptions:

1. Sets the RST pin used to interface with the RC522. This can be changed if using a board other than NodeMCU ESP-12E.
2. Sets the SS pin used to interface with the RC522. This can be changed if using a board other than NodeMCU ESP-12E.
3. Gets the RST and SS pin numbers and creates a RC522 instance.
4. Creates the unformatted string that the RFID ID is stored on.
5. Creates the formatted RFID ID char array.

2.1.5 WiFi Setup

```
1 char ssid[] = "WiFi␣Username";          // WiFi Username
2 char pass[] = "WiFi␣Password";        // WiFi Password
3 WiFiClient client;
```

Line descriptions:

1. This is the location of the WiFi username. It can be changed to whatever the network's name is. Note: keep the username in the quotation marks. Ex: "username".
2. This is the location of the WiFi password. It can be changed to whatever the network's password is. Note: keep the password in the quotation marks. Ex: "password".
3. Initializes the WiFiClient class.

2.1.6 Time Server Setup

```
1 WiFiUDP ntpUDP;
2 const long utcOffsetInSeconds = -18000;
```

```
3 NTPClient timeClient(ntpUDP, utcOffsetInSeconds);
4 String timer;
5 char timerarray[9];
```

Line descriptions:

1. Initializes the WiFiUDP class and declares a ntpUDP object.
2. Sets the time offset from UTC time. In CST this is -18000 seconds. This can be changed for different time-zones.
3. Initializes the NTPClient class giving it the ntpUDP object from line 1 and the time offset from line 2.
4. Creates unformatted string that the time is stored on.
5. Creates the formatted time char array.

2.1.7 Connecting to WiFi

```
1 Serial.printf("\nConnecting␣to␣%s", ssid);
2 WiFi.begin(ssid, pass);
3  while (WiFi.status() != WL_CONNECTED) {
4    delay(500);
5    Serial.print(".");
6  }
7 Serial.println("\nConnected␣to␣network");
8 Serial.print("My␣IP␣address␣is:␣");
9 Serial.println(WiFi.localIP());
```

Line descriptions:

1. Prints to the serial console that it is connecting to WiFi and includes the WiFi username. Nothing is printed whenever the RFID boxes are not connected to a computer.
2. Connects to WiFi
3. Starts a loop that stops whenever connected to the WiFi.
4. Waits 0.5 seconds before moving to the next line. If there is no delay, then "....." would spam the console and make it hard to understand.
5. Prints a "." to the serial console for testing.
6. Ends the loop started in line 3. The only way to get past this line in the program is to have a successful WiFi connection.
7. Prints to the serial console that the WiFi connection was successful.
8. Prints to the serial console "My IP address is: ".
9. Prints to the serial console the IP address of the RFID box.

2.1.8 Connecting to MySQL Database

```
1 IPAddress server_addr(MySQL IP address);
2 char user[] = "MySQL Username";                // MySQL Username
3 char password[] = "MySQL Password";          // MySQL Password
4 char location[] = "ClassT231";
5 MySQL_Connection conn(&client);
6 MySQL_Cursor* cursor;
```

Line descriptions:

1. Declares the IP address of the MySQL database.
2. This is the location of the MySQL database username. It can be changed to whatever the MySQL username is. Note: keep the username in the quotation marks. Ex: "admin".
3. This is the location of the MySQL database password. It can be changed to whatever the MySQL password is. Note: keep the password in the quotation marks. Ex: "password".
4. This is the location of the RFID box location.
5. Connects to the MySQL database using the username and password declared in lines 2 and 3.
6. Sets the location to interface with the MySQL database.

2.1.9 Connecting to Time Server

```
1 timeClient.begin();
```

Line descriptions:

1. Connects to the time server.

2.1.10 Checking Connection / Startup and Connecting Successful LED Blink

```
1 if(WL_CONNECTED && conn.connected()){
2       digitalWrite(LED_BUILTIN, HIGH);
3       delay(100);
4       digitalWrite(LED_BUILTIN, LOW);
5       delay(100);
6       digitalWrite(LED_BUILTIN, HIGH);
7       delay(100);
```

```
8         digitalWrite(LED_BUILTIN, LOW);
9         delay(100);
10        digitalWrite(LED_BUILTIN, HIGH);
11        delay(100);
12        digitalWrite(LED_BUILTIN, LOW);
13        delay(100);
14        digitalWrite(LED_BUILTIN, HIGH);
15 }
```

Line descriptions:

1. Checks if the RFID box is successfully connected to the WiFi and MySQL database. The lines after this only run if these connections are successful.
2. Turns off the LED on the NodeMCU ESP-12E.
3. Waits 0.1 seconds.
4. Turns on the LED on the NodeMCU ESP-12E.
5. Waits 0.1 seconds.
6. Turns off the LED on the NodeMCU ESP-12E.
7. Waits 0.1 seconds.
8. Turns on the LED on the NodeMCU ESP-12E.
9. Waits 0.1 seconds.
10. Turns off the LED on the NodeMCU ESP-12E.
11. Waits 0.1 seconds.
12. Turns on the LED on the NodeMCU ESP-12E.
13. Waits 0.1 seconds.
14. Turns off the LED on the NodeMCU ESP-12E.
15. Ends the conditional in item 1. Lines after this will run whether the connection was successful or not, as checked in line 1.

2.1.11 Waiting for RFID Tag

```
1 Serial.println(F("Read␣personal␣data␣on␣a␣MIFARE␣PICC:"));
2 void loop() {
3   if ( ! mfrc522.PICC_IsNewCardPresent()) {
4     return;
5   }
6
7   if ( ! mfrc522.PICC_ReadCardSerial()) {
8     return;
```

```
9     }
```

Line descriptions:

1. Prints to the console that the RFID box is ready to read RFID tags.
2. Begins the main loop of the program. Everything within this loop runs repeatedly after the setup.
3. Checks if a new RFID tag is present. Line 4 only runs if this is not true and there is no RFID tag present.
4. Since there is no RFID tag present, the loop is restarted and the program goes back to line 2.
5. Ends the if statement from line 3. Everything after this runs only if a new card is present, since if a new card is not present, the loops keeps restarting before getting to this point.
6.
7. Checks if the RFID tag is readable. Line 8 only runs if this is not true and the RFID tag is not readable.
8. Since the RFID tag is not readable, the loop is restated and the program goes back to line 2.
9. Ends the if statement from line 7. Everything after this runs only the scanned RFID tag is readable.

2.1.12 Retrieve RFID ID and Time

```
1 uidstring = String(mfrc522.uid.uidByte[0], HEX)
2 + "␣" + String(mfrc522.uid.uidByte[1], HEX)
3 + "␣" + String(mfrc522.uid.uidByte[2], HEX)
4 + "␣" + String(mfrc522.uid.uidByte[3], HEX);
5 uidstring.toUpperCase();
6 uidstring.toCharArray(uid, 12);
7 timer = (timeClient.getFormattedTime());
8 timer.toCharArray(timerarray, 9);
```

Line descriptions:

1. Sets the unformatted RFID ID string to include the first byte (two hexadecimal digits) of data. Note: in the actual program, lines 1-4 are all on the same line. The only reason they are separated in this example for formatting.
2. Adds a space and the second byte of data to the RFID ID string.
3. Adds a space and the third byte of data to the RFID ID string.

