Transaction Server with RFID Terminal / POS Terminal / Toll Services

*NSAPDEV Final Project*

Members:

Azevedo, Marquus Joss

Fulcher, Lander Gabriel

Labindao, Carl Denzel

Malia, Gideon Andrew

**Change Control**

|  |  |  |
| --- | --- | --- |
| Version | Date | Comments |
| 0.1 | 07/20/2024 | Initial draft |
| 0.2 | 08/05/2024 | Final Output |

**Table of Contents**

[I. Project Description 3](#_Toc173795359)

[II. Project Capability and Scope 5](#_Toc173795360)

[III. System Architecture 5](#_Toc173795361)

[IV. Project Performance 6](#_Toc173795362)

[V. Server Application 8](#_Toc173795363)

[VI. Client Application 10](#_Toc173795364)

[Figure 1 Project Diagram for RFID 4](#_Toc173795350)

[Figure 2 CSV File Output in Proxmox 7](#_Toc173795351)

[Figure 3 Proxmox Server with RFID details 8](#_Toc173795352)

[Figure 4 Client connecting and registering RFID to server 8](#_Toc173795353)

[Figure 5 Libraries used by the Server 9](#_Toc173795354)

[Figure 6 Server.py Code 9](#_Toc173795355)

[Figure 7 Server.py Output in Proxmox Server 10](#_Toc173795356)

[Figure 8 Client.py Code 11](#_Toc173795357)

[Figure 9 Client Output 12](#_Toc173795358)

# Project Description

The project focuses on the development of a specialized Transaction Server optimized for RFID Terminal operation, which is essential for managing and processing transactions from RFID-tagged devices. This server is crucial for ensuring seamless financial transactions and operational continuity in the service environment. Drawing from the foundational server architecture principles, our solution prioritizes scalability and reliability. We will implement robust concurrency techniques, such as multithreading and multiprocessing, to handle concurrent transactions efficiently. This capability allows the server to manage multiple transaction requests simultaneously, thereby ensuring high efficiency and responsiveness.

At its core, the server software is programmed to receive and process real-time data from the RFID tag readers. It validates the balance associated with each RFID tag and executes transactional actions based on current balance information. Emphasizing security, the system includes robust measures for secure transaction handling and user authentication, safeguarding against unauthorized access, and prioritizing data integrity. The project seamlessly integrates RFID Terminal hardware, utilizing server sockets and web application frameworks to establish a flexible and dependable infrastructure. This approach ensures that the server meets the distinct operational requirements of RFID-based transaction environments, maintains robust performance, and adheres to stringent operational standards.

The project represents an application of advanced server technologies tailored specifically for RFID Terminal operation. By combining theoretical insights with practical implementation, our goal is to enhance transactional efficiency, security, and reliability in contemporary financial and service sectors.

