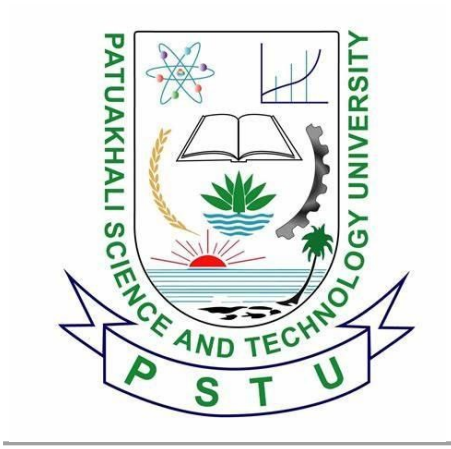


PATUAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

COURSE CODE EEE-212
Electrical Technology Sessional

Project Report

SUBMITTED TO:

Md. Naimur Rahman
Professor
Department of Electrical and Electronics Engineering
Faculty of Computer Science and Engineering

SUBMITTED BY:

Md. Sadman Kabir Bhuiyan
ID: **2102020**,
Registration No: **10147**
Faculty of Computer Science and Engineering

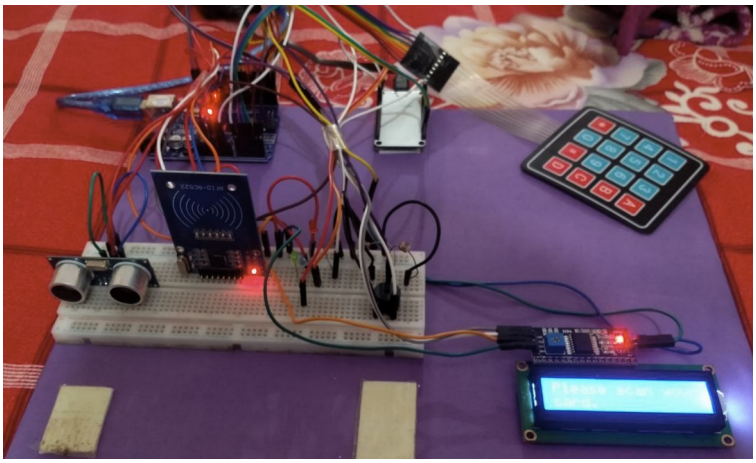
Md. Sharafat Karim
ID: **2102024**,
Registration No: **10151**
Faculty of Computer Science and Engineering

Date of submission: **29 October 2024**
Project title: **Metro Recharge Point**

Metro Recharge Point

Project Overview

The **Metro Recharge System** is a microcontroller-based, smart recharge station for metro cards. It incorporates RFID card scanning, balance management, and interactive features to enhance user experience at recharge stations. The system provides easy card balance checking and recharge options via both on-site and online methods. It combines multiple technologies, including Arduino, ESP32, and a Python-based graphical display.



Objectives

- To enable balance checks and recharges for metro cards via RFID.
- To use an intuitive user interface for easy navigation and user engagement.
- To provide an online component that supports data storage and update via RESTful APIs.
- To utilize visual indicators (LEDs and screen displays) and sound signals to enhance user feedback.

System Architecture

1. **Arduino Metro Station Control**: Arduino manages the RFID card reading, balance display, and navigation of options for users.
2. **ESP32 Internet Connectivity**: Communicates with a remote server to retrieve or update card balances. Visual feedback is provided via LEDs.
3. **Metro Backend API**: A RESTful API with FastAPI and SQLAlchemy allows users to store, retrieve, and update balances remotely.
4. **Python GUI**: Displays balance information in real-time, using Tkinter for the full-screen GUI.

Hardware Specs

1. Ultrasonic Sensor

An ultrasonic sensor measures distance by using ultrasonic waves. It sends out a sound wave at a frequency above the range of human hearing. When the sound wave

hits an object, it reflects back to the sensor. The sensor calculates the distance based on the time it took for the sound wave to return.

2. RFID Card

RFID (Radio Frequency Identification) technology uses radio waves to read and capture information stored on a tag attached to an object. An RFID system consists of a reader, an antenna, and a card/tag.