4. Adds a space and the fourth byte of data to the RFID ID string.
5. Sets all the characters in the unformatted RFID ID string to be capital for formatting reasons.
6. Changes the unformatted RFID ID string to a char array for compatibility with the MySQL library and stores this formatted string in the uid char array.
7. Retrieves the time from the time server and stores it in the unformatted time string.
8. Changed the unformatted time string to a char array for compatibility with the MySQL library and stores this formatted string in the timearray char array.

2.1.13 Send Time, RFID ID, and Location to MySQL Database

```
1 char INSERT_SQL[50];
2 sprintf(INSERT_SQL, "INSERT␣INTO␣Classrooms_Prod.%s
3 (timedate,␣rfid_id,␣location)␣VALUES␣('%s',␣'%s',␣'%s')",
4 location, timerarray, uid, location);
5 Serial.println(INSERT_SQL);
6 if (conn.connected())
7   cursor->execute(INSERT_SQL);
```

Line descriptions:

1. Creates a char array to store the command that will be sent to the MySQL database.
2. sprintf() allows for the program to insert char arrays into other char arrays. Starts the char array.
3. Continues the char array.
4. Gives the method the char arrays for location, time, and RFID ID.
5. Prints the finished MySQL command to the serial console.
6. Checks if the MySQL connection is successful. Line 7 only runs if this is true and the connection is successful.
7. Sends the MySQL command to the database.

2.1.14 Confirm Successful Scan with 1 Second Flash

```
1 digitalWrite(LED_BUILTIN, LOW);
2 delay(1000);
3 digitalWrite(LED_BUILTIN, HIGH);
```

The purpose of this section of code is to confirm to the user that their RFID tag has been successfully scanned.

Line descriptions:

1. Turns the LED on.
2. Waits 1 second.
3. Turns the LED off.

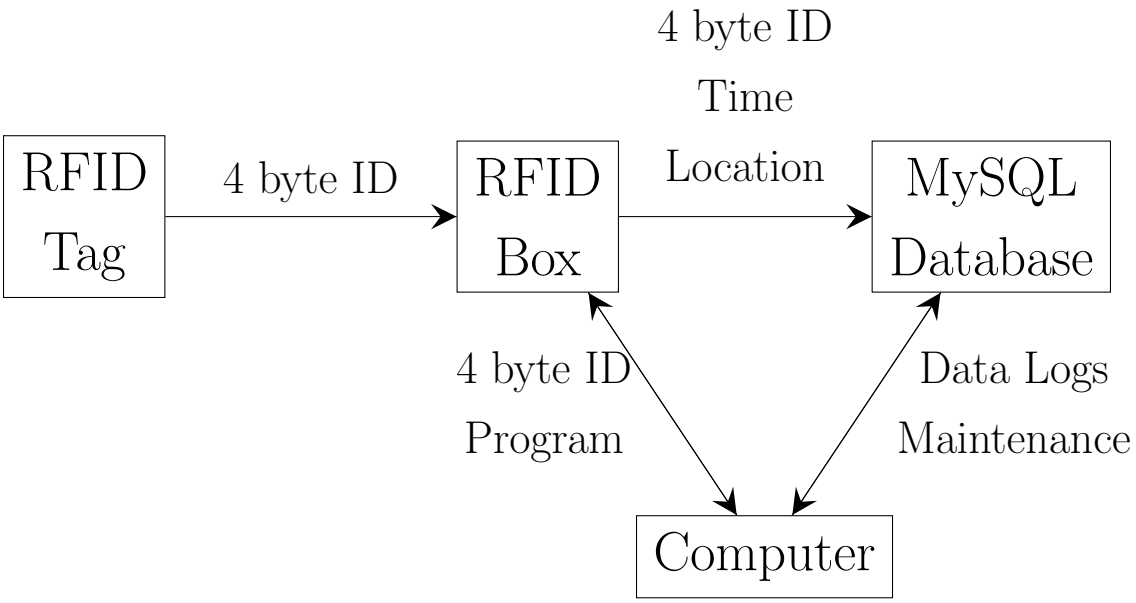
2.2 MySQL Database

A good way to think of the MySQL database is like an Excel spreadsheet. Here is a picture of the current iteration of the MySQL table:

timedate	rfid_id	location
10:00:23	02 43 1F 14	ClassT231
9:55:40	1A 7B 91 B2	ClassT231
9:56:35	AA 50 85 B1	ClassT231
9:58:25	4E 16 C2 E1	ClassT231
9:58:50	7E B6 53 2E	ClassT231
9:59:10	3D D9 B9 6C	ClassT231

The columns in this table represent a type of data, whereas every row represents a different log.
The timedate is the time of the scan, rfid_id is the RFID ID of the scan, and the location is the location of the scan. The timedate data is in military time to avoid needing am/pm.

2.3 Data Connections



This shows the data that is transferred from and to all the components of this system.

3 Buying Parts

The parts mentioned in the following breakdown were bought on Amazon. Now that the economic effects of the pandemic seem to be over, these parts can be ordered online though a site like Alibaba for much cheaper since the parts would be bought in bulk and straight from a factory.f.

3.1 Cost Breakdown

For an entire project cost breakdown, refer to section 10.

3.1.1 RFID Box

Single Box Cost:

Name	Price	Quantity	Cost
NodeMCU ESP-12E	$4.67	1	$4.67
RC522	$2.56	1	$2.56
Micro-B Cable	$1.00	1	$0.06
M3 12mm Screws	$0.06	4	$0.24
M3 Locknuts	$0.05	4	$0.20
Box Material	$0.89	1	$0.89
			$8.62

Total cost per RFID box: $8.62
If the school wants to use 50 boxes, then the cost would be $431.

3.1.2 RFID Tag

Name	Price	Quantity	Cost
13.56MHz RFID Key Fob Write Only (Black)	$0.18	360	$64.80

Total cost per RFID tag: $0.18
Assuming the school has 360 , then the cost for all RFID tags would be $64.80.

