AUTODESK Instructables

Smart Toll System With Web Integration

By SPLASH ELECTRONIC in CircuitsArduino
Published Apr 13th, 2025

Introduction: Smart Toll System With Web Integration

This is a **SMART TOLL GATE SYSTEM WITH WEB INTEGRATION** , this is a prototype project of a toll gate system where the user(vehicles) can scan their RFID cards for toll just like FAST-TAG in India , and then there will be a proprietary software in the toll booth for the authentication and new user creation.

CHECKOUT THE WEBSITE: checkout_website_now

CHECKOUT THE REPO : check_out_git_repository_of_project (LegitCoconut)

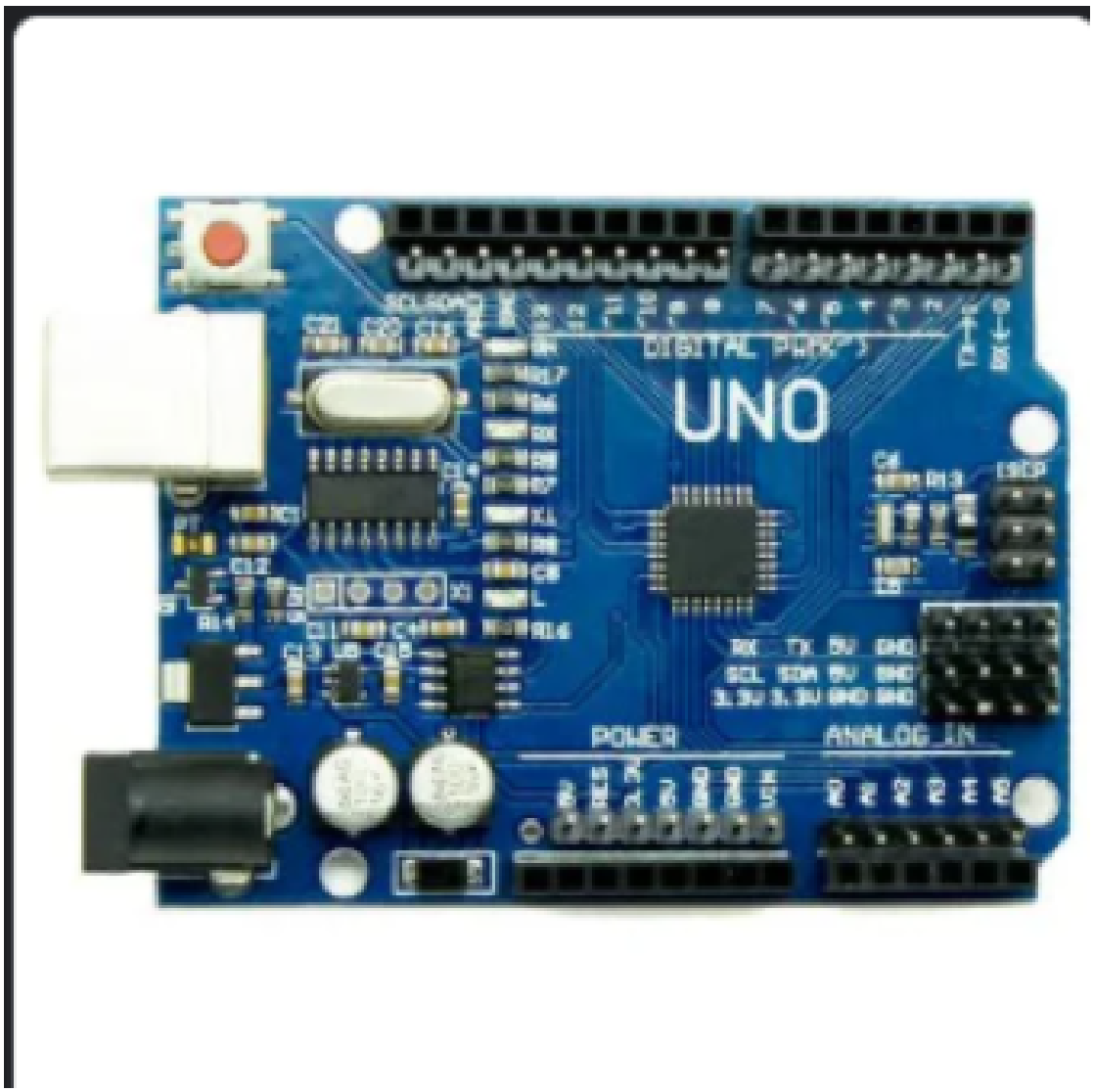
1. If the incoming vehicle don't has a vehicle pass (RFID card) , A new user account is created with the vehicle license number and a minimum recharge amount is got from the user, Toll