3. Buzzer

A buzzer is an audio signaling device, which can be mechanical, electromechanical, or piezoelectric. It emits sound when an electric signal is applied, commonly used in alarms, timers, and user notifications.

4. LDR

LDR detects light intensity, by chaning it's resistance value. It's used to measure up the presence of light.

4. Rain Sensor

A rain sensor detects water and is used to detect rainfall. It consists of a rainboard and a control module. When rainfall is detected, the sensor outputs a signal that can be used to trigger actions, such as closing windows or alerting users.

5. LiquidCrystal I2C Display

An I2C LiquidCrystal display is a 16x2 or 20x4 LCD display that communicates over I2C, reducing the required number of pins. It is ideal for Arduino projects where the number of pins is limited.

6. Matrix Keypad

A matrix keypad consists of a grid of buttons in rows and columns. When a button is pressed, it completes a circuit, and the microcontroller detects the row and column to determine which button was pressed.

7. ESP32

The ESP32 is a powerful microcontroller with built-in WiFi and Bluetooth, making it ideal for IoT projects. It has multiple GPIO pins and supports various communication protocols, making it highly versatile.

8. Arduino Uno

The Arduino Uno is a popular microcontroller board based on the ATmega328P. It has 14 digital input/output pins, 6 analog inputs, and a USB connection for programming. It's widely used in DIY electronics projects.

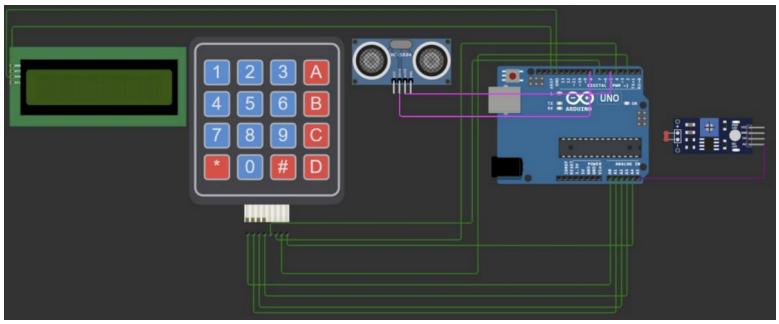
Software

- **FastAPI**: Backend API framework for managing user balances.
- **SQLAlchemy**: ORM for database interaction in the backend API.
- **Tkinter**: Python library for GUI display on full-screen devices.
- **Python**: For serial communication and display interaction with the Arduino.

Procedures

1. Front end

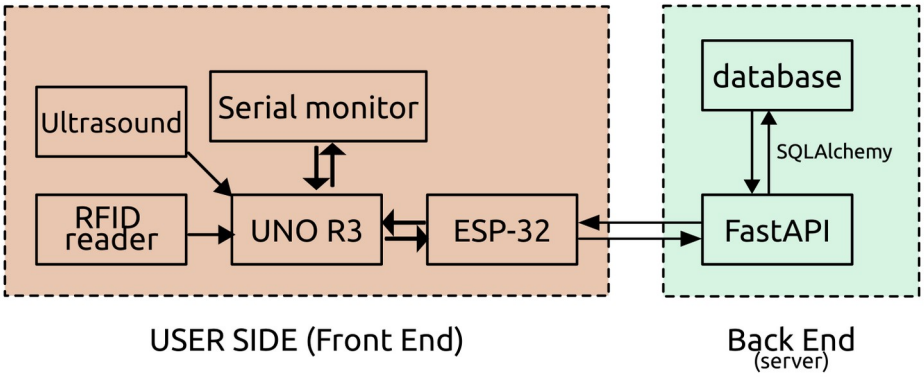
1. **Connect RFID Reader** to Arduino:
    - Connect SS to Pin 10, RST to Pin 9, and SPI pins to respective Arduino SPI pins (MISO, MOSI, SCK).
2. **Connect LCD Display**:
    - Connect SCL and SDA of the LCD to Arduino's SCL and SDA.
3. **Set Up Keypad**:
    - Connect the keypad rows and columns to Arduino digital pins as defined in the code.