3.1.3 MySQL Database

Name	Price	Quantity	Cost
db.t2.micro	$0.0116/hr	12 Months/12hrs a day	$50.81

Total Cost of Database: $50.81 per year

3.2 Where to Buy Parts

3.2.1 NodeMCU ESP-12E

https://www.amazon.com/HiLetgo-Internet-Development-Wireless-Micropython/dp/B081CSJV2V/ref=sr_1_3?dchild=1&keywords=nodemcu+esp12e&qid=1614801567&sr=8-3

3.2.2 RFID Reader RC522

https://www.amazon.com/Qunqi-Sensor-Module-Arduino-Raspberry/dp/B07QBPGYBF/ref=sr_1_4?dchild=1&keywords=rc522&qid=1614801411&sr=8-4

3.2.3 Micro-B Cable

https://www.dollartree.com/e-circuit-micro-usb-cables-39-in/259578

3.2.4 M3 12mm Screws

https://www.amazon.com/Uxcell-a15070200ux0058-Stainless-Phillips-Screws/dp/B012TE1TBS/ref=sr_1_5?dchild=1&keywords=m3+x+12+screw&qid=1619772138&sr=8-5

3.2.5 M3 Locknuts

https://www.amazon.com/100Pcs-Stainless-Self-Lock-Inserted-Clinching/dp/B075ZZW7VL/ref=sr_1_3?dchild=1&keywords=m3+locknut&qid=1619772083&sr=8-3

3.2.6 3D Printing Filament for RFID Box Case

https://www.microcenter.com/product/485634/inland-175mm-black-pla-3d-printe

3.2.7 RFID Tag

https://www.amazon.com/gp/product/B0897KHNHV/ref=ppx_yo_dt_b_asin_title_o07_s00?ie=UTF8&psc=1

3.2.8 MySQL Database

https://aws.amazon.com/rds/mysql/

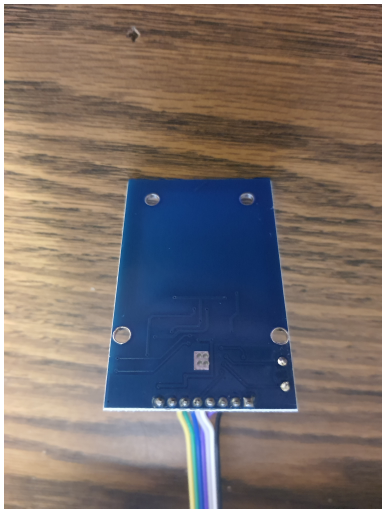
4 RFID Box Setup

4.1 Printing RFID Box Case

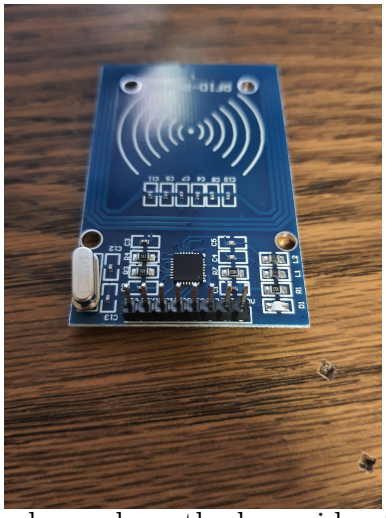
Print the RFID box case with support turned on. The other settings do not matter.

4.2 Soldering RC522

An eight pin long straight row pin is soldered on so that it is easy to connect the RC522 to NodeMCU ESP-12E



Bottom of RC522 to show soldering.
The pins are soldered on the bottom.



Top of RC522 to show where the long side of pins should be.
The long end of the pins come out of the top.

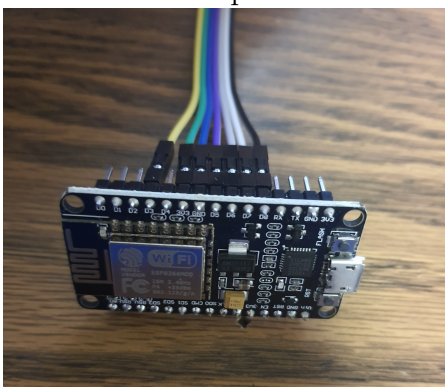
4.3 Wiring RC522 with NodeMCU ESP-12E

Using jumper wires or 22 AWG wires. The exact method does not matter as long as these pins are connected:

RC522	⟶	Node ESP-12E
3.3V	⟶	3.3V
RST	⟶	D3
GND	⟶	GND
IRQ	⟶	Not Connected
MISO	⟶	D6
MOSI	⟶	D7
SCK	⟶	D5
SS	⟶	D8

Pictures of Wiring:

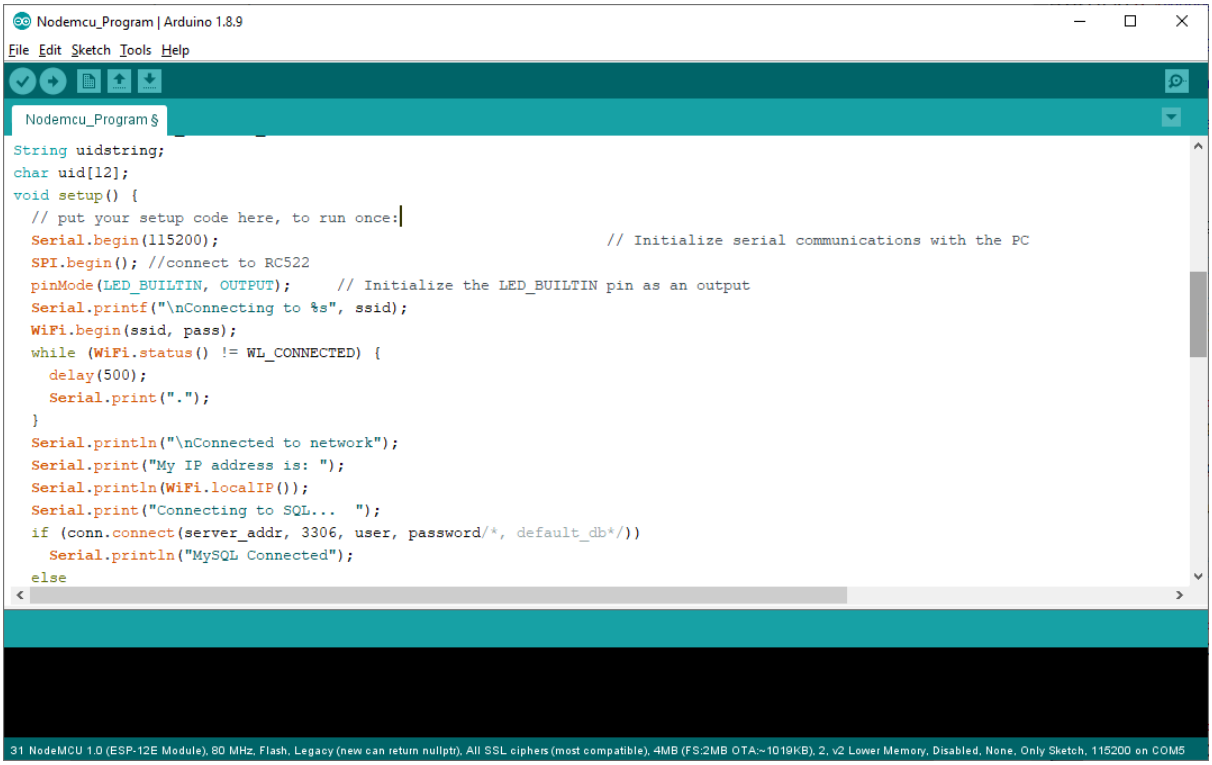
RC522 pinout



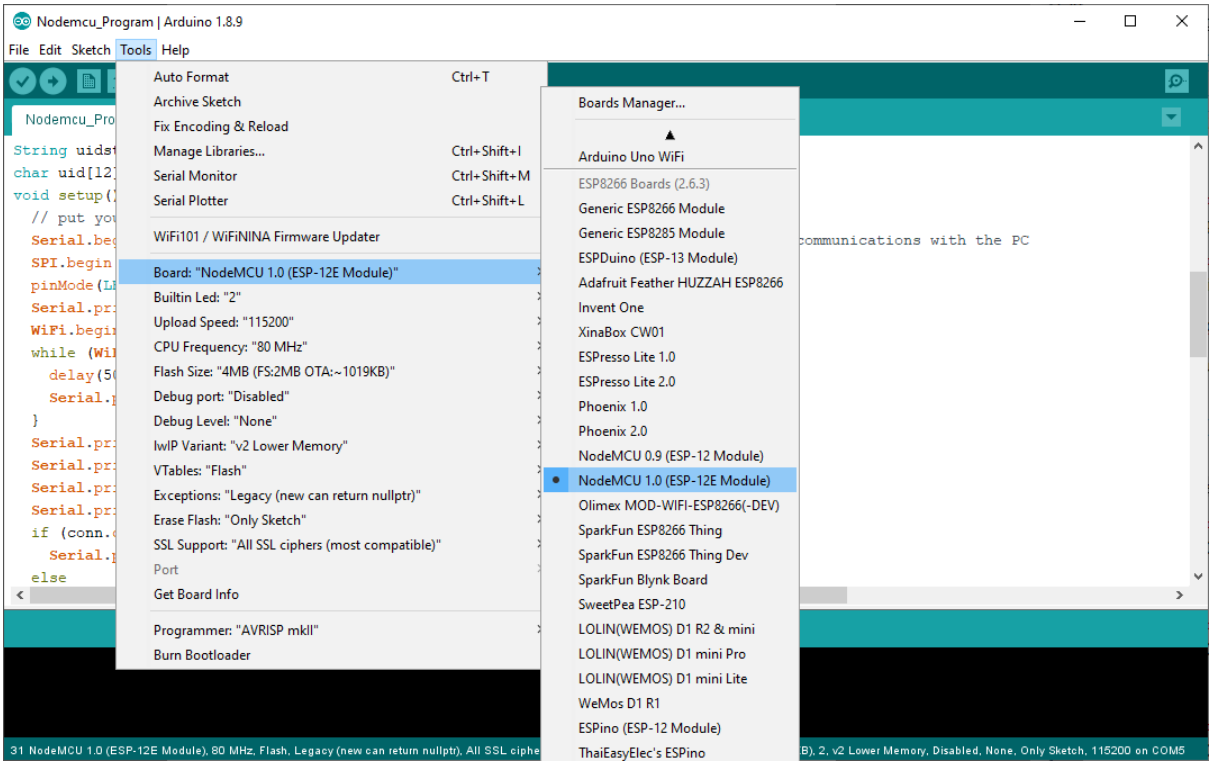
NodeMCU ESP-12E pinout

4.4 Setting Up Arduino Software

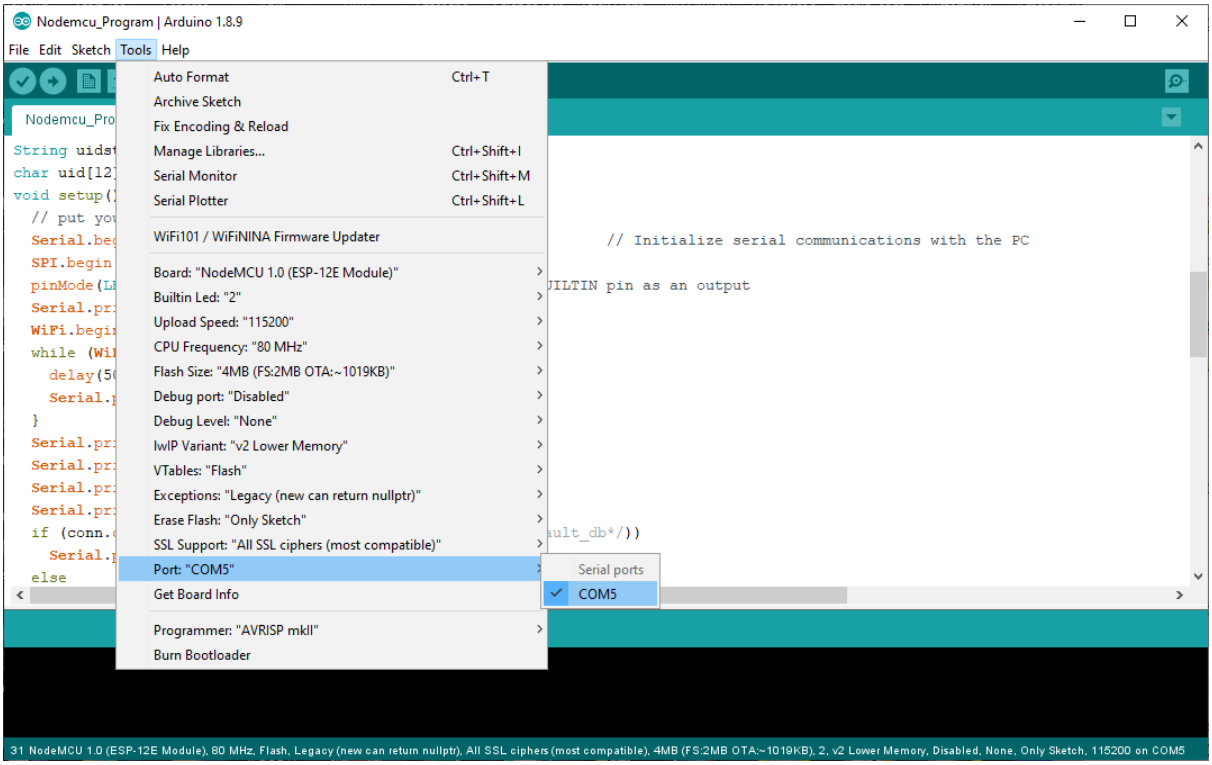
Open up the Nodemcu_Program.ino file.
The screen should look like this:



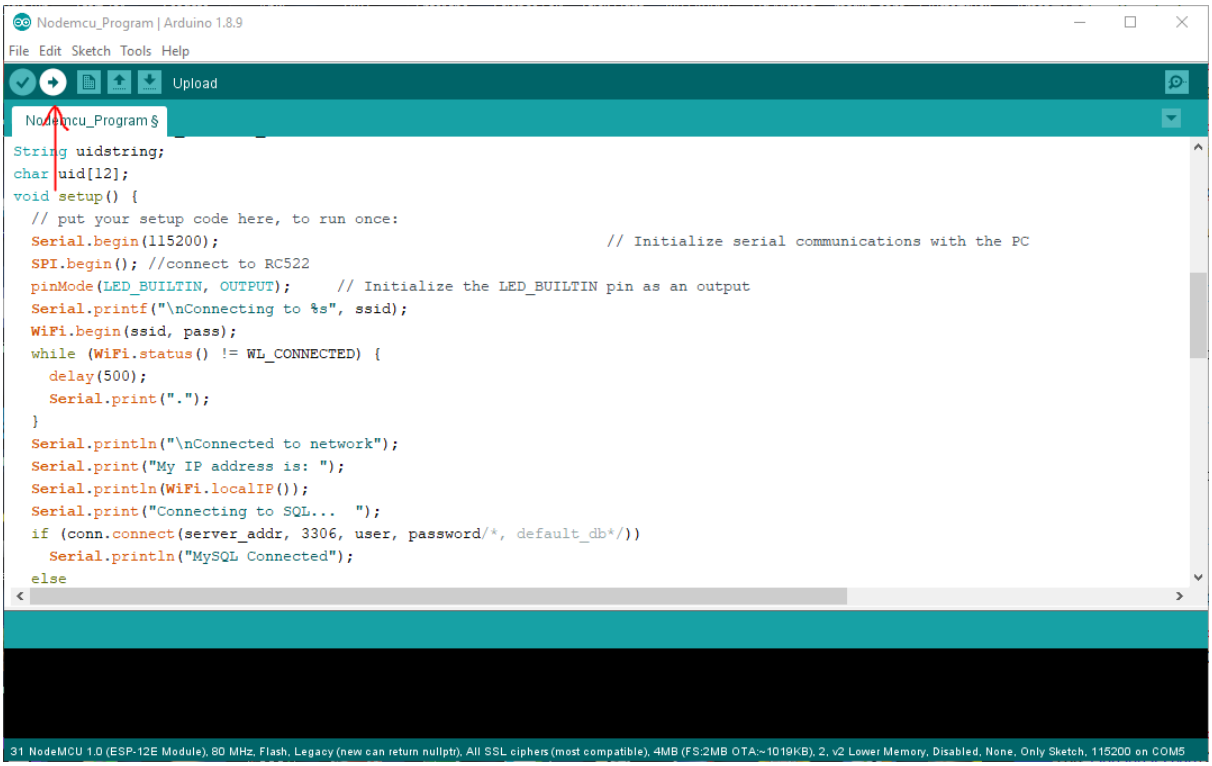
Make sure that the board selected is NodeMCU 1.0 (ESP-12E Module).



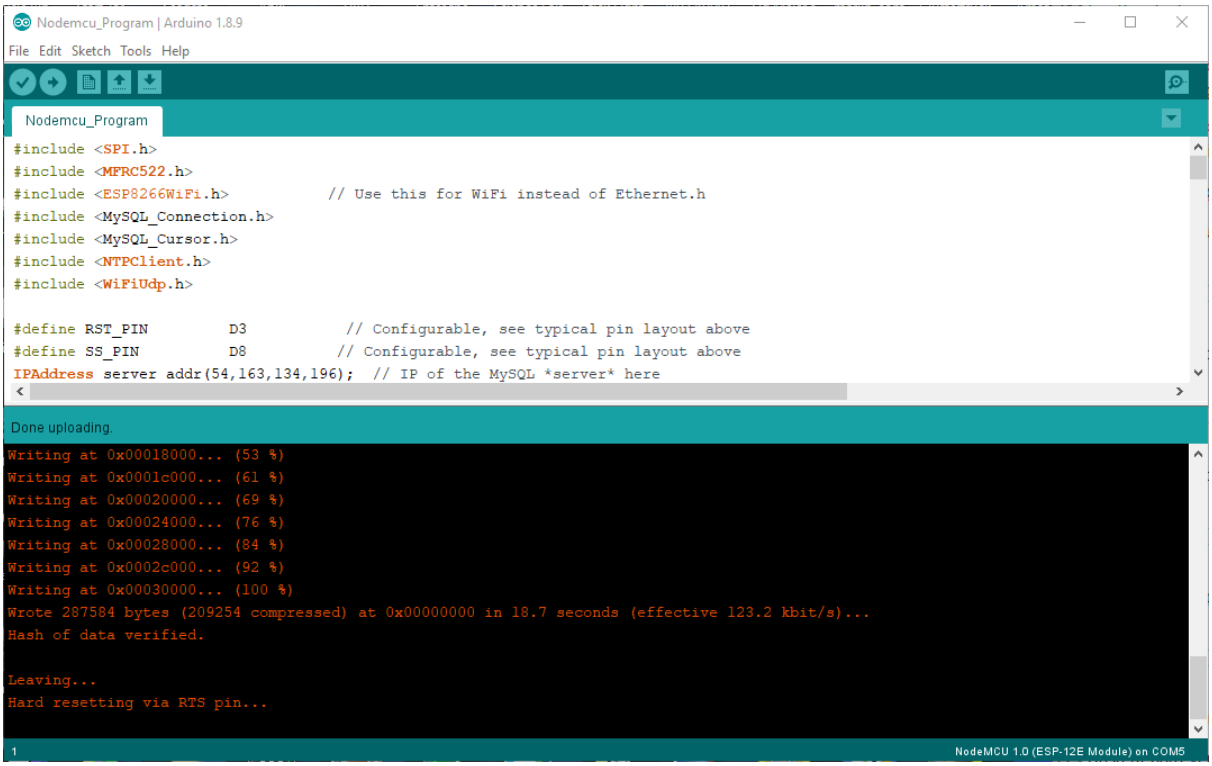
Make sure that the correct port is selected. This may take a couple tries to get the correct port.



Next, click the upload button in the top left corner. Before uploading the program, make sure that the WiFi, location, and MySQL are all set up.