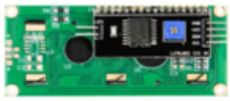
amount is deducted and allowed to pass through the toll
2. If the incoming vehicle has an account but don't has a minimum balance then a recharge prompt is shown and the card is recharged , Then money deducted and allowed to pass
3. If the incoming vehicle has an account and also the minimum balance then , Toll amount is deducted and allowed to pass through the toll

All the user Details, Toll gate log are stored in an online instance of **mongoDB** for fast and reliable centralized acess. An online proprietary web page is also there to see the logs and recharge the account of users.

This project can be developed very much to introduce a centralized server for handling all the loads, Perfect user authenticated recharge portal and automatic user creation and A portal of the authorities to check the cash collected and other necessary data and AI models can also be introduced to take care of peak traffic hours by making vehicles according to their size go through different toll gates.

Supplies





Since this is an initial prototype, only fundamental components have been used. Advanced hardware options will be proposed in later stages of development.

The following components were used in the prototype:

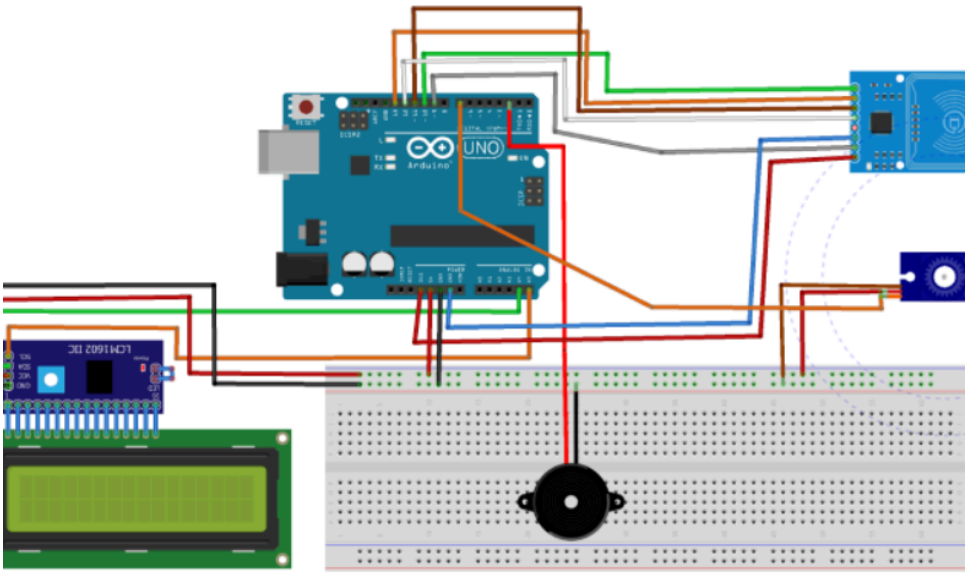
1. **Arduino® Uno**
2. **RFID Reader (MFRC522) with Tags**
3. **16×2 Liquid Crystal Display (LCD) with I^{2}C Interface**
4. **Servo Motor**
5. **Breadboard and Jumper Wires**

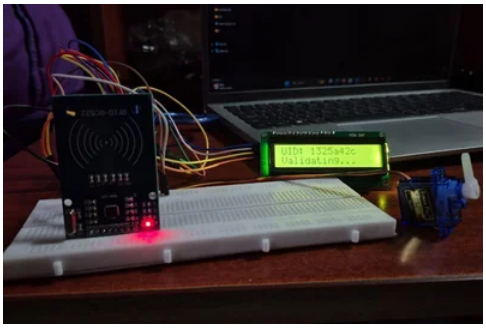
EXPLAINING WHAT ARE ALL THE USE OF THOSE COMPONENTS

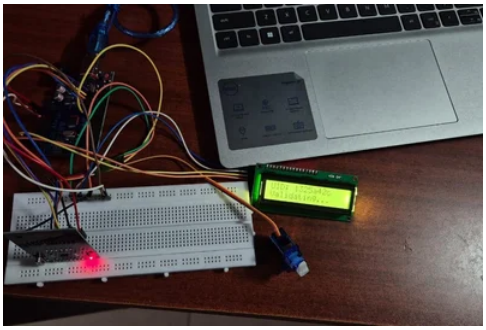
1. **Arduino Uno** as the brains of the system which can get the sensor data , display details in the LCD control the servo motor and all.
2. **RFID reader :** every car will have a rfid card attached to the windshield or the driver can tap the rfid on the sensor , for getting the info about the car and the vehicle id and all.

