DormGuard

A Digital Hostel Attendance System

By

Kale Atharva Anna (202151069)

Mehatre Shubham Shivaji (202152323)

Department of CSE

Indian Institute Of Information Technology

Vadodara

TABLE OF CONTENTS

Introduction

In an era marked by technological advancements and the pursuit of efficiency, our project, titled 'DormGuard,' addresses a fundamental aspect of hostel management – attendance tracking. DormGuard is a cutting-edge digital hostel attendance system designed to streamline the process of monitoring students residing within the hostel premises.

The project's nomenclature, 'DormGuard,' succinctly encapsulates its primary function – safeguarding hostel attendance through digital means. By leveraging RFID technology, we have devised a system that digitally registers both the out-time and in-time of students, eliminating the need for manual data entry.

Traditional attendance systems in hostels involve a manual, offline process where personnel at entry/exit gates record student's details in a register. Recognizing the limitations of this approach, particularly the time-consuming nature of repetitive data entry, our project endeavors to revolutionize this process through digitalization.

The overarching objective of DormGuard is to digitally store and manage hostel attendance efficiently. By adopting RFID technology, we aim to achieve multiple goals, such as reducing the time students spend on repetitive data entry, minimizing queues at entry/exit points, and contributing to environmental sustainability by eliminating paper waste.

In the context of a rapidly modernizing world, where digitalization is reshaping various facets of daily life, DormGuard stands as a testament to innovation in hostel management. By discarding the conventional paper-based attendance system, DormGuard not only aligns with contemporary technological trends but also offers a more efficient, time-saving, and environmentally friendly alternative.

In the subsequent sections of this report, we delve into the methodology, benefits, and implementation of DormGuard, highlighting how this digital hostel attendance system addresses the shortcomings of traditional methods while ushering in a new era of efficiency and sustainability.

Literature Review

Various scholarly contributions have explored the utilization of RFID technology across diverse domains, with a specific focus on addressing challenges associated with academic attendance monitoring. In, the development of a secure and portable embedded reader system for extracting biometric data from electronic passports is detailed. This system employs Global System of Mobile Communications (GSM) network authentication to enhance reliability, security, and privacy. The communication data integrity between the identification center and the e-passport reader is safeguarded through AES encryption during transmission over the GSM network.

In, a comprehensive review of RFID applications, with an emphasis on supply chain management, is presented. The authors establish a taxonomic framework for literature classification, facilitating efficient content analysis to identify potential avenues for future research. Meanwhile, examines RFID's role in an integrated-circuit (IC) packaging house, specifically addressing inventory transaction issues and highlighting significant improvements in water receiving and inventory processes, resulting in reduced labor costs and human errors.

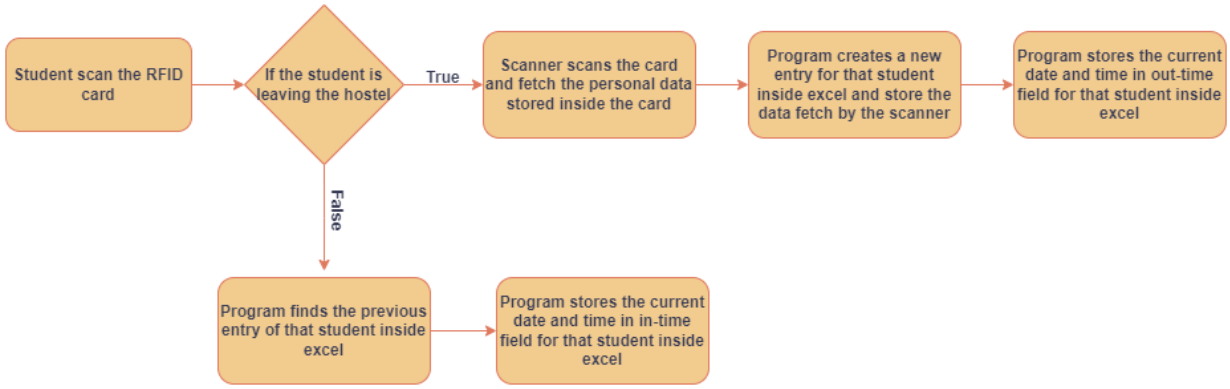
The implementation of an automated attendance management system is outlined in, covering both electronic and mobile platforms. The system employs AR 400 RFID readers and Symbol MC9000-G handheld RFID readers, showcasing a client/server architecture for classroom entrance monitoring. Notably, the electronic platform faces a limitation wherein RFID tag read rates degrade when in close proximity to electronic devices.

In an automatic attendance system utilizing fingerprint verification is proposed, relying on abnormal point extraction on the ridge of a user's fingerprint or minutiae technique. The system performs one-to-one comparisons for user authentication, signaling true or false attendance based on the logical result. introduces a biometric system employing fingerprint identification for employee attendance automation in an organizational setting.

Additionally, suggests a student wolf pack club tracking system for expediting the distribution of tickets for athletic events. The proposed model emphasizes simplicity, reliability, and cost-effectiveness, utilizing existing student ID card chips as passive tags with supplementary short message services providing weekly attendance summaries to parents.

Methodology

Block Diagram

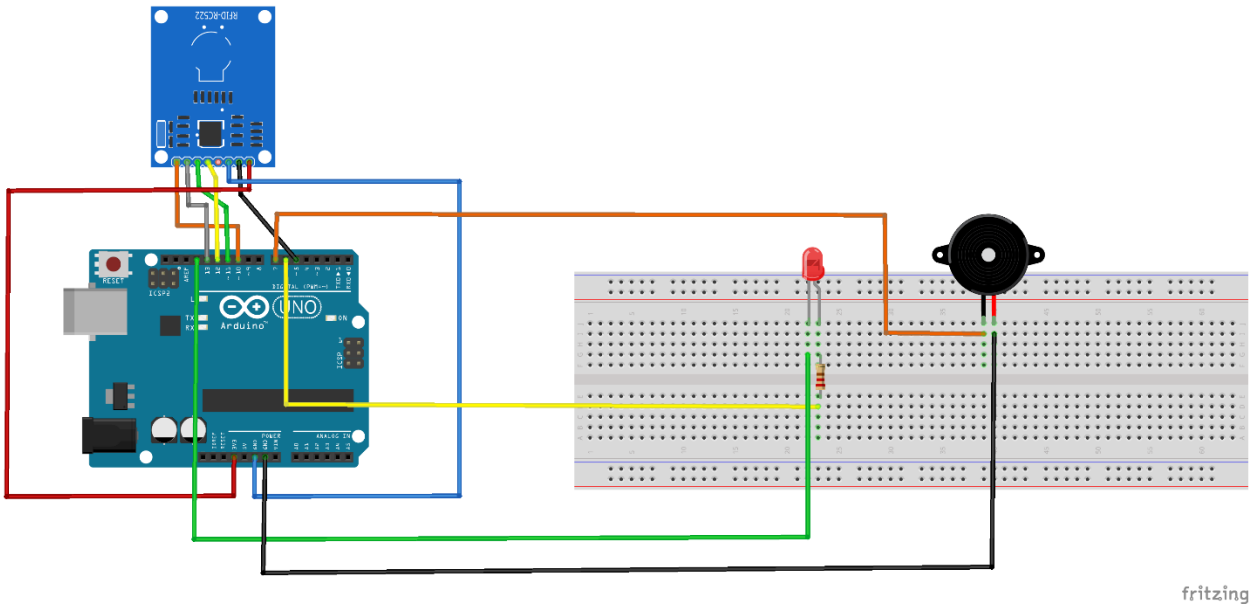


Description

The block diagram delineates the operational flow of our project. Initiated by a student scanning their RFID card at the RFID reader, the system dynamically responds based on the student's activity. When a student is exiting the hostel premises, the RFID scanner retrieves the comprehensive personal data stored in the RFID card. Subsequently, the program generates a new entry in the Excel sheet, populating the relevant columns with the fetched personal information. The system then captures the current date and time, recording it in the out-time column while leaving the in-time section blank.