If everything went successfully, the console should print "Wrote 287584 bytes (209254 compressed) at 0x00000000 in 18.7 seconds (effective 123.2 kbit/s)...
Hash of data verified." and look like:



4.4.1 Configuring WiFi

Insert the WiFi username and password into the two fields below:
Keep the quotation marks around the username and password.

```
1 char ssid[] = "WiFi Username";
2 char pass[] = "WiFi Password";
```

4.4.2 Configuring Location

Insert the RFID box's location into the field below:
Keep the quotation marks around the location.

```
1 char location[] = "location";
```

4.4.3 Configuring MySQL Database Connection

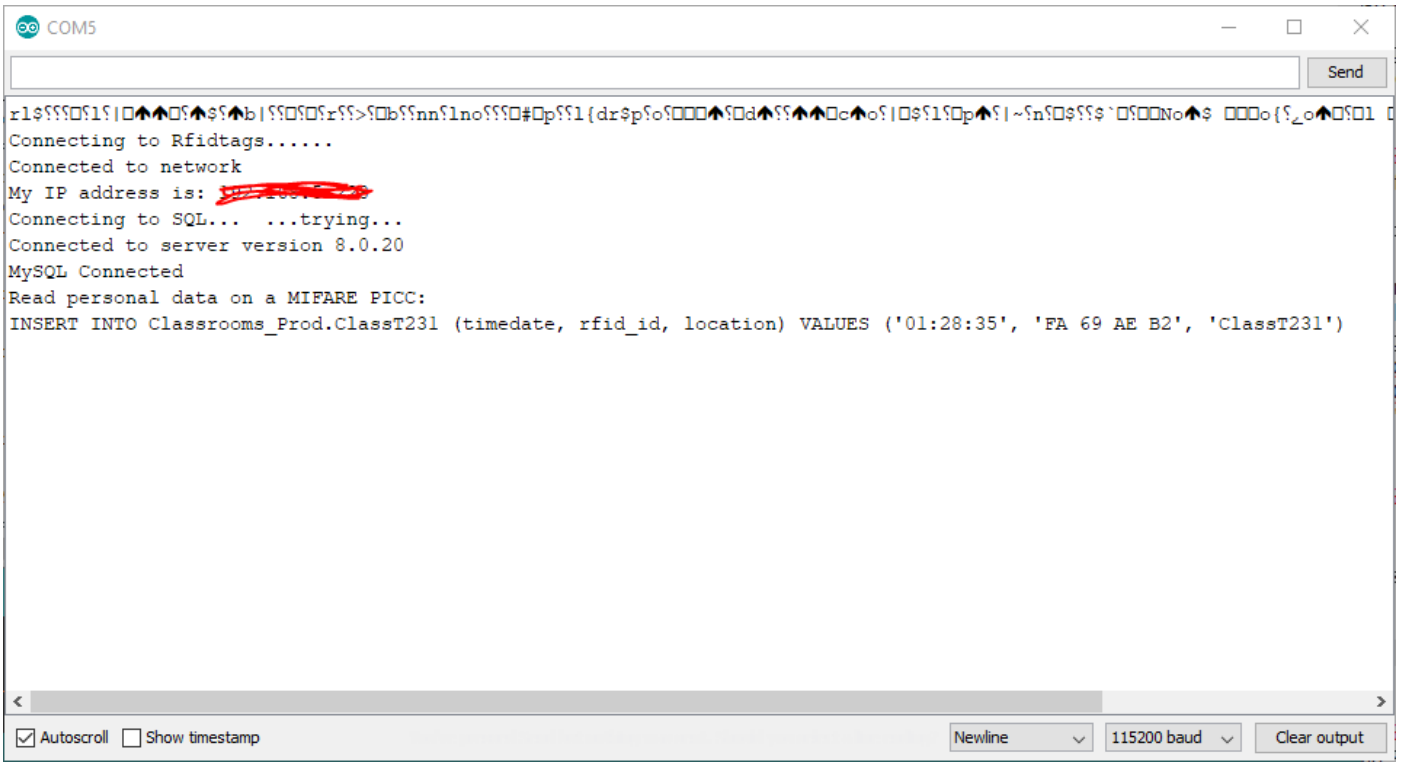
Insert the MySQL IP Address, username, and password into the three fields below:

Keep the quotation marks around the username and password. In the MySQL IP address exchange the periods for commas. Ex: 192,54,563,654 when the IP is 192.54.563.654.

```
IPAddress server_addr(IP Address);
char user[] = "MySQL Username";
char password[] = "MySQL Password'';
```

4.5 Testing Box

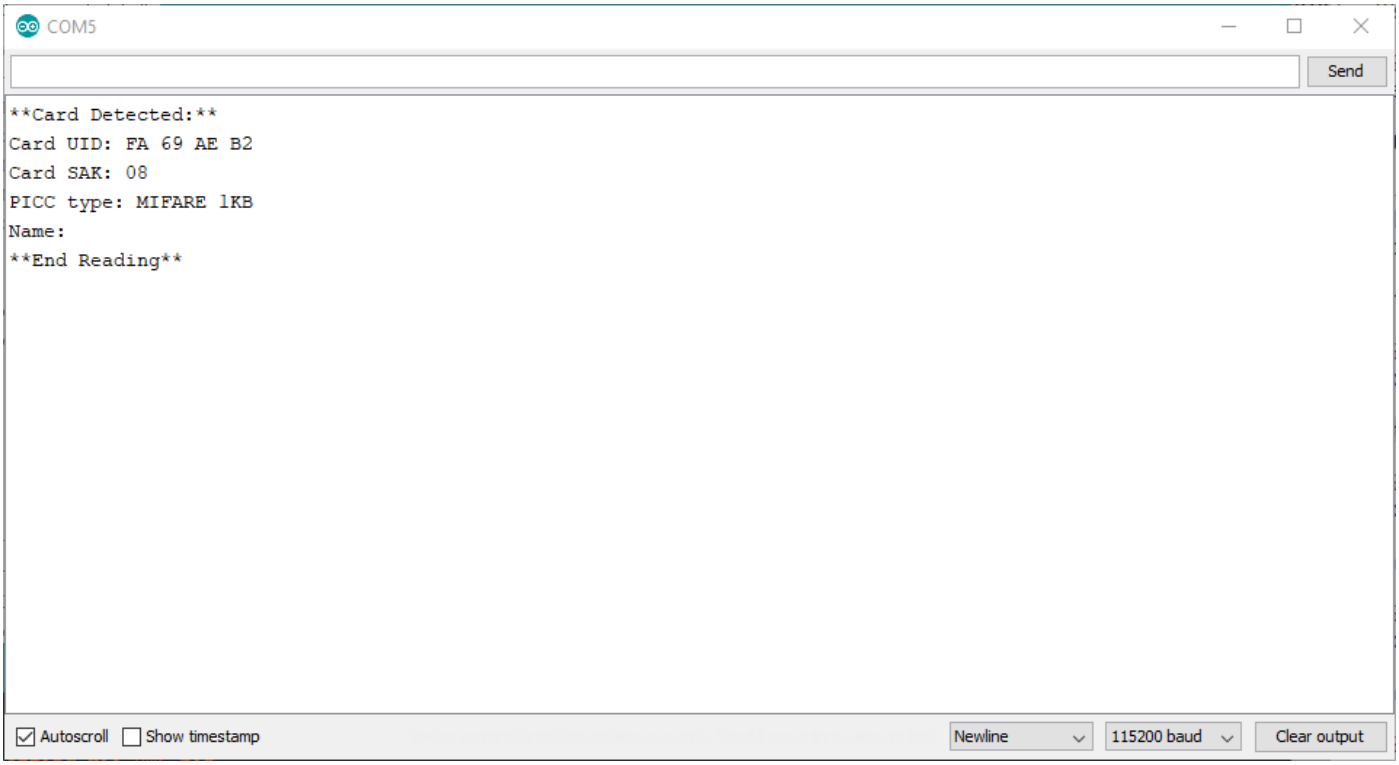
Once the program is uploaded, open the serial monitor. The following should appear:



If there is an error, repeat the process and ensure the WiFi, location, and MySQL are all correctly configured.