3. **16x2 LCD display:** Act as a mode of communication with the driver or the user in the toll gate
4. **Servo Motor:**

Step 1: SETTING UP THE HARDWARE CONNECTIONS







Please make sure that you power the system using the barrel jack with power source between 7v and 12v, use USB for only data transfer.

Connecting the RFID sensor

1. VCC -> 3.3v (only power from 3.3 else it will burn)
2. GND -> GND
3. RST -> D9
4. MISO -> D12
5. MOSI -> D11
6. SCK -> D13
7. SDA -> D10

Connecting the LCD screen

1. VCC -> 5v
2. GND -> GND
3. SDA -> A4
4. SCL -> A5

Connecting the Servo Motor

1. VCC -> 5v
2. GND -> GND
3. SIGNAL -> D6

Step 2: SETTING UP ARDUINO CODE

1. Library Inclusions

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <MFRC522.h>
#include <Servo.h>
```

1. **Wire/I2C**: For LCD communication via I2C.
2. **SPI**: Required by MFRC522 RFID reader.
3. **MFRC522**: Library to interact with the RFID module.
4. **Servo**: For controlling servo gate movement.

2. Pin Definitions & Object Initialization

```
#define SS_PIN 10
#define RST_PIN 9
#define SERVO_PIN 6

MFRC522 mfrc522(SS_PIN, RST_PIN);
LiquidCrystal_I2C lcd(0x27, 16, 2);
Servo gateServo;
```

1. SS_PIN, RST_PIN: Chip Select and Reset for RFID.
2. SERVO_PIN: Servo PWM signal.
3. mfrc522: Object to manage RFID functions.
4. lcd: 16x2 LCD on I2C address 0x27.
5. gateServo: Servo motor controller.

3. Setup Function

```
void setup() {
Serial.begin(9600);
SPI.begin();
mfrc522.PCD_Init();
lcd.init();
lcd.backlight();
gateServo.attach(SERVO_PIN);

lcd.setCursor(0, 0);
lcd.print("Waiting for Toll");
Serial.println("System Ready...");
}
```

1. Serial, SPI, RFID, LCD, and Servo initialized.
2. LCD displays *“Waiting for Toll”* at startup.
3. Useful for user feedback and debugging.

4. Main Loop – Card Detection & Handling

```
if (!mfrc522.PICC_IsNewCardPresent() || !mfrc522.PICC_ReadCardSerial()) {
```

```
return;
}
```

1. **Card polling logic**.
2. Returns if no card is present or can't be read.

5. UID Reading and Sending

```
String uidStr = "";
for (byte i = 0; i < mfrc522.uid.size; i++) {
uidStr += String(mfrc522.uid.uidByte[i], HEX);
}
```

1. Builds UID string from card's byte data (in HEX).
2. Acts as a unique identifier.

```
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("UID: " + uidStr);
lcd.setCursor(0, 1);
lcd.print("Validating...");
Serial.println(uidStr);
```

1. Displays UID and sends it to the laptop via Serial.
2. Laptop is expected to handle backend validation.

6. Balance Reception from Laptop

```
String balance = "";
while (Serial.available() == 0) {
delay(100);
}
balance = Serial.readString();
balance.trim();
```

1. Waits for laptop to respond with the user’s balance.
2. Reads and cleans the balance string.