Upon the student's return to the hostel, the program locates the pre-existing entry for that student and updates it with the current date and time in the in-time column. This streamlined process ensures accurate tracking of student movements.

Circuit Diagram



Description

The circuit configuration for our project involves the integration of several components to achieve efficient operation. The following components were employed in this circuit:

Arduino UNO: The central microcontroller serving as the core of the system.

RC522 RFID Card Reader Module: Facilitating RFID card authentication and communication.

Piezo Buzzer: Employed for audible feedback in the system.

LED: A Light Emitting Diode used for visual signaling.

Resistor (200 ohm): A resistor integrated into the circuit for current limitation.

Wires: Interconnecting elements to establish electrical connections.

RFID Cards and Tags: Utilized for the RFID-based identification process.

The interconnection of these components was meticulously carried out as follows:

Arduino UNO and RC522 RFID Card Reader Module Connections:

Establishing a seamless interface between the Arduino UNO and the RC522 RFID card reader module, the following pin connections were implemented:

RC522 RFID Card Reader Module	Arduino UNO
3.3 V pin	3.3 V pin
RST pin	5 number pin

GND pin	GND pin
MISO pin	12 number pin
MOSI pin	11 number pin
SCK pin	13 number pin
SDA pin	10 number pin

Buzzer and LED Connections:

Incorporating a breadboard into the circuit design, a 200-ohm resistor, LED, and buzzer were strategically positioned. The buzzer's Vcc (pin 1) was connected to pin 7 on the Arduino UNO, and the GND (pin 2) was linked to the Arduino UNO's GND. The LED's anode was connected to one end of the resistor, and its cathode was connected to the Arduino UNO's GND. The resistor's other end was connected to pin 6 on the Arduino UNO.

This systematic arrangement ensures the effective integration of each component, facilitating the desired functionality of the overall circuit.

Logic

Upon enrollment in the hostel, a new student is issued a blank RFID card devoid of any stored data. Subsequently, an authorized figure, such as the hostel warden, initializes the card by inputting essential personal data like student ID, name, phone number, and room number. This pivotal step renders the student eligible to utilize the RFID card for attendance tracking upon exiting or returning to the hostel.

Consider a scenario where a registered student departs from the hostel. Upon RFID card scanning, the system initiates a program to scrutinize the Excel sheet for the student's latest entry. Should the fields of this entry be entirely populated, the program triggers the creation of a new entry. Subsequently, all pertinent personal data is transcribed into this fresh entry, and concurrently, the current date and time are logged in the out-time column.

Upon the student's return to the hostel, a subsequent RFID card scan prompts the program to seek the nearest entry once again. If the in-time field of the identified entry remains unpopulated, the program proceeds to record the current date and time in this field, thus concluding the attendance tracking cycle. This intricate logic ensures the systematic and accurate recording of student movements, epitomizing the technical sophistication underlying the project's functionality.

Codes

Read-Write-Reset-RFID

```
#include <SPI.h>
#include <MFRC522.h>

/*Using Hardware SPI of Arduino */
/*MOSI (11), MISO (12) and SCK (13) are fixed */
/*You can configure SS and RST Pins*/
#define SS_PIN 10  /* Slave Select Pin */
#define RST_PIN 7  /* Reset Pin */

MFRC522 mfrc522(SS_PIN, RST_PIN); /* Create an instance of MFRC522 */
MFRC522::MIFARE_Key key; /* Create an instance of MIFARE_Key */

/* Set the block to which we want to write data */
/* Be aware of Sector Trailer Blocks */
int blockNum_ID = 4; // 4th block is used to store the Student ID
int blockNum_Name = 5; // 5th block is used to store the Name of Student
int blockNum_Phone = 6; //6th block is used to store the phone number of the
student
int blockNum_Room = 8; // 8th block is used to store the romm number of the
student

// byte blockData_ID [16] = {"202151069"}; //Creating an 16 bytes of array to
store Student ID
// byte blockData_Name [16] = {"Atharva Kale"}; //Creating an 16 bytes of array
to store Name of the student
// byte blockData_Phone [16] = {"9867915375"}; //Creating an 16 bytes of array to
store Phone number of student
// byte blockData_Room [16] = {"E-704"}; //Creating an 16 bytes of array to store
Room number of student
/* This is the actual data which is going to be written into the card */

//Inorder to reset the data stored inside the RFID Card, Comment the above four
Lines and Uncomment the below code
byte blockData_ID [16] = {0};
byte blockData_Name [16] = {0};
byte blockData_Phone [16] = {0};
byte blockData_Room [16] = {0};
```

```
/* Create another arrays to read data from Block */
/* Legthn of buffer should be 2 Bytes more than the size of Block (16 Bytes) */
byte bufferLen = 18;
byte readBlockData_ID[18];
byte readBlockData_Name[18];
byte readBlockData_Phone[18];
byte readBlockData_Room[18];

MFRC522::StatusCode status;

void setup()
{
  /* Initialize serial communications with the PC */
  Serial.begin(9600);
  /* Initialize SPI bus */
  SPI.begin();
  /* Initialize MFRC522 Module */
  mfrc522.PCD_Init();
  Serial.println("Scan a MIFARE 1K Tag to write data...");
}

void loop()
{

  /* Prepare the ksy for authentication */
  /* All keys are set to FFFFFFFFFFFFh at chip delivery from the factory */
  for (byte i = 0; i < 6; i++)
  {
    key.keyByte[i] = 0xFF;
  }
  /* Look for new cards */
  /* Reset the loop if no new card is present on RC522 Reader */
  if ( ! mfrc522.PICC_IsNewCardPresent())
  {
    return;
  }

  /* Select one of the cards */
  if ( ! mfrc522.PICC_ReadCardSerial())
  {
    return;
  }
  Serial.print("\n");
```

```
  Serial.println("**Card Detected**");
  /* Print UID of the Card */
  Serial.print(F("Card UID:"));
  for (byte i = 0; i < mfrc522.uid.size; i++)
  {
    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
  Serial.print("\n");
  /* Print type of card (for example, MIFARE 1K) */
  Serial.print(F("PICC type: "));
  MFRC522::PICC_Type piccType = mfrc522.PICC_GetType(mfrc522.uid.sak);
  Serial.println(mfrc522.PICC_GetTypeName(piccType));

   /* Call 'WriteDataToBlock' function, which will write data to the block */
   Serial.print("\n");
   Serial.println("Writing to Data Block...");
   WriteDataToBlock(blockNum_ID, blockData_ID);
   WriteDataToBlock(blockNum_Name, blockData_Name);
   WriteDataToBlock(blockNum_Phone, blockData_Phone);
   WriteDataToBlock(blockNum_Room, blockData_Room);

   /* Read data from the same block */
   Serial.print("\n");
   Serial.println("Reading from Data Block...");
   ReadDataFromBlock(blockNum_ID, readBlockData_ID);
   ReadDataFromBlock(blockNum_Name, readBlockData_Name);
   ReadDataFromBlock(blockNum_Phone, readBlockData_Phone);
   ReadDataFromBlock(blockNum_Room, readBlockData_Room);
   /* If you want to print the full memory dump, uncomment the next line */
   //mfrc522.PICC_DumpToSerial(&(mfrc522.uid));

   /* Print the data read from block */
   Serial.print("\n");
   Serial.print("Data in Block:");
   Serial.print(blockNum_ID);
   Serial.print(" --> ");
   for (int j=0 ; j<16 ; j++)
   {
     Serial.write(readBlockData_ID[j]);
   }
   Serial.print("\n");

   Serial.print("\n");
   Serial.print("Data in Block:");
```

```
   Serial.print(blockNum_Name);
   Serial.print(" --> ");
   for (int j=0 ; j<16 ; j++)
   {
     Serial.write(readBlockData_Name[j]);
   }
   Serial.print("\n");

   Serial.print("\n");
   Serial.print("Data in Block:");
   Serial.print(blockNum_Phone);
   Serial.print(" --> ");
   for (int j=0 ; j<16 ; j++)
   {
     Serial.write(readBlockData_Phone[j]);
   }
   Serial.print("\n");

   Serial.print("\n");
   Serial.print("Data in Block:");
   Serial.print(blockNum_Room);
   Serial.print(" --> ");
   for (int j=0 ; j<16 ; j++)
   {
     Serial.write(readBlockData_Room[j]);
   }
   Serial.print("\n");
}



void WriteDataToBlock(int blockNum, byte blockData[])
{
  /* Authenticating the desired data block for write access using Key A */
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNum,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK)
  {
    Serial.print("Authentication failed for Write: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  else
  {
    Serial.println("Authentication success");
```

```
  }


  /* Write data to the block */
  status = mfrc522.MIFARE_Write(blockNum, blockData, 16);
  if (status != MFRC522::STATUS_OK)
  {
    Serial.print("Writing to Block failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  else
  {
    Serial.println("Data was written into Block successfully");
  }

}

void ReadDataFromBlock(int blockNum, byte readBlockData[])
{
  /* Authenticating the desired data block for Read access using Key A */
  byte status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A,
blockNum, &key, &(mfrc522.uid));

  if (status != MFRC522::STATUS_OK)
  {
     Serial.print("Authentication failed for Read: ");
     Serial.println(mfrc522.GetStatusCodeName(status));
     return;
  }
  else
  {
    Serial.println("Authentication success");
  }

  /* Reading data from the Block */
  status = mfrc522.MIFARE_Read(blockNum, readBlockData, &bufferLen);
  if (status != MFRC522::STATUS_OK)
  {
    Serial.print("Reading failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }
  else
  {
```

```
    Serial.println("Block was read successfully");
  }

}
```