5 RFID Tag Setup

Open the rfid_read_personal_data.ino example file for the MRFC522 library. Load the program onto a box and scan a RFID tag.



The data after “Card UID: “ is the RFID tag ID. In this example the ID is “FA 69 AE B2”.

6 MySQL Database Setup

In MySQL Workbench run these commands:

- CREATE DATABASE Classrooms_Prod;
- USE Classrooms_Prod;

This creates a database on the MySQL server and selects it for further commands.

7 Adding a Location

Run these commands:

- USE Classrooms_Prod;
- CREATE TABLE “location name” (
- timedate VARCHAR(20) PRIMARY KEY,
- rfid_id TEXT NOT NULL,
- location TEXT NOT NULL

-)

Change “location name” with the name of the location. Do not include the “”. This should be the same location name that is in the Arduino program.

8 Remove a Location

Run these commands:

- DROP TABLE “location”

Change “location name” with the name of the location. Do not include the “”. This should be the same location name that is in the Arduino program.

9 Future of Project

This project is far from being finished. The items in this section would make the project more complete and a better system. In the future, the items in this section would be added to the system.

9.1 Redundancy

Having redundancy in a system like this is paramount. In a normal attendance system, the redundancy is to write student names down on a piece of paper. While this can be a final layer of redundancy for this system as well, it would be better to have a faster solution. Also, considering that this system has many pieces, there needs to be redundancy for each piece.

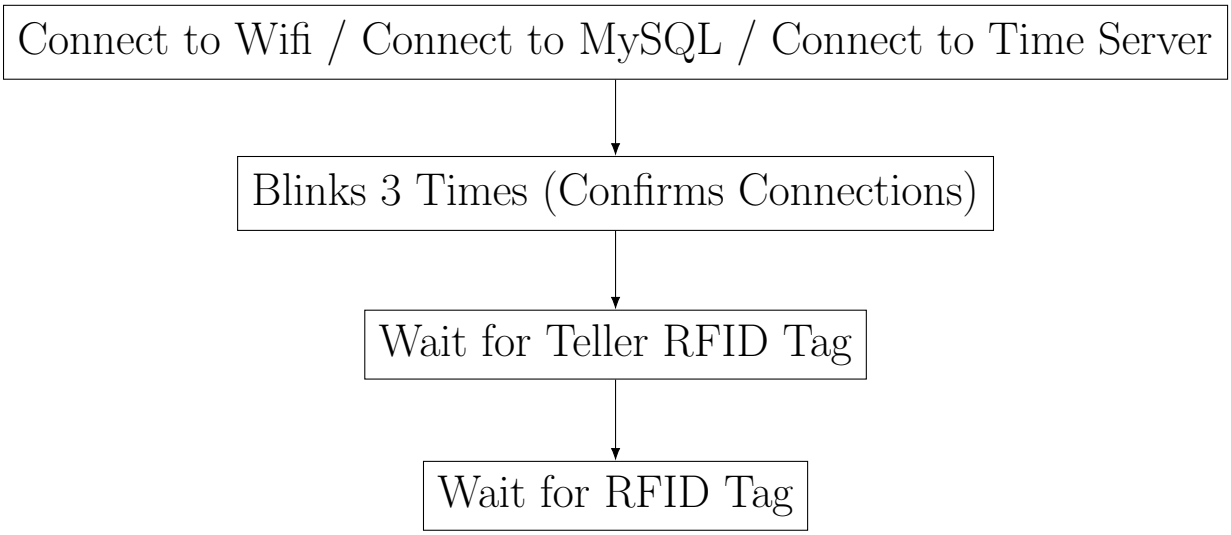
We need redundancy for these scenarios:

1. A student loses their RFID tag
2. A RFID box breaks
3. The MySQL database doesn’t respond
4. Attendance gets entered incorrectly

9.1.1 RFID Tag Backup

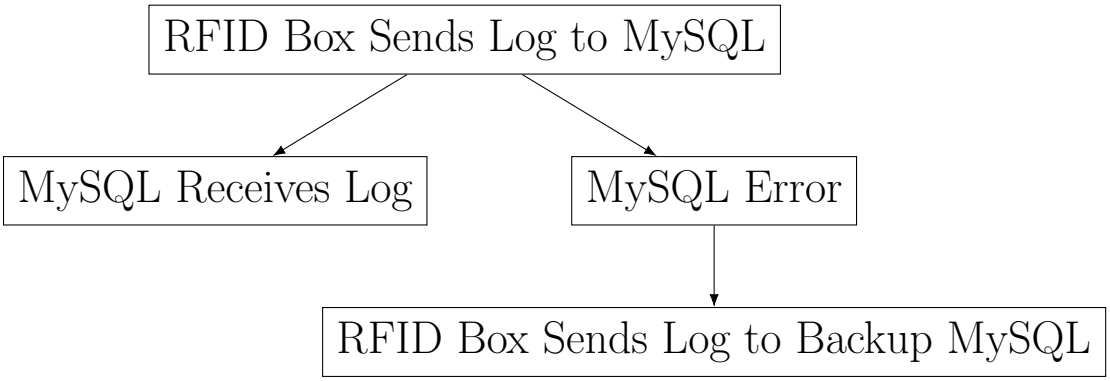
Students can easily lose their RFID tag if it falls off their lanyard or breaks. Students should have a person they can go to who can assign them a new RFID tag and update their tag ID in the system. The good news is that this is not a long or complicated process, just not implemented yet.