```
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Balance: Rs." + balance);
```

1. Balance is displayed on LCD.

7. Gate Control (Servo)

```
gateServo.write(90); // Open gate
delay(2000);
gateServo.write(0); // Close gate
```

1. Moves servo to open position for 2 seconds.
2. Returns it to closed state after delay.

8. Reset System Display

```
delay(2000);
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Waiting for Toll");
```

1. Waits and resets display for next RFID interaction.

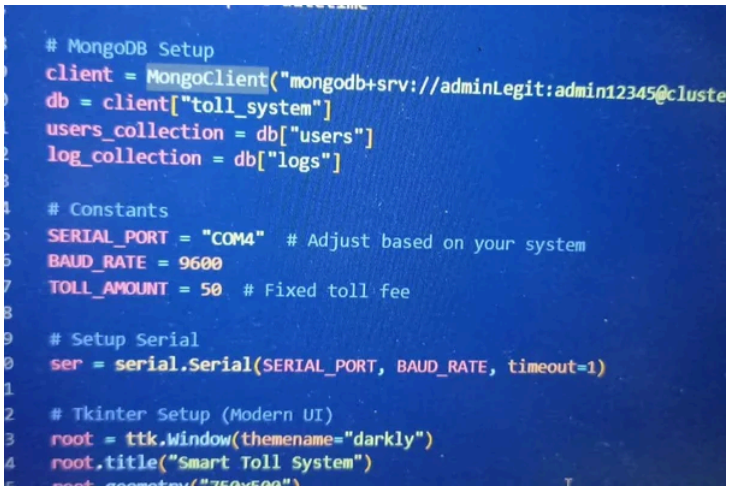
⚙️ Overall Functionality (Summary)

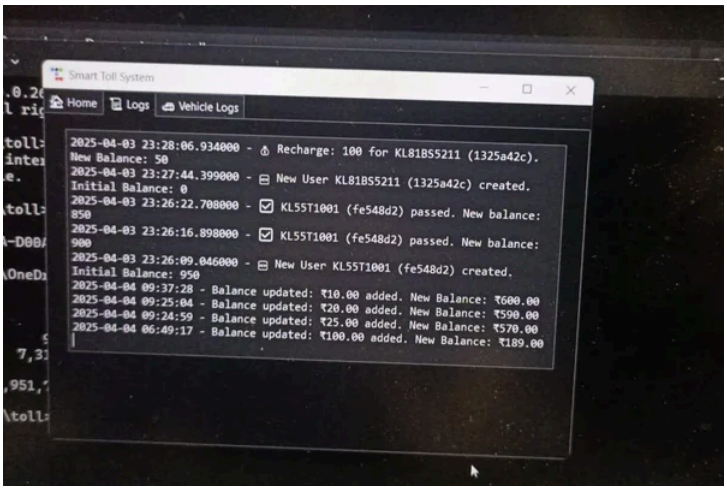
This code creates a **simple RFID-based toll authentication system**:

1. **Detects RFID cards** via MFRC522 module.
2. **Sends the UID** to a connected laptop through the serial port.
3. **Laptop processes and returns balance** info.
4. **Displays balance** on an LCD screen.
5. **Controls a servo** to simulate gate opening based on balance verification.
6. **Resets** to standby after completing the operation.

It’s a closed-loop system where the **Arduino handles I/O**, while the **laptop manages authentication logic**.

Step 3: SETTING UP THE PYTHON SCRIPT





MAKE SURE TO CREATE AN MONGODB CLUSTER ONLINE AND CONNECT IT THERE

1. Data Input – UID from Arduino

1. Arduino sends UID over serial (e.g., from an RFID card).