**Project Diagram**

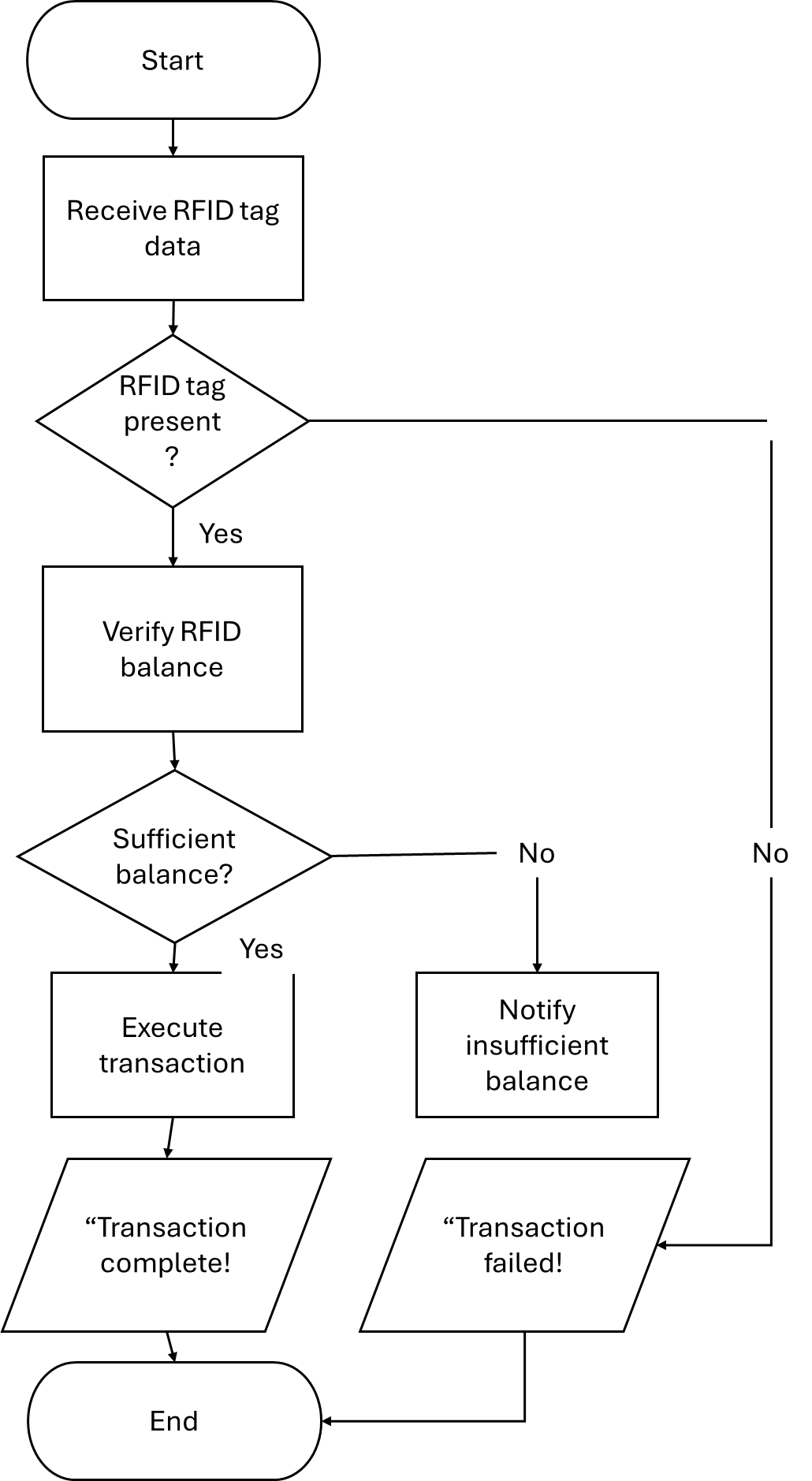


Figure 1 Project Diagram for RFID

The flowchart outlines the transaction process facilitated by the RFID tag. Initially, the data were collected from the RFID tag. Subsequently, the system verified the presence and validity of the tag. If confirmed, the RFID balance is checked. Sufficient funds trigger transaction execution. Otherwise, a transaction failure notification is issued to the user.

# Project Capability and Scope

The project’s capability includes:

* Communication between RFID tag readers
* Detect RFIDs
* Processing financial transactions based on the RFID tag data
* Determining RFID balance is sufficient or not
* Securing transaction records
* Authentication of user devices to prevent unauthorized access

The physical aspect of the project will have a limit in terms of scalability wherein transactions will only be from one to two transactions.

The project's capabilities include managing multiple client connections simultaneously, enabling seamless communication between RFID tag readers and the server. It detects and processes RFID tag data to handle financial transactions efficiently. The system verifies whether the RFID tag balance is sufficient for the transaction and ensures the security of transaction records without relying on a traditional database engine. The scope of the project involves developing a server application capable of processing data from RFID-enabled client devices, such as POS terminals and toll services. The server will track and log transactions in real-time, performing actions based on the data received from the RFID readers

# System Architecture

**Overall Logical Architecture**

Overall, the client is the RFID tag reader that will read the RFID tags that will send the data to the server. The server will read the data, check the balance, and perform necessary acts. Another is that the client architecture uses will be the Arduino IDE in which libraries for the wireless connection are used and also client functions. The libraries include **‘Wifi.h’** which is used for WiFi connection, **‘HTTPClient.h’** for HTTP communication, **‘SPI.h’** to communicate with RFID reader using SPI communication, **‘MFRC522.h’** which is used for RFID reader interface, **‘ArduinoJson.h’** for JSON serialization and deserialization, **‘TaskScheduler.h’** which is used for scheduling tasks on the ESP32. Functions used are **‘collectData()’** in which it read RFID data and sends it to the server,  **‘setup()’** for initialization of hardware and connecting to the WiFi, and **‘loop()’** for looping and executing scheduled tasks. The server software architecture, client hardware and software architecture go hand in hand for the RFID to read the data, obtain the data, send the data, checks balance and executing transactions.