9.1.2 RFID Box Backup



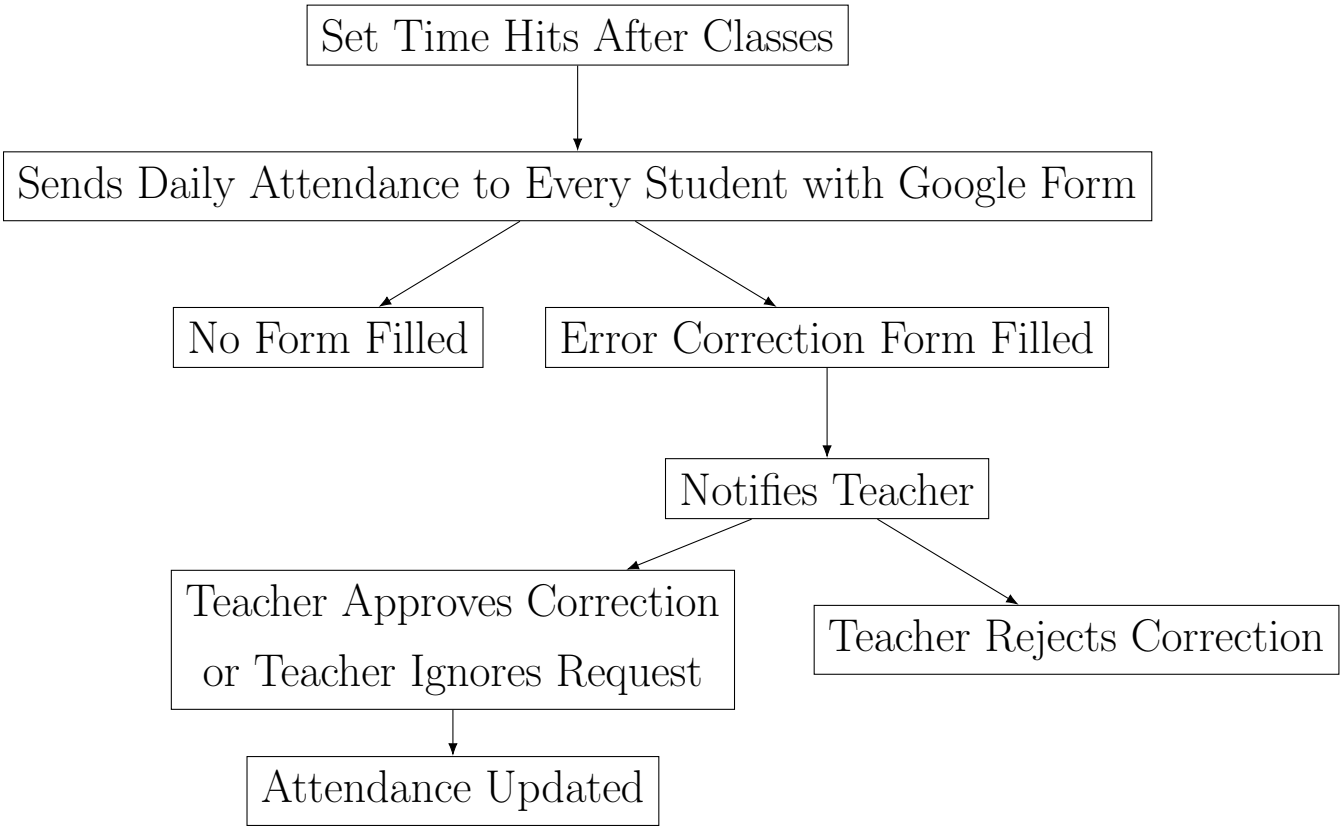
The addition to the project here is that during the setup phase (before the RFID box waits for a RFID Tag) the RFID box waits for a teller tag. This teller tag will have a location ID on it. The RFID box uses this location ID to set its current location. These backup RFID boxes would be in common areas near classrooms for easy access. The RFID teller tags would be left on a desk or on a wall hook in each location. Having backup RFID boxes ensures that if a box is to break, then there is a quick and easy backup minimizing downtime for the system.

9.1.3 MySQL Database Backup



The addition to the project here is that there is two possibilities after the RFID box tries to send a location log to the MySQL database. In the current iteration of this project, if a log fails to send, nothing happens. In this new iteration, if this happens, then the RFID box would send the log to a different MySQL database. Having this ability allows for the box to have a much lower chance of failing to send a location log.

9.1.4 Attendance Correction



The addition to the project here is that students have the ability to correct a wrong attendance log. For example, if two RFID boxes get mixed up somehow or a student's RFID tag fails to read, then the attendance will still be correct. If a student's attendance is wrong, they can fill out a google form. This form's data is parsed through and another email is sent to teachers asking them if the attendance correction is correct. There are two possibilities at this point. If the teacher approves the correction or ignores the email, then the attendance correction is assumed to be correct and the attendance logs are updated. On the other hand, if the teacher rejects the correction, then the attendance correction is assumed to be incorrect, and the attendance logs are not updated.

9.2 RFID Box Wall Holder

One problem with the current implementation of this project is that students might have troubles getting their RFID tags close enough to the RFID boxes to get a read. A solution to this problem would be having a wall holder for each RFID box.

There are two possible implementations of this solution:

Implementation 1 would be to have a command strip hook that would be stuck to the wall. The RFID boxes would have a hole in the back of them to hang on the command hooks. This is a good option for temporary setup since the command hooks can be taken down easily.

Implementation 2 would be to have a sleeve that is screwed into the wall. The RFID box would be inserted into this sleeve. This is the better option for long term implementation. One of the downsides of this is that since the RFID sleeves are screwed into the wall, it is hard to change the position of them.

The best solution would likely be a mix of both implementations with implementation two being used more for classrooms and building entrances where the positions are constant and implementation one being used for events where the locations are not always the same.

9.3 Injection Molded Case

Although having a 3D printed case is good for rapid prototyping, having the case be injection molded would have many benefits.

- Cheaper (in bulk)
- Faster to produce
- Less work needed (current box needs to have 3D printing support removed)
- Stronger
- Does not wear down as fast
- Stays clean longer (space between 3D printing lines can collect dust)

9.4 BlackBaud Integration

BlackBaud is the backend for many school systems. This project could interface with BlackBaud using BlackBaud's API. The location logs could be downloaded and parsed by python, which would send the data to BlackBaud.

9.5 Downloading Data at Night

At night, all of the data from the MySQL database could be downloaded. This will allow for there to be a backup of the data and allow for the data to be parsed.

9.6 Clips Inside for RFID Reader RC522

Currently, the RC522 board is aligned using pins in the model. However, it would be much nicer and last longer if clips were added to the model to secure the board down.

9.7 Automating Textbook Check-Out

If textbooks were each given a RFID sticker, students could quickly grab textbooks, scan them, then scan their ID and the system could automatically check out the textbook to the student. This would allow for the textbook checkout and return process to go much faster.

10 Conclusion

To conclude, this project has massive potential for speeding up everyday school processes and ensuring that students do not crowd together or share common items.
I personally feel that if this gets implemented at a school, the following effects could be achieved:

- Teachers would have more time to teach since they no longer need to take attendance.
- Student would get sick less due to not being as crowded and sharing less items.
- Processes like speed bumping and signing into events would go faster/smoother.

For the low cost of $546.61 the first year and $50.81 each subsequent year, implementing this system is very cost effective considering teachers spend 350 hours a year on attendance alone and many hours speed bumping and other processes where students declare their location.

Assuming this system is used for attendance since the time taken is easy to calculate the cost would be:

- First Year: $1.56 per hour saved
- Subsequent Years: $0.145 per hour saved