2. serial_read() runs in a thread, continuously checks ser.readline() for incoming UID.
3. Once UID is received, it's passed to process_uid(uid).

2. Processing UID

1. handle_uid(uid) checks MongoDB (users_collection) for a document with that UID.
2. Two cases:
3. **Existing user:**
4. Gets balance and vehicle from the user document.
5. If balance >= TOLL_AMOUNT (50):
6. Deduct toll: new_balance = balance - 50
7. Update DB: set new balance.
8. Add log to both global (log_collection) and user (logs array in user doc).
9. Send new_balance back to Arduino via serial.
10. Else:
11. Show recharge form to recharge balance.
12. **New user:**
13. Show recharge form and ask for vehicle number.

3. Recharge Flow

1. User enters recharge amount (and vehicle if new user).
2. process_recharge(...):
3. Validates input.
4. If **new user**:
5. Create a document with:
6. uid, vehicle, balance = amount - 50, and empty logs list.
7. If **existing user**:
8. Add amount to balance, deduct 50, update balance in DB.
9. Add a log entry.
10. Send new_balance to Arduino via serial.

4. Logs Handling

1. update_log(uid, vehicle, message):
2. Creates a log entry with time and event.
3. Adds it to:
4. logs array in user document.
5. log_collection (global logs).
6. Logs are shown in:
7. **Logs tab**: All entries from log_collection, newest first.