Description of the code:

This Arduino code is tailored for interfacing with an RFID card reader module (MFRC522) to read and write data on MIFARE RFID cards. The code initializes the RFID module, establishes connections to specific pins on the Arduino UNO, and defines the block numbers for storing various pieces of student data, including ID, name, phone number, and room number.

The RFID card contains four data blocks (blockNum_ID, blockNum_Name, blockNum_Phone, and blockNum_Room), each 16 bytes in size, where different aspects of student information are stored.

The program utilizes the MFRC522 library to interact with the RFID module. It establishes communication with the RFID card, authenticates the card using Key A, and writes or reads data from the specified data blocks accordingly.

The setup() function initializes serial communication with the computer, sets up the SPI bus and the MFRC522 module, and prints an initial message to prompt the user to scan an RFID card.

The loop() function continuously checks for the presence of a new RFID card. When a card is detected, it reads the UID and type of the card and proceeds to write and read data from the specified data blocks. The code includes comments to explain each section, enhancing its readability.

Additionally, this code incorporates functionality to reset data stored in the RFID card. By commenting out the lines initializing blockData_ID, blockData_Name, blockData_Phone, and blockData_Room, and uncommenting the lines initializing these arrays with zeros, the user can reset the data inside the RFID card. This feature adds flexibility to the system, allowing for the erasure of existing data when needed.

RFID – Logger

```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9
MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance.
MFRC522::MIFARE_Key key;

int readsuccess;
byte readcard[4];
char str[32] = "";
String StrUID;
String StudentID;
String Name;
String Phone;
String Room;

// Buzzer and LED setup
int buzzerPin = 7;  // Connect the buzzer to pin 7
int ledPin = 6;    // Connect the LED to pin 13

void setup() {
  Serial.begin(9600); // Initialize serial communications with the PC
  SPI.begin();      // Init SPI bus
  mfrc522.PCD_Init(); // Init MFRC522 card

  pinMode(buzzerPin, OUTPUT);
  pinMode(ledPin, OUTPUT);

  delay(1000);
}

void loop() {
  while (true) {

    mfrc522.PCD_Init(); // Re-initialize the RFID reader

    for (byte i = 0; i < 6; i++) {
      key.keyByte[i] = 0xFF;
    }
```

```
    if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
      // Function to activate Buzzer and LED
      activateBuzzerAndLED();

      for (int i = 0; i < 4; i++) {
        readcard[i] = mfrc522.uid.uidByte[i]; // Storing the UID of the tag in
readcard
        array_to_string(readcard, 4, str);
        StrUID = str;
      }

      // Read data from specific blocks (4, 5, 6, 8)
      StudentID = readBlock(4);
      Name = readBlock(5);
      Phone = readBlock(6);
      Room = readBlock(8);

      // Print RFID data to serial monitor
      Serial.println(StrUID + "," + StudentID + "," + Name + "," + Phone + "," +
Room);

      // Halt the card and add a delay after successfully reading a card
      mfrc522.PICC_HaltA();
      delay(3000);
    }
  }
}

// Function to activate Buzzer and LED
void activateBuzzerAndLED() {
  tone(buzzerPin, 3200);  // Turn on the buzzer
  digitalWrite(ledPin, HIGH);     // Turn on the LED
  delay(400);  // Keep them on for 1 second
  noTone(buzzerPin);   // Turn off the buzzer
  digitalWrite(ledPin, LOW);      // Turn off the LED
}

// Function to get RFID card ID and read associated data
int getid() {
  if(!mfrc522.PICC_IsNewCardPresent()){
    return 0;
  }
  if(!mfrc522.PICC_ReadCardSerial()){
    return 0;
```

```
  }

  for(int i=0; i<4; i++){
    readcard[i] = mfrc522.uid.uidByte[i]; // Storing the UID of the tag in
readcard
    array_to_string(readcard, 4, str);
    StrUID = str;
  }

  // Read data from specific blocks (4, 5, 6, 8)
  StudentID = readBlock(4);
  Name = readBlock(5);
  Phone = readBlock(6);
  Room = readBlock(8);

  mfrc522.PICC_HaltA();
  return 1;
}

// Function to read data from a specific block
String readBlock(int blockNumber) {
  MFRC522::StatusCode status;
  byte buffer[18];
  byte size = sizeof(buffer);

  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNumber,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    return "";
  }

  status = mfrc522.MIFARE_Read(blockNumber, buffer, &size);
  if (status == MFRC522::STATUS_OK) {
    String data = "";
    for (byte i = 0; i < 16; i++) {
      char c = (char)buffer[i];
      if (c != '\0') {
        data += c;
      }
    }
    return data;
  } else {
    return "";
  }
}
```

```
// Function to convert byte array to string
void array_to_string(byte array[], unsigned int len, char buffer[]) {
  for (unsigned int i = 0; i < len; i++) {
    byte nib1 = (array[i] >> 4) & 0x0F;
    byte nib2 = (array[i] >> 0) & 0x0F;
    buffer[i*2+0] = nib1  < 0xA ? '0' + nib1  : 'A' + nib1  - 0xA;
    buffer[i*2+1] = nib2  < 0xA ? '0' + nib2  : 'A' + nib2  - 0xA;
  }
  buffer[len*2] = '\0';
}
```

Description of the code:

This Arduino code interfaces with an RFID card reader module (MFRC522) to read data from specific blocks on MIFARE RFID cards. It also includes functionality to activate a buzzer and LED upon successful card detection. Here's a description of the code:

The code begins by initializing the RFID module, defining pins, and setting up the serial communication. It configures pins for the buzzer and LED, initializing the RFID reader, and introducing a delay for stability.

The main loop continuously checks for a new RFID card. Upon detection, the activateBuzzerAndLED() function is invoked to provide a visual and auditory indication. The UID of the card is then read and converted into a string (StrUID).

The readBlock() function is utilized to read data from specific blocks (4, 5, 6, 8) on the RFID card, corresponding to Student ID, Name, Phone, and Room. The data is stored in String variables (StudentID, Name, Phone, Room).

The Python script created by us interfaces with an Arduino through serial communication, reading RFID data and updating a CSV/xslx file with relevant information.

The HaltA() function is called to halt the RFID card, and a delay is introduced to prevent rapid and unintended reads.

The activateBuzzerAndLED() function activates the buzzer and LED for a brief period to provide a visual and auditory indication of a successful card read.

