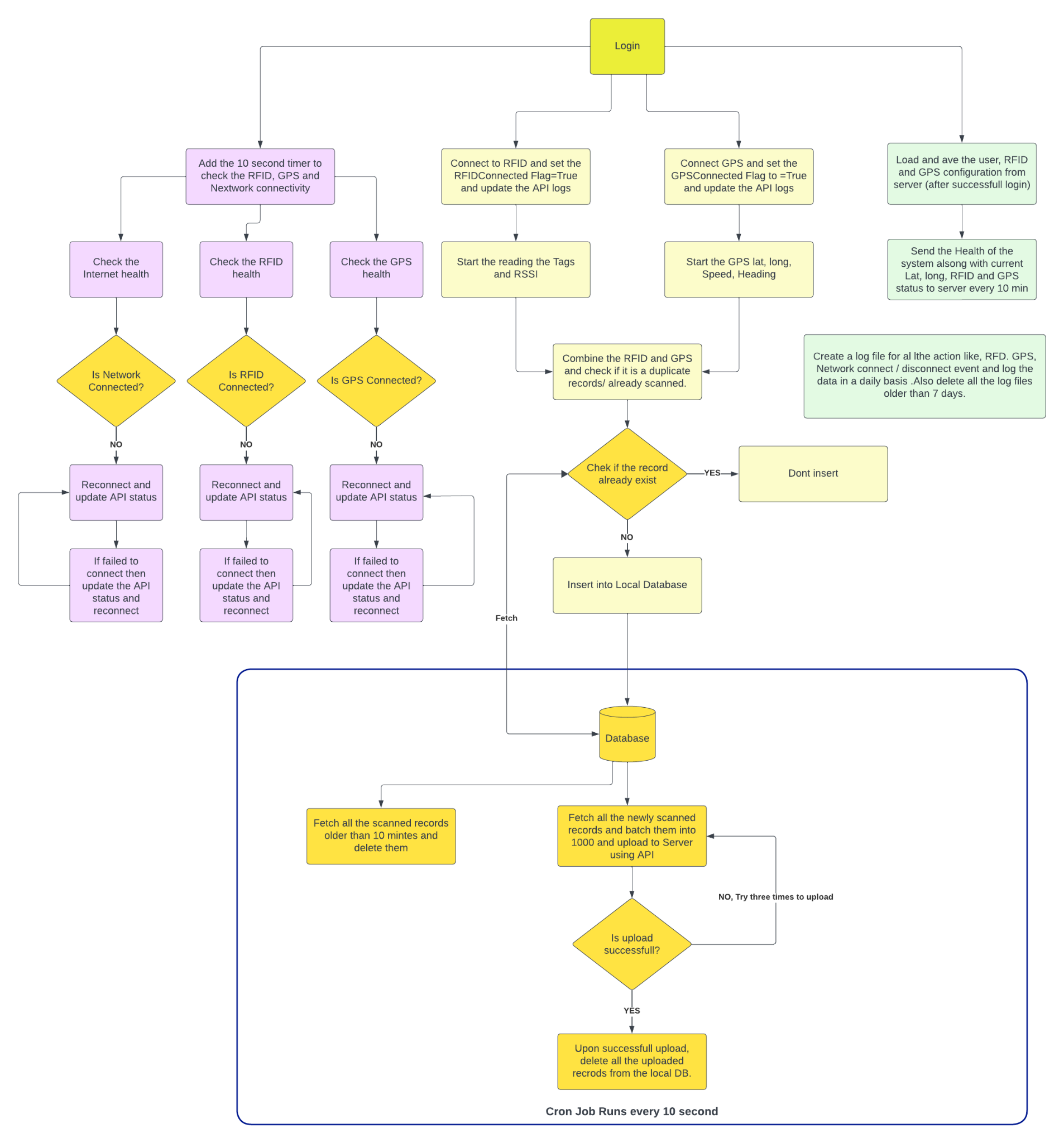
# **RFID Inventory System Application Requirements - Python**

The RFID reader is a very light client application installed on a spotter truck and connected to RFID reader, GPS and internet.