**Server Software Architecture**

The server software will receive data from the client and read the content. The content is the RFID data. From there, it will check its balance, display and perform actions, depending on the balance received. It is built using the Python language and uses the **‘socket’** library for communication between server and client. The **‘threading’** library will also be used to handle multiple client connection. The architecture includes the **‘socket’** module for network communication, the **‘threading’** module for management of multiple client connections, the **‘json’** module to serialize and deserialize data, the **‘datetime’** module to create and handle timestamps and the **‘csv’** and **‘os’** modules for creating a CSV file and logging the transactions to the CSV file. Functions used in the architecture are **‘handle\_client(conn, addr)’** function which is used for managing individual client connections, **‘write\_to\_csv(log\_entry)’** function which is used for logging transactions to the CSV file and the **‘main()’** function to start the server and listen to incoming client connections. A thread lock is also used for ensuring that shared resources are accessed safely.

**Client Hardware and Software Architecture**

The client hardware will mainly consist of using an RFID reader and a tag to which it will be read by the reader, alongside the ESP32. An LED bulb will also be used for indication that the information is sent to the server. Its libraries will eventually require the drivers of the RFID reader to function properly. Functions will usually consist of reading the tag reader, getting its information, and then transporting the data to the server to be processed. Authentication will also be implemented to ensure security of the data and that it is the authorized user who is communicating with the RFID. The architecture still uses the **‘socket’** and **‘json’** module for client-server connection and serialization and deserialization of data.

# Project Performance

Tests were conducted in the project in terms of reading the RFID data, executing transactions, checking the balance, and connections between server and multiple clients. However, the students encountered many challenges in this project such as reading the data from the RFID tag, lack of time, and creating the transaction process. However, the most challenging is the connection between the server and the client since there are many errors in connecting the server to the client, data is sometimes does not send, and receivers blocks the data in which the client was not able to obtain the data sent by the server. The students were able to resolve some of the challenges such as the reading of RFID tags in which it can already be converted to a JSON object and sent to a csv file, creation of transaction process and connection between the server and client.

A screen shot of a computer

Description automatically generated

Figure 2 CSV File Output in Proxmox

In Figure 2, it is observed that the csv file from the Proxmox server is displayed due to the data sent by the client. Each column has their corresponding fields such as the RFID user ID, type, terminal ID, and the timestamp. The RFID user ID is for the identification of the user using the RFID. The ‘type’ field is considered the type of card used by the user. The terminal ID is the place where the user tapped the card. The ‘timestamp’ field is the time wherein the user tapped the card on the RFID reader.

A screen shot of a computer

Description automatically generated

Figure 3 Proxmox Server with RFID details

Figure 3 shows the output of the Proxmox Server in which the details of the RFID tag from the client are displayed. It also shows the server’s response to the client.

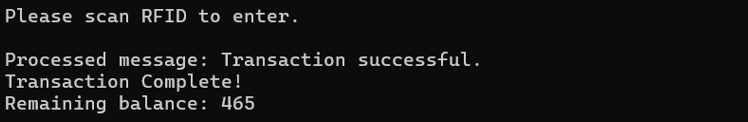


Figure 4 Client connecting and registering RFID to server

Figure 4 shows the client connecting to the server. After it is connected, it would request the user to scan the RFID to register. A message will be shown in which the RFID user ID and balance is displayed. This would then consider the user with the RFID registered.

# Server Application

The programming language used for the Server Application is Python in which it utilizes threading in handling multiple connections and processes. This also helps in ensuring that the client requests are handled well while maintaining efficient communication process and transaction process.