The getid() function provides an alternative method to read the RFID card ID and associated data. It returns 1 upon successful read and 0 otherwise.

The readBlock() function uses the PCD_Authenticate() and MIFARE_Read() functions to read data from a specified block. The data is then converted to a string and returned.

The array_to_string() function converts a byte array to a string for UID processing.

In summary, this code offers a comprehensive RFID card reading and data retrieval system with visual and auditory feedback, making it suitable for applications such as attendance tracking or access control.

Store data in CSV

```
import os
import serial
import pandas as pd
from datetime import datetime

# Serial port settings
ser = serial.Serial('COM9', 9600)  # Adjust COM2 to your Arduino's serial port

# CSV file settings
csv_file_path = 'data.csv'

def update_csv(user_id, current_time, student_id, name, phone, room):
    try:
        # Check if the CSV file exists, create an empty DataFrame if it doesn't
        if not os.path.exists(csv_file_path):
            df = pd.DataFrame(columns=['User_ID', 'Student_ID', 'Name', 'Phone
No.', 'Room No.', 'OutTime', 'InTime'])
        else:
            # Read existing data
            df = pd.read_csv(csv_file_path)

        # Check if the user is already present
        user_exists = df['User_ID'].eq(user_id).any()

        if user_exists:
            # User exists, find the last index for the given user_id
            user_index = df.index[df['User_ID'] == user_id].max()

            if pd.isnull(df.at[user_index, 'InTime']):
                # OutTime is empty, update OutTime using loc
                df.loc[user_index, 'InTime'] = current_time
            else:
                # Both InTime and OutTime are present, add a new entry
                new_entry = {'User_ID': user_id, 'Student_ID': student_id,
'Name': name, 'Phone No.': phone,
                             'Room No.': room, 'OutTime': current_time, 'InTime':
None}
                df = pd.concat([df, pd.DataFrame([new_entry])],
ignore_index=True)
        else:
```

```
            # User doesn't exist, add a new entry
            new_entry = {'User_ID': user_id, 'Student_ID': student_id, 'Name':
name, 'Phone No.': phone,
                         'Room No.': room, 'OutTime': current_time, 'InTime':
None}
            df = pd.concat([df, pd.DataFrame([new_entry])], ignore_index=True)

        # Save the updated DataFrame to CSV
        df.to_csv(csv_file_path, index=False)

    except Exception as e:
        print(f"Error updating CSV file: {e}")

# Continuous loop for reading RFID data from Arduino
try:
    while True:
        # Read RFID data from Arduino
        rfid_data = ser.readline().decode().strip()

        # Print the received RFID data for debugging
        print("Received RFID data:", rfid_data)

        # Extract data from the received RFID data
        data_parts = rfid_data.split(',')
        if len(data_parts) == 5:
            user_id, student_id, name, phone, room = map(str.strip, data_parts)

            # Convert timestamp to datetime
            current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

            # Update CSV file with the specified conditions
            update_csv(user_id, current_time, student_id, name, phone, room)

except KeyboardInterrupt:
    ser.close()
    print("Serial port closed.")
except Exception as e:
    print(f"An error occurred:")
    ser.close()
```

Store data in Excel

```
import os
import serial
import pandas as pd
from datetime import datetime

# Serial port settings
ser = serial.Serial('COM9', 9600)  # Adjust COM9 to your Arduino's serial port

# Excel file settings
excel_file_path = 'data.xlsx'

def update_excel(user_id, current_time, student_id, name, phone, room):
    try:
        # Check if the Excel file exists, create an empty DataFrame if it doesn't
        if not os.path.exists(excel_file_path):
            df = pd.DataFrame(columns=['User_ID', 'Student_ID', 'Name', 'Phone
No.', 'Room No.', 'OutTime', 'InTime'])
        else:
            # Read existing data
            df = pd.read_excel(excel_file_path)

        # Check if the user is already present
        user_exists = df['User_ID'].eq(user_id).any()

        if user_exists:
            # User exists, find the last index for the given user_id
            user_index = df.index[df['User_ID'] == user_id].max()

            if pd.isnull(df.at[user_index, 'InTime']):
                # OutTime is empty, update OutTime using loc
                df.loc[user_index, 'InTime'] = pd.to_datetime(current_time)  #
Explicitly cast to datetime64
            else:
                # Both InTime and OutTime are present, add a new entry
                new_entry = {'User_ID': user_id, 'Student_ID': student_id,
'Name': name, 'Phone No.': phone,
                             'Room No.': room, 'OutTime':
pd.to_datetime(current_time), 'InTime': None}  # Explicitly cast to datetime64
                df = pd.concat([df, pd.DataFrame([new_entry])],
ignore_index=True)
        else:
            # User doesn't exist, add a new entry
```

```
            new_entry = {'User_ID': user_id, 'Student_ID': student_id, 'Name':
name, 'Phone No.': phone,
                         'Room No.': room, 'OutTime':
pd.to_datetime(current_time), 'InTime': None}  # Explicitly cast to datetime64
            df = pd.concat([df, pd.DataFrame([new_entry])], ignore_index=True)

        # Save the updated DataFrame to Excel
        df.to_excel(excel_file_path, index=False)

    except Exception as e:
        print(f"Error updating Excel file: {e}")

# Continuous loop for reading RFID data from Arduino
try:
    while True:
        # Read RFID data from Arduino
        rfid_data = ser.readline().decode().strip()

        # Print the received RFID data for debugging
        print("Received RFID data:", rfid_data)

        # Extract data from the received RFID data
        data_parts = rfid_data.split(',')
        if len(data_parts) == 5:
            user_id, student_id, name, phone, room = map(str.strip, data_parts)

            # Convert timestamp to datetime
            current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

            # Update Excel file with the specified conditions
            update_excel(user_id, current_time, student_id, name, phone, room)

except KeyboardInterrupt:
    ser.close()
    print("Serial port closed.")
except Exception as e:
    print(f"An error occurred:")
    ser.close()
```

Description of the code:

Both the code provided above are Python script for reading RFID data from an Arduino through a serial port and updating either a CSV file or an Excel file with the received data. The script uses the `serial` library to communicate with the Arduino and the `pandas` library to handle and manipulate data in either a CSV file or an Excel file.

Here's a breakdown of the key components of the script:

Serial Communication

- The script uses the `serial` library to establish a connection with the Arduino through a specified serial port (`COM2` in this case) at a baud rate of 9600.

File Handling

- The script defines either a CSV file (`data.csv`) or an Excel file (`data.xlsx`) to store the RFID data.

Function for Updating File

- There is a function named `update_csv` or `update_excel`, depending on the version of the script, which takes RFID data and updates the CSV or Excel file based on certain conditions.

Continuous Loop

- The script enters a continuous loop to read RFID data from the Arduino. It prints the received RFID data for debugging purposes.
- The received RFID data is then parsed into individual components (user_id, student_id, name, phone, room).
- The script then calls the respective update function (`update_csv` or `update_excel`) to update the file with the parsed data.