4. **Wire the LEDs and Buzzer**:
    - Connect LEDs and buzzer to separate GPIO pins on the ESP32, making sure to set these pins as outputs.
5. **Connect ESP32**:
    - Use jumper wires to connect the ESP32 to the Arduino's TX and RX pins if needed for serial data exchange.
    - Power the ESP32 and connect it to Wi-Fi (configured in the code).
6. **Set Up Ultrasonic Sensor and LDR**:
    - Connect the ultrasonic sensor's trigger and echo pins to Arduino.
    - Connect the LDR to an analog pin and set up the brightness control on the LCD.

2. Back end

1. **Upload Arduino Code**:
    - Flash the provided Arduino code to handle RFID scanning, keypad interaction, and balance display.
    - Ensure `Serial.begin(9600);` matches the serial baud rate used by the Python GUI script.



2. **Upload ESP32 Code**:
    - Flash the ESP32 with the provided code for handling Wi-Fi connection, HTTP requests to the backend, and LED feedback based on the balance retrieved or updated.
    - Configure the Wi-Fi credentials and server URL in the ESP32 code.
3. **Set Up Backend API**:
    - Create a backend server using FastAPI and SQLAlchemy.
    - Set up endpoints for fetching and updating balances. Host the API on a public server (like Render or Heroku).
4. **Set Up Python GUI**:
    - Install necessary Python libraries (`pyserial` for serial communication and `tkinter` for GUI).
    - Run the Python script to display the balance and real-time data from Arduino on a full-screen GUI.

Code Implementation

Code is available on GitHub.
Link https://github.com/SharafatKarim/metro-api

Backend API

The **Metro-API** backend is built with FastAPI and SQLAlchemy to manage user balances. It includes endpoints for creating users, fetching balances, and updating balances upon recharge or discharge.

Endpoints

- `GET /users/{id}` - Retrieves a user's balance.
- `POST /users` - Creates a new user.
- `PUT /users/+/{id}` - Recharges a user's balance.
- `PUT /users/-/{id}` - Deducts from a user's balance.
- `DELETE /users/{id}` - Deletes a user account.

Testing and Troubleshooting

Common Issues and Solutions:

- **Serial Communication Errors**: Ensure baud rates match across Arduino and Python scripts.
- **Wi-Fi Connection Issues**: Double-check SSID and password for ESP32 connection.

- **Display Delays**: If data doesn't show immediately on the GUI, check the Arduino-Python serial connection.

Scope

- **Hardware Integration**: Utilize components such as RFID Readers, ESP32, PIR sensors, and buzzers for card detection and system feedback.
- **Backend Development**: Develop a FastAPI-based RESTful API to handle user balance management, including recharge, discharge, and user creation functionalities.
- **Mobile Banking Integration**: Enable secure mobile banking transactions for recharging metro cards.
- **Real-time Data Processing**: Ensure real-time updates to user balances via cloud-hosted services. It can also be deployed on a local server.

Benefits

- **User Convenience**: Quick and easy metro card recharges from any location using mobile banking or specific machines. Further systems can be easily developed for using the same backend. For example, bus ticket system.
- **Real-time Updates**: Users can track balances and manage their metro cards through a user-friendly interface. Or, they can also use an interface hosted on web to quickly access. Besides, there can be admin panels and administrations control access, in order to debug things out.
- **Scalability**: Designed to handle large volumes of users and transactions efficiently. Besides, it can be integrated into other systems, as well as handling multiple purposes.
- **Data Analyze:** Backend servers can also implement anonymous data collection. Which will be a great help for future crowd controlling and preparing more efficient schedule for the entire train system.

Future Improvements

- Implement additional security measures, such as user authentication.
- Expand the system to handle multiple RFID cards concurrently.
- Provide more payment options, such as mobile wallet integration.

Conclusion

This project presents a comprehensive solution for metro card recharge stations, using a blend of hardware components and software technologies. It effectively demonstrates how microcontrollers, Wi-Fi modules, and server communication can be used to develop an efficient, user-friendly recharge system. The balance-checking and recharge features allow users to have an easy and seamless experience at metro recharge points.