A screen shot of a computer program

Description automatically generated

Figure 5 Libraries used by the Server

In Figure 5, these are the libraries used for the server side in which it utilizes threading for efficient handling of multiple client connections and locking them in efficient communication and transaction processing; A CSV library is also included for putting JSON objects to the CSV file. The socket library is also used for the server to connect with the client.

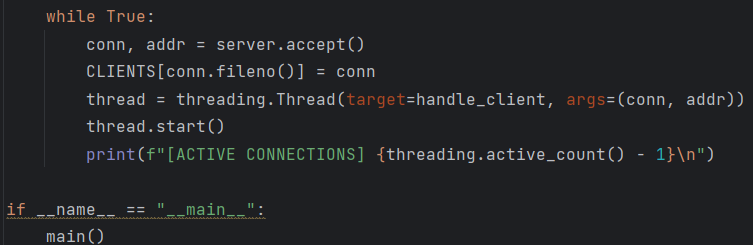
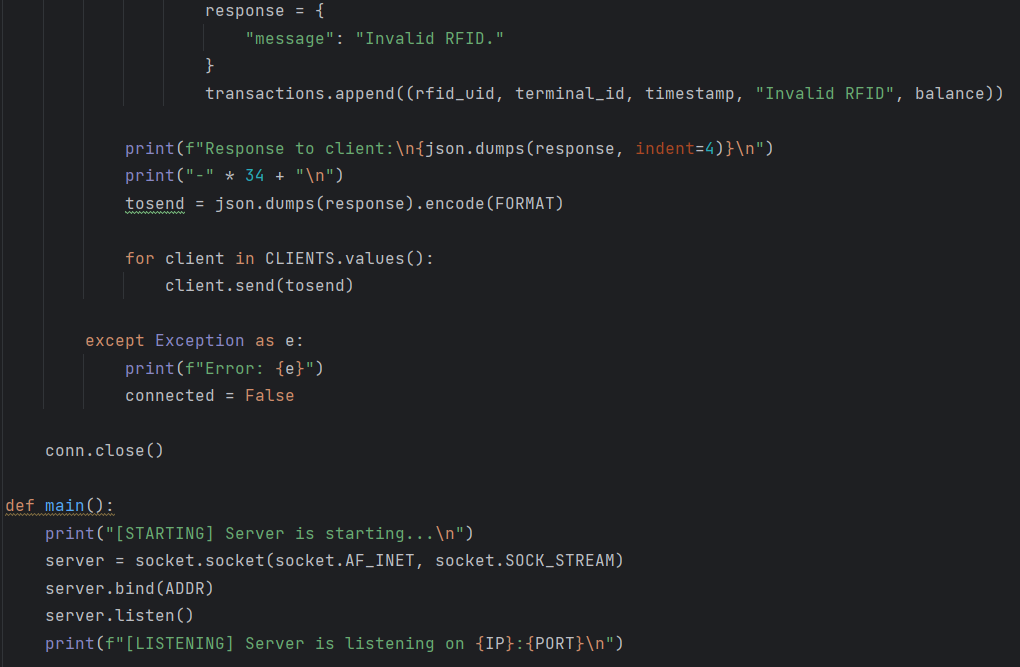
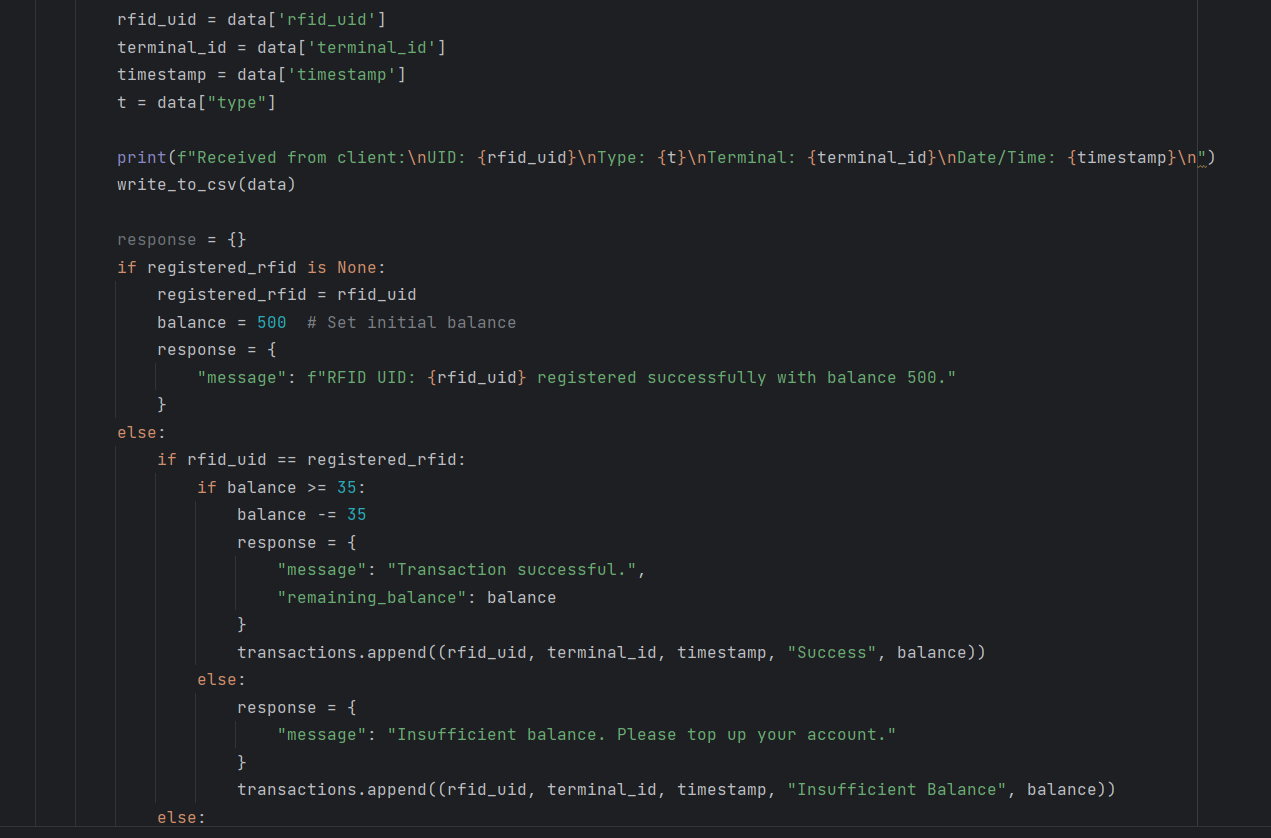
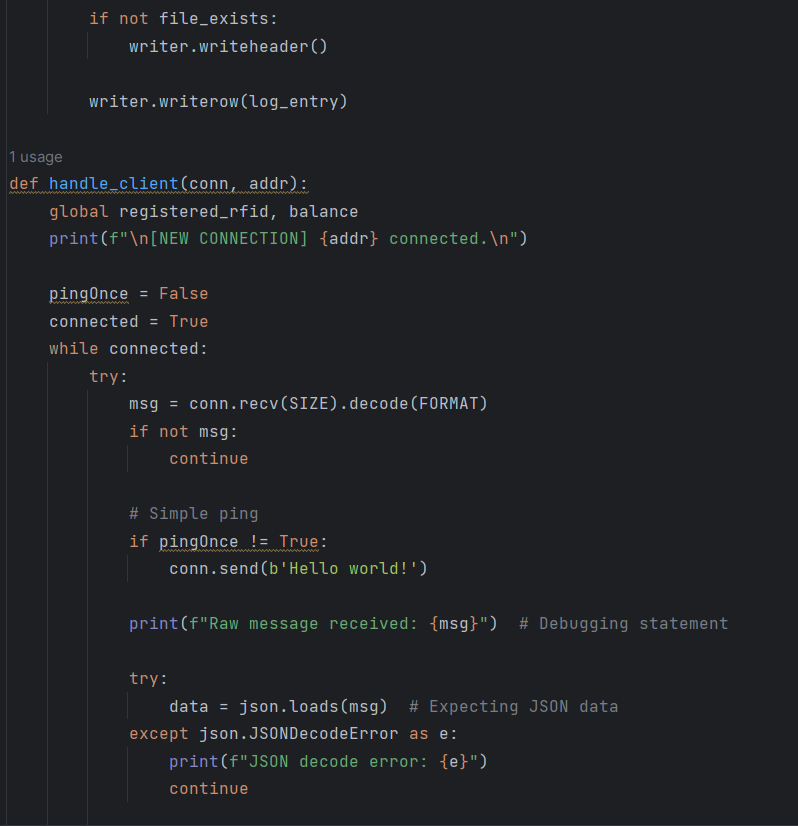
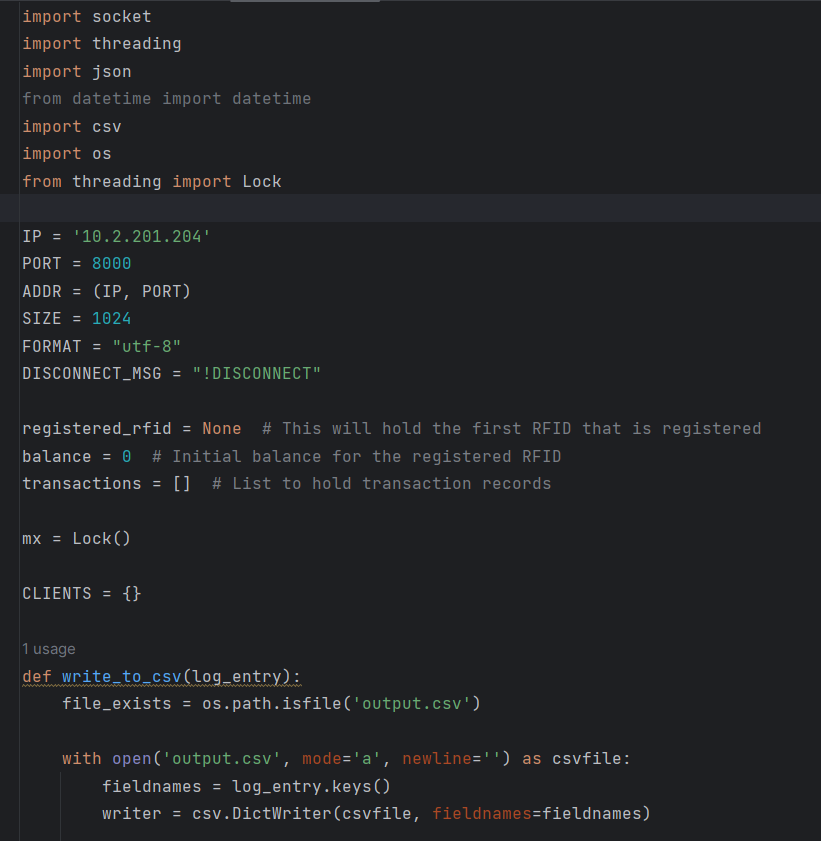