Error Handling

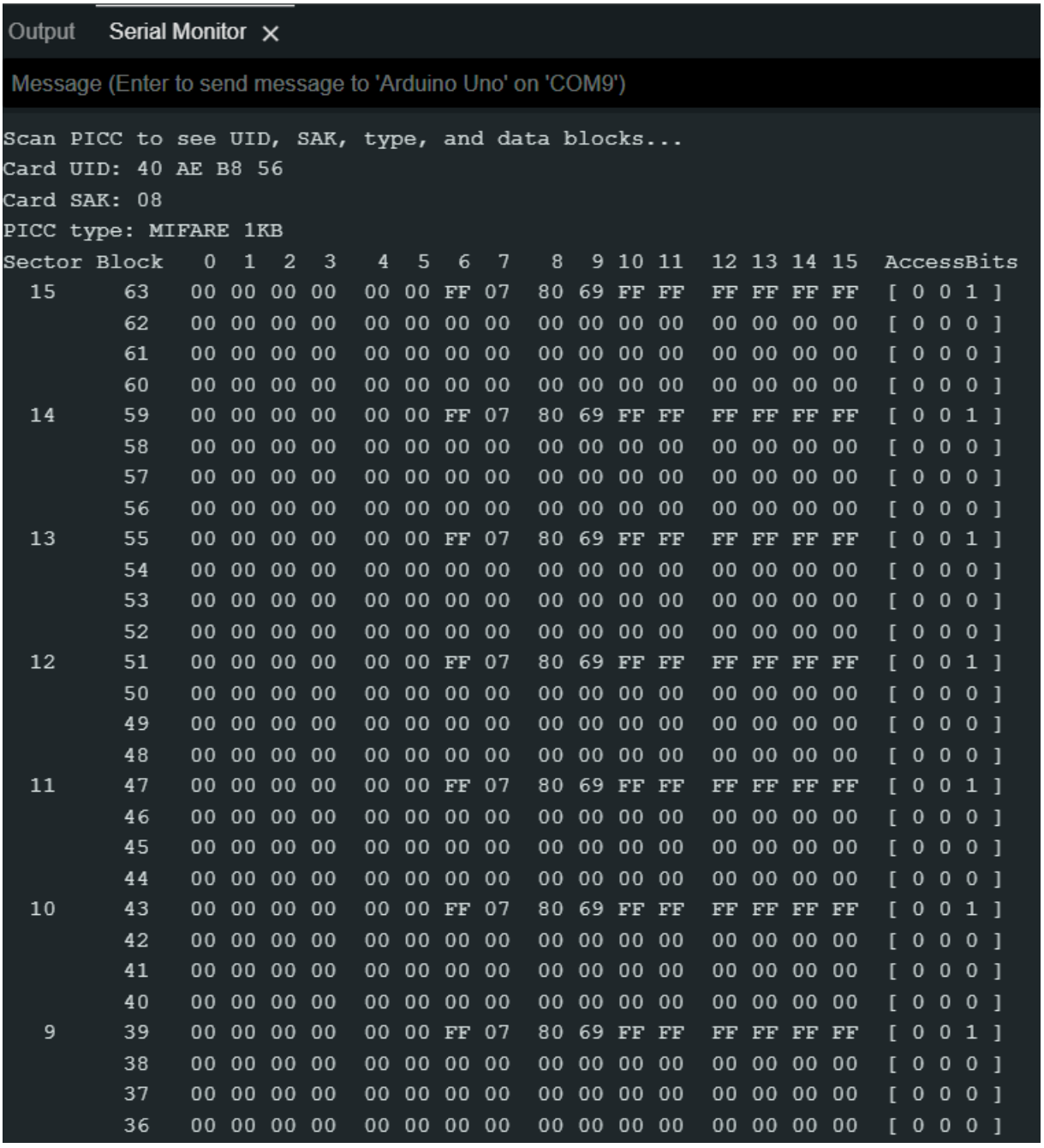
- The script includes exception handling to catch any errors that may occur during file updates or serial communication.
- If a KeyboardInterrupt (e.g., user pressing Ctrl+C) is detected, the serial port is closed, and a message is printed.

Timestamp Handling

The current timestamp (`current_time`) is generated using the `datetime` module and is included in the file update.

Results

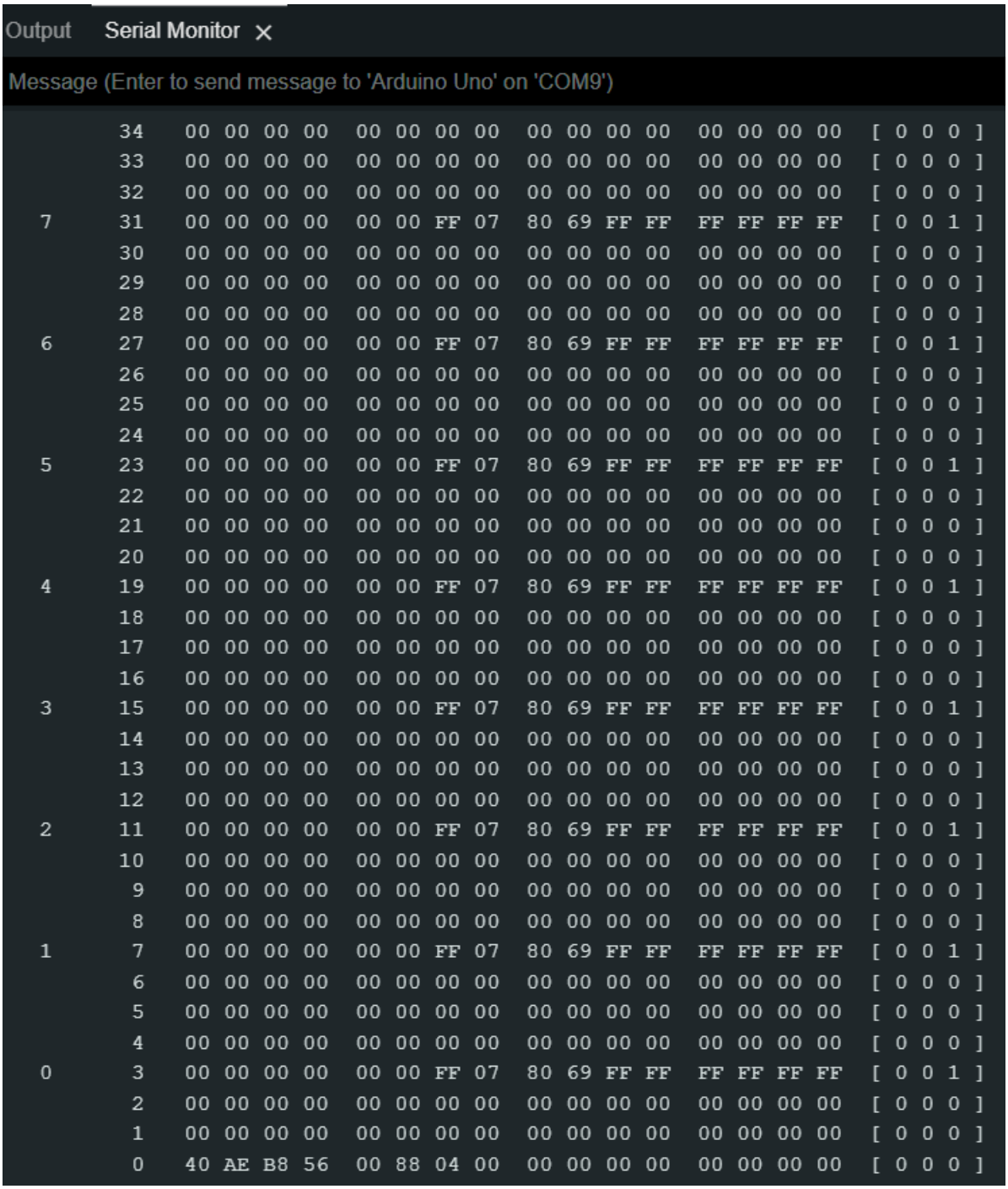
Memory format of a RFID card.