8. **Vehicle Logs tab**: Fetches logs of selected vehicle from user's logs array.

5. Vehicle Dropdown

1. Periodically (every 5 seconds), load_vehicle_dropdown() pulls all vehicle numbers from users_collection and updates the dropdown list in UI.

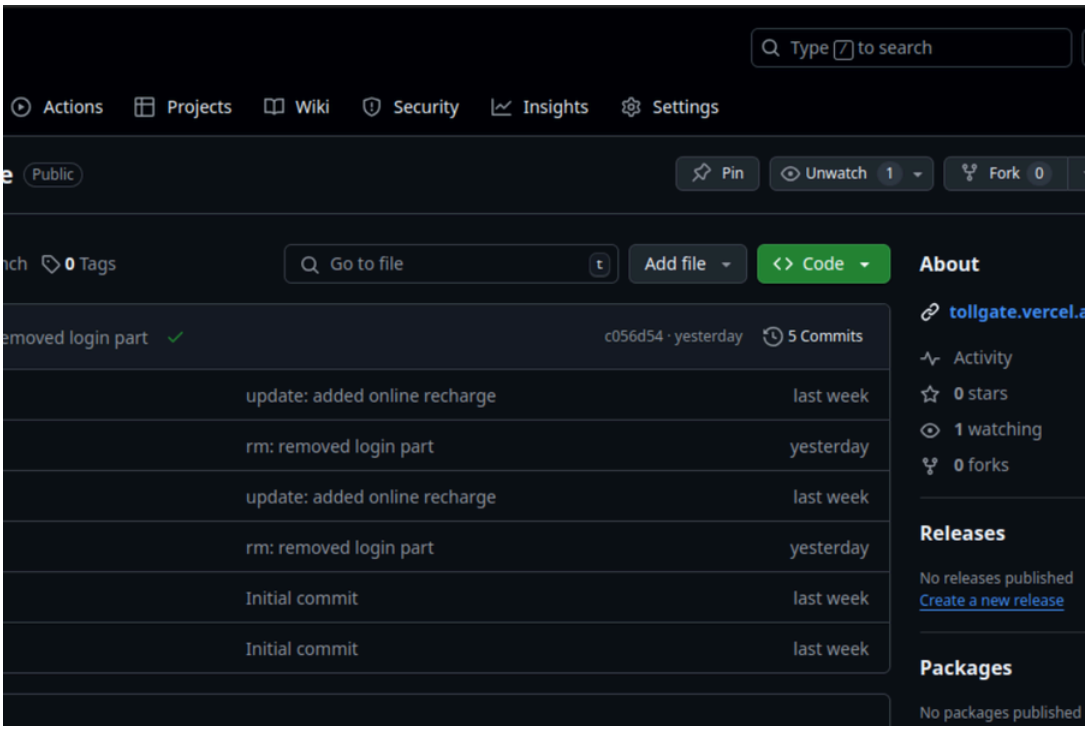
6. Final Output to Arduino

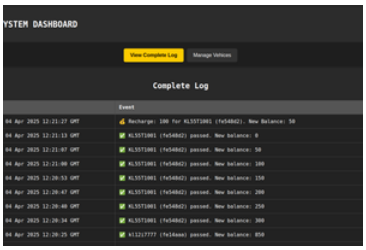
1. After processing, send_to_arduino(balance) sends the current balance back to Arduino as bytes.

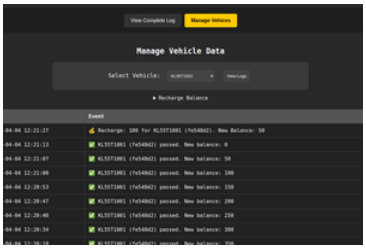
Summary:

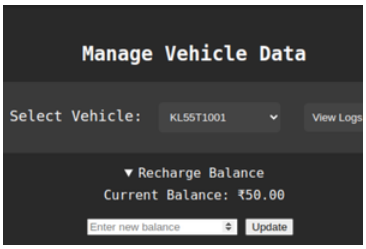
1. **Input**: UID from serial → check DB.
2. **Process**:
3. If user exists → check balance → deduct toll if enough.
4. Else → prompt for recharge/new user.
5. **Output**: Updated balance to Arduino, logs to DB, updates in UI.

Step 4: SETTING UP THE WEBPAGE





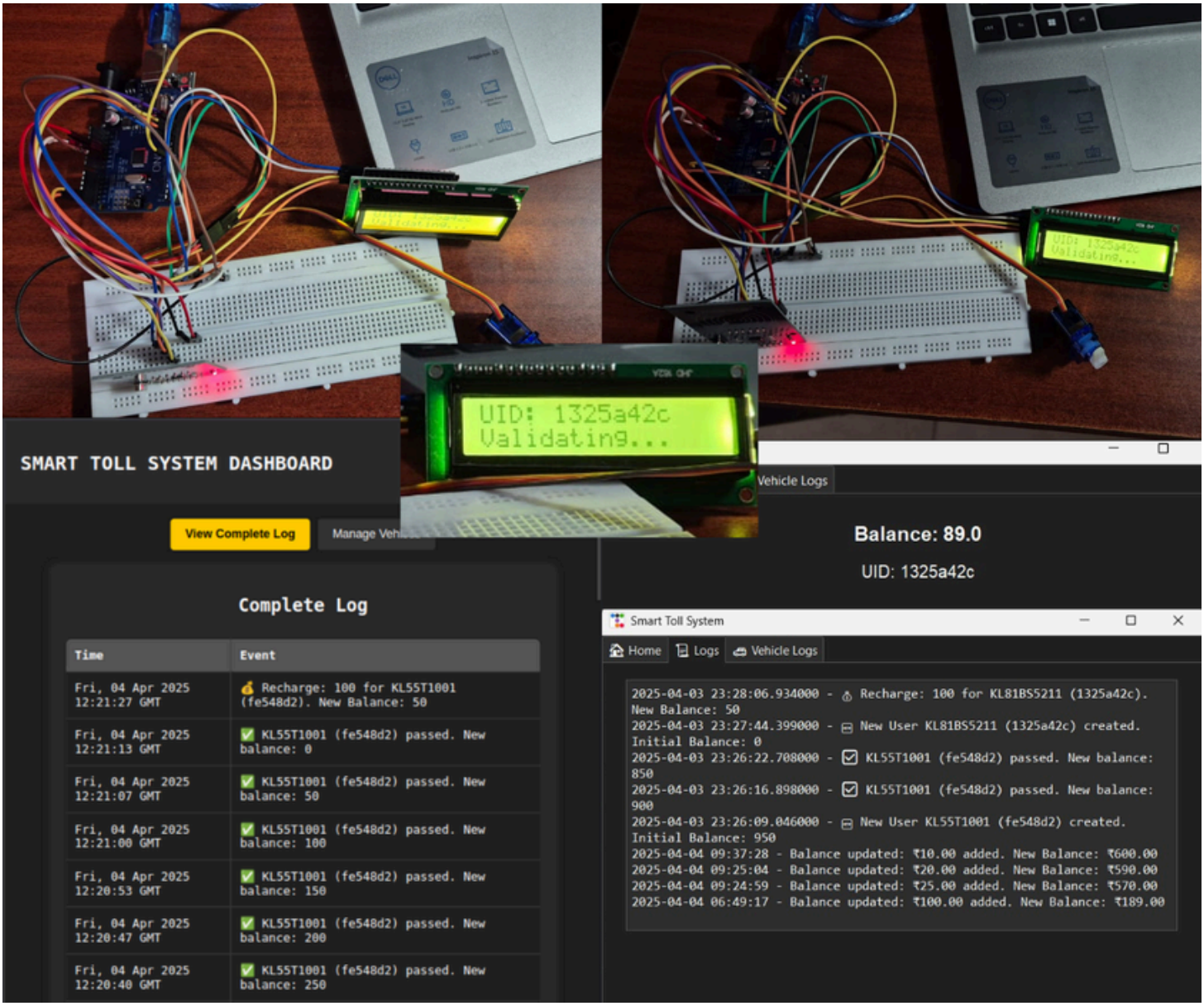




To setup the web page, fork the project_repo and make an account on vercel for hosting the web-page, and import the repo onto vercel, then while deployment add username , password and the MongoDB URI as environmental variables and then deploy the web page.

for more info on how to setup the web page visit the GitHub repo

Step 5: THE PROJECT IS COMPLETE



If you see there need to be more improvements in the projects fork the repo and make all the chnages and make a pull request

fell free to put out sugestions