Figure 6 Server.py Code

A screenshot of a computer program

Description automatically generated

Figure 7 Server.py Output in Proxmox Server

Figure 6 shows the server code in which the RFID details are saved. It also contains a CSV writing function in which it logs transactions on a CSV file. The **‘handle\_client()’**  function is used for managing the different client connections. Figure 7 shows the output of the server python file in Proxmox Server wherein we can see the transactions made using the RFID.

# Client Application

The client.py script connects to a server using a socket and interacts with it by sending and receiving JSON-formatted messages. After establishing a connection, the client waits for responses from the server that indicate the status of RFID registrations and transactions. It processes these responses to display appropriate messages, such as confirming successful registration or transactions, notifying about insufficient balance, or indicating invalid RFID. The client also manages user prompts based on the registration status to guide further RFID scans. Figure 8 shows the client.py code that was used for the program:

A screen shot of a computer program

Description automatically generated

Figure 8 Client.py Code

When the script runs, it first connects to the server and prompts the user to scan their RFID for registration. Upon successful registration, the script acknowledges with a balance update and prompts for further scans. If an invalid RFID is scanned, the script provides feedback and requests another scan. Following successful transactions, it displays the updated balance and continues to prompt the user to scan their RFID for entry, reflecting the real-time status updates based on server responses just like on the Figure 9 below.

A screenshot of a computer

Description automatically generated

Figure 9 Client Output