Output Serial Monitor ×

Message (Enter to send message to 'Arduino Uno' on 'COM9')

```
Scan PICC to see UID, SAK, type, and data blocks...
Card UID: 40 AE B8 56
Card SAK: 08
PICC type: MIFARE 1KB
Sector Block   0  1  2  3   4  5  6  7   8  9 10 11  12 13 14 15  AccessBits
  15      63   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
          62   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          61   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          60   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
  14      59   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
          58   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          57   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          56   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
  13      55   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
          54   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          53   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          52   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
  12      51   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
          50   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          49   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          48   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
  11      47   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
          46   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          45   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          44   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
  10      43   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
          42   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          41   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          40   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
   9      39   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
          38   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          37   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
          36   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
```

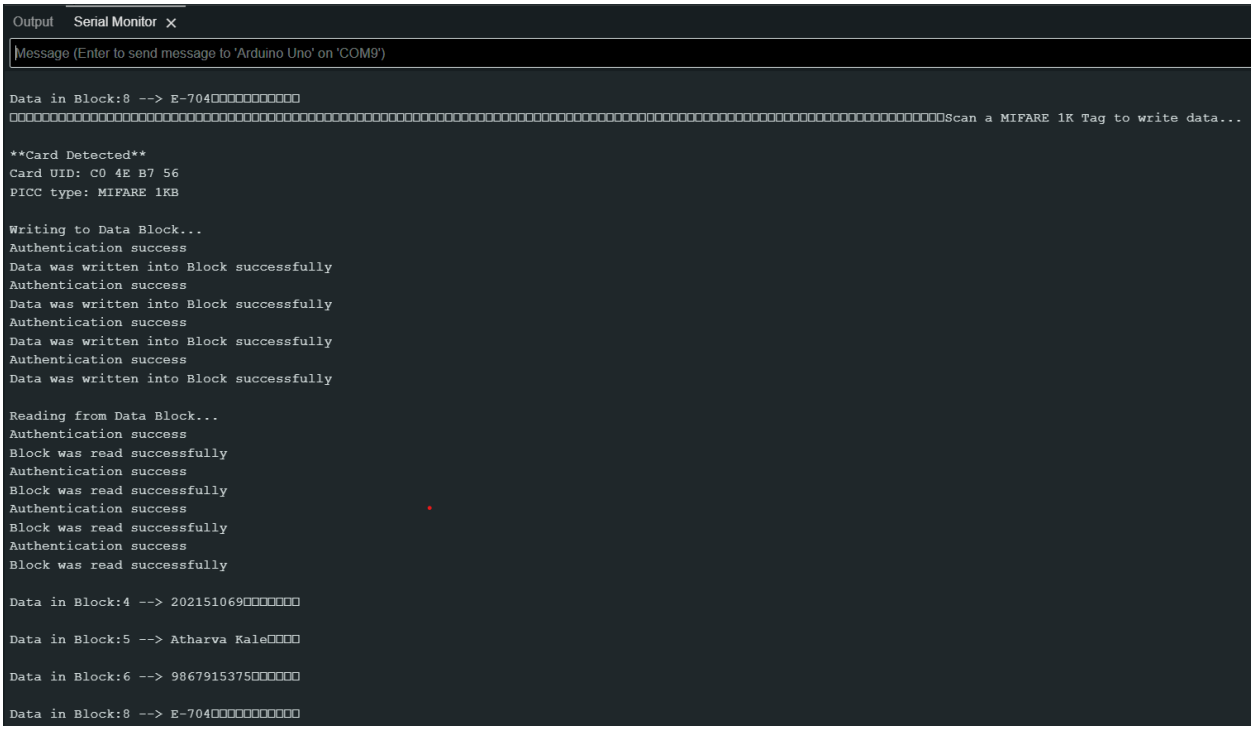


Description of memory of RFID card:

The first line shows the firmware version of the MFRC522 IC. After scanning the RFID Card, we get the UID, SAK and Type of RFID tag. In this case, the UID is '$0 AE B8 56', SAK is '08' and the type of card is MIFARE 1K. Here, we can see the actual memory dump of the MIFARE 1K Tag. A typical MIFARE 1K RFID tag has 1K Byte of memory organized into 16 Sectors (Sector 0 to Sector 15). Each Sector consists of 4 Blocks. Block 0 of Sector 0 is reserved for storing Manufacturer Data. Usually, this Block

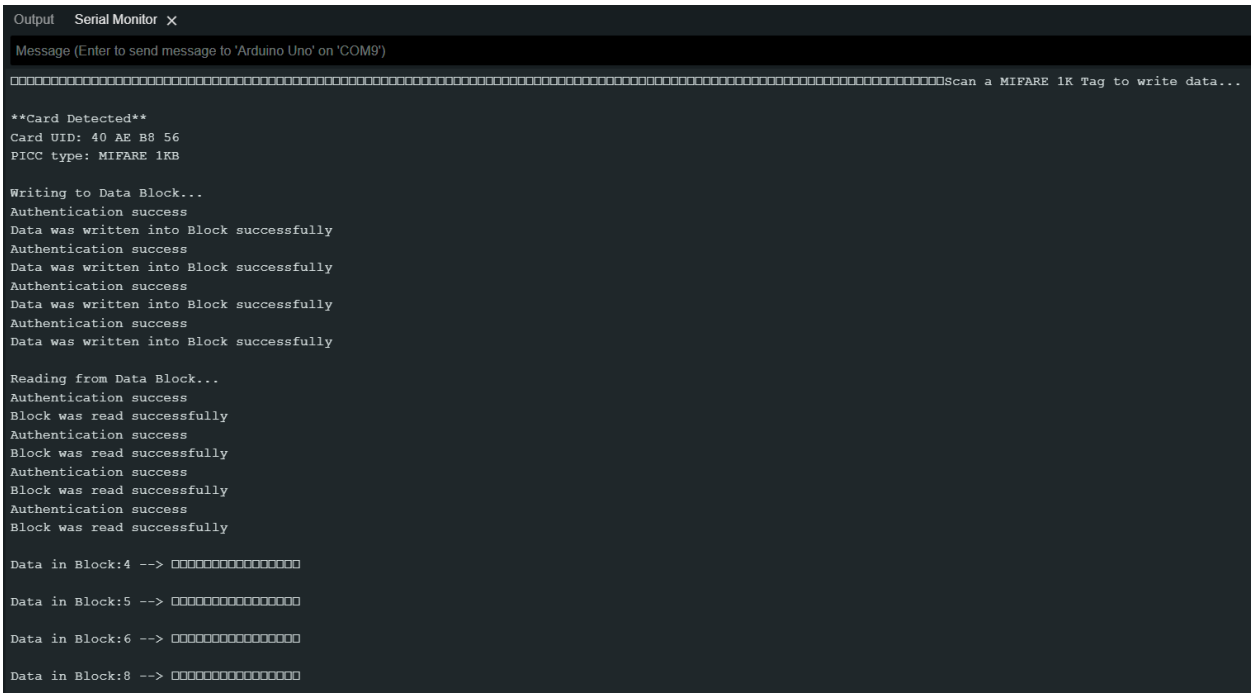
contains 4 Byte UID (Unique ID) in case of MIFARE 1K Tags (and also MIFARE 4K, MIFARE Mini tags from NXP). Advanced Tags like MIFARE Plus, MIFARE Ultralight, MIFARE DESFire consists of a 7 Byte UID. Each Sector consists of three Data Blocks, which can be used for storing user data. The last Block of each Sector i.e., Block 3 in case of Sector 0, Block 7 in case of Sector 1 and so on is known as Sector Trailer. As there are 16 Sectors, there are 16 Sector Trailers.

Writing Data into a RFID Card.



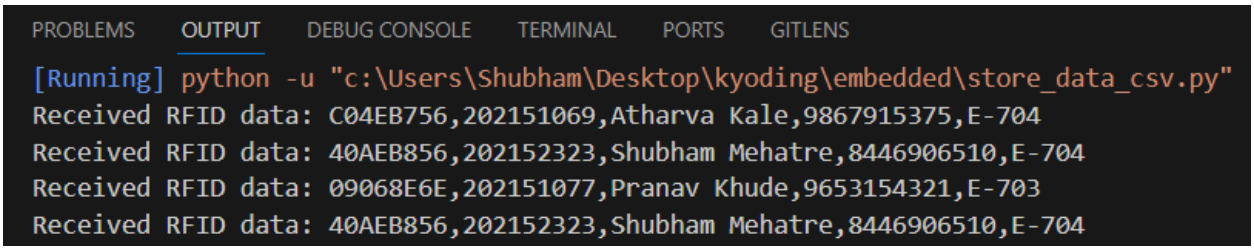
Here, with the help of Read-Write-Reset code, we have written data inside the RFID card. We have made changes in block no – 4, 5, 6 and 8 and stored the Student ID, Name, Phone number and Room number of that student inside the card.

Resetting Data of a RFID Card.

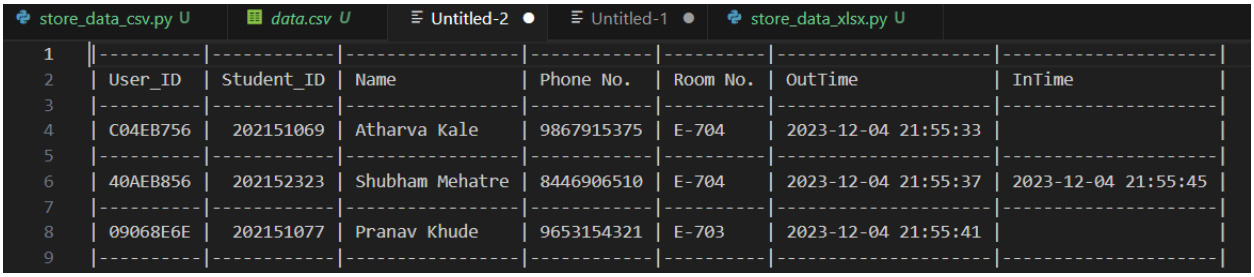


Here, with the help of same Read-Write-Reset code, we have deleted the data stored inside the RFID card. We have made changes in block no – 4, 5, 6 and 8 and deleted the stored data of Student ID, Name, Phone number and Room number of that student from the card.

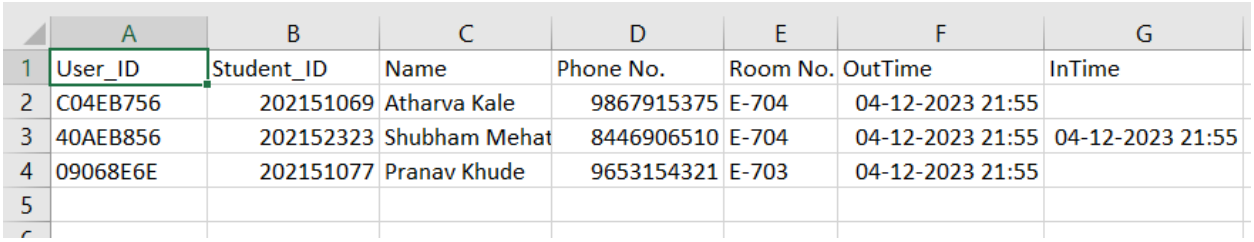
Example



CSV file



Excel file

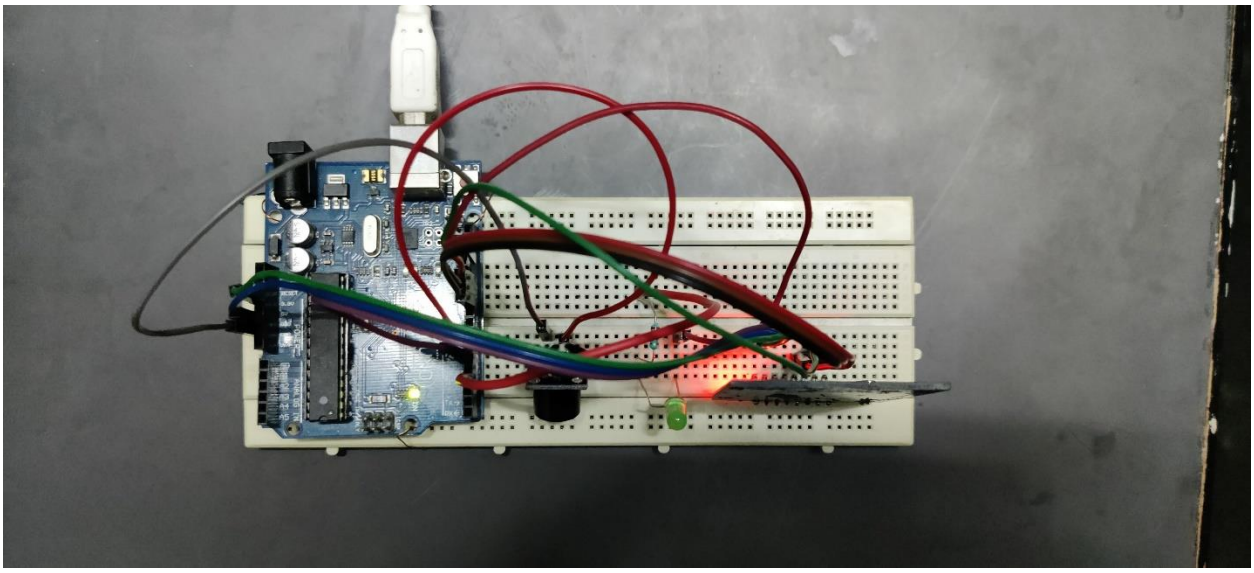


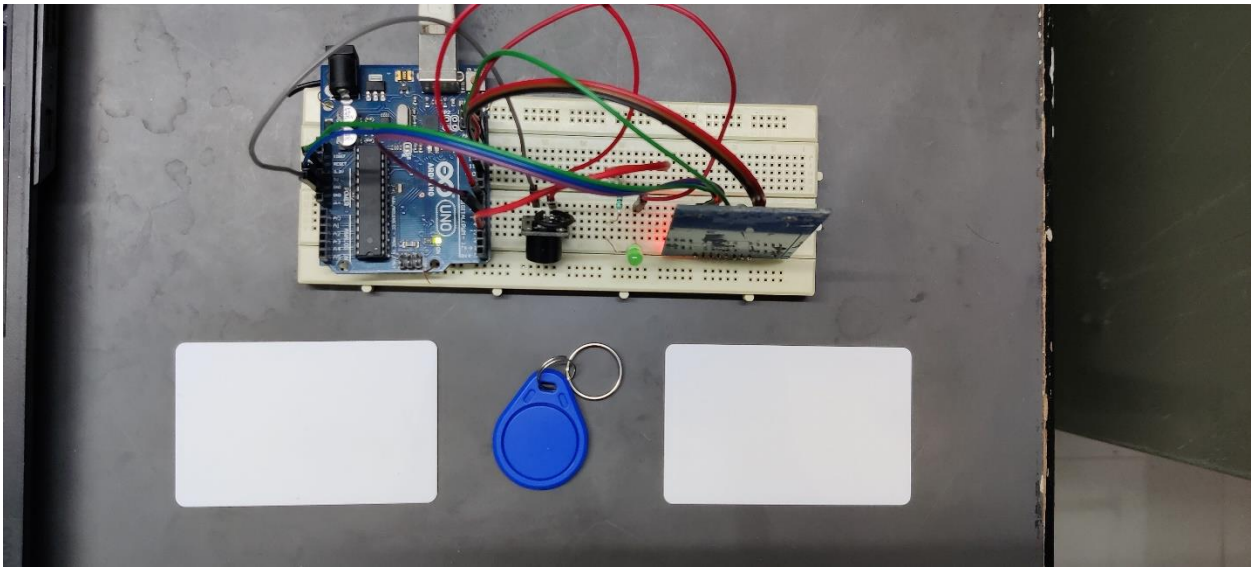
	A	B	C	D	E	F	G
1	User_ID	Student_ID	Name	Phone No.	Room No.	OutTime	InTime
2	C04EB756	202151069	Atharva Kale	9867915375	E-704	04-12-2023 21:55	
3	40AEB856	202152323	Shubham Mehat	8446906510	E-704	04-12-2023 21:55	04-12-2023 21:55
4	09068E6E	202151077	Pranav Khude	9653154321	E-703	04-12-2023 21:55	
5							

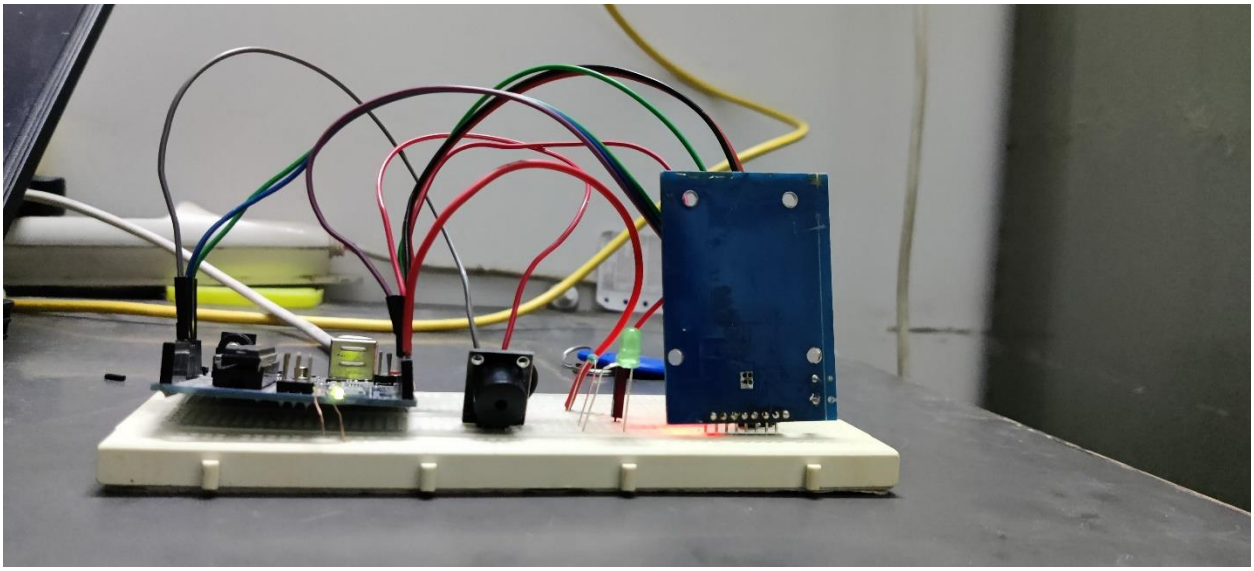
Here, we can see from the first image which is screenshot of the terminal of the VS Code, we can see that, first a student named ‘Atharva Kale’ scans his card and goes out of the hostel. As soon as he scans the card, since he is going out of the hostel, a new entry is created and all his personal data is pasted in that row. Now, since he is going out so current date and time is stored in the out-time field. After that other two students ‘Shubham Mehatre’ and ‘Pranav Khude’ also go out of the hostel. So, program performs the same thing. Creates new entry for both of them, paste their personal data and store current date and time in out-time field. Now, after sometime, the student named ‘Shubham Mehatre’ returns to the hostel. So, he scans his card. Now, since already an entry for him was present, so program just updates the current date and time in in-time field. The other two students have not returned to the hostel yet. So, the in-time field for both of them is empty.

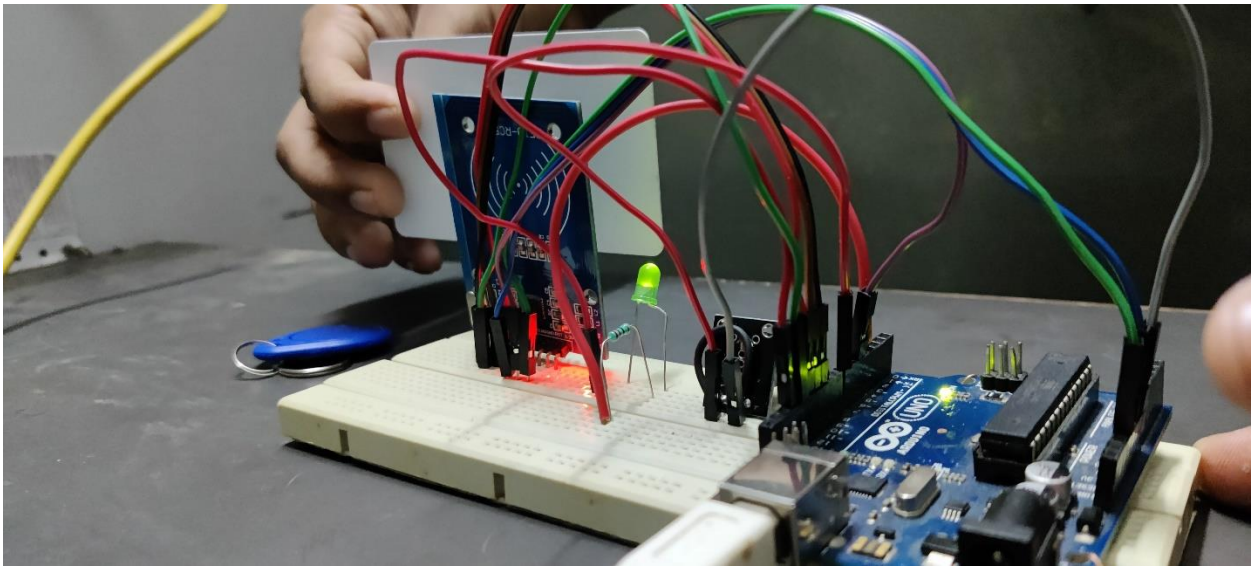
Above we have attached both layout of how the data of the attendance of the students will be stored in CSV file as well as Excel file.

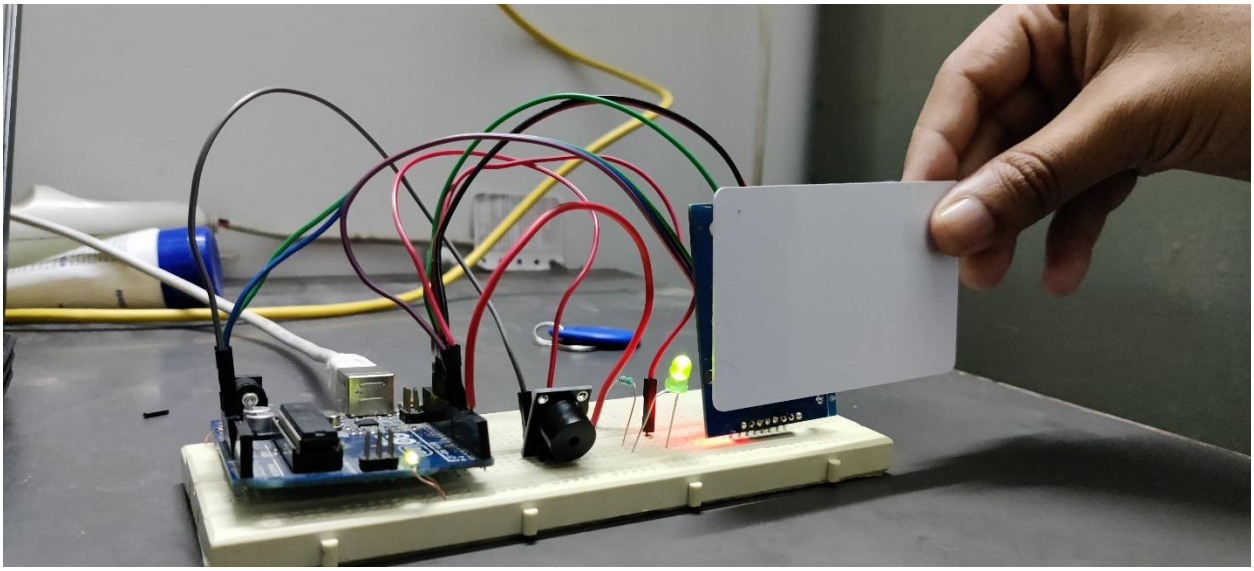
Images of our project











GitHub Link of our project:

https://github.com/shubhamm-26/RFID-attendance-System.git

References

Arduino UNO documentation:

https://docs.arduino.cc/hardware/uno-rev3

MFRC522 Library and RC522 RFID card reader module documentation:

https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

Research Paper based on RFID based attendance system for reference:

https://www.researchgate.net/publication/235598499_RFID-Based_Students_Attendance_Management_System