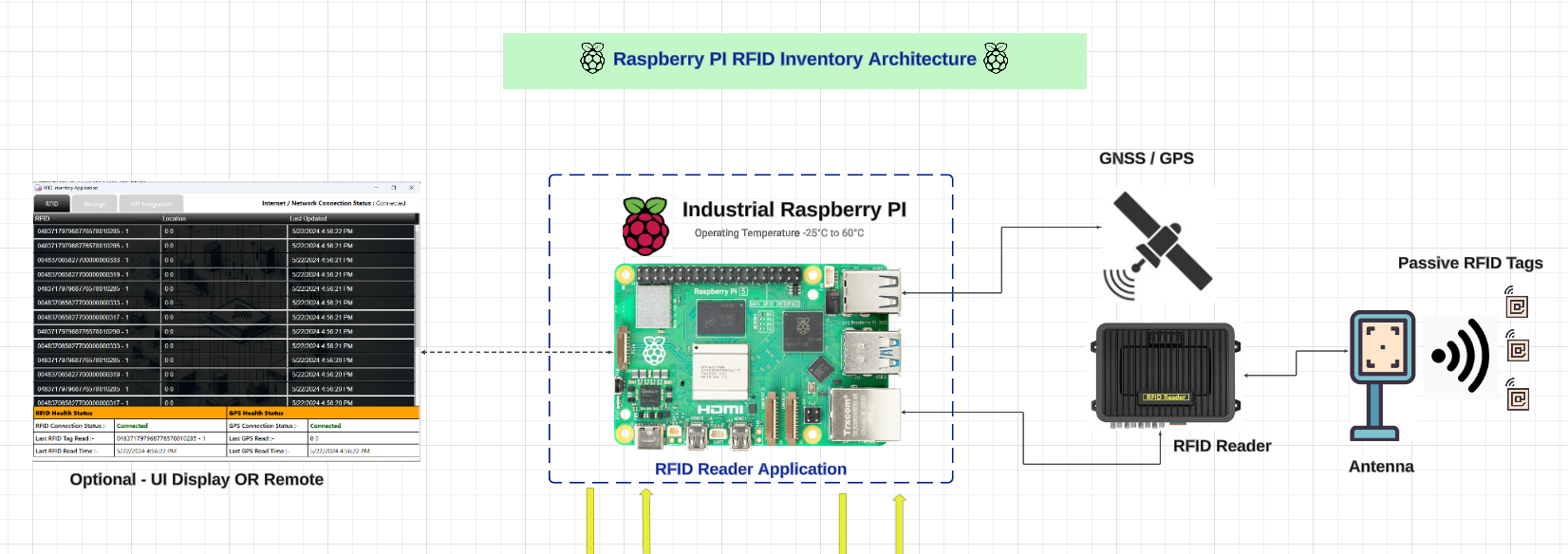
The reader application has a strong logic which automatically connects to the RFID reader and GPS. It also has auto reconnect and disconnect logic built in.

You can see the live Tag reading on the main page with health status information for RFID reader and GPS

**Architecture-**

****

**System Designs- System should be compatible with Raspberry, Ubuntu/Linux, Windows and Mac**

****

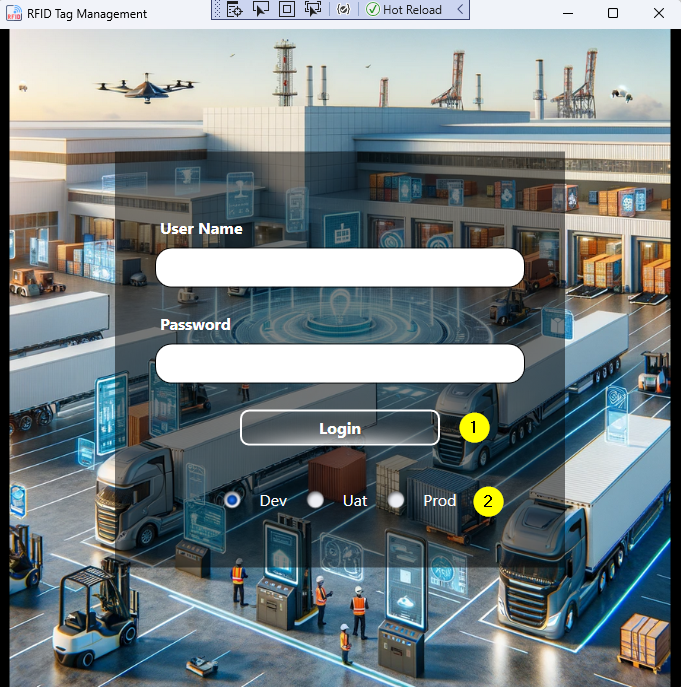
**UI/UX Designs and functional descriptions -**

**\*\*\*\*\*\*\* Login Page -**

1. I will get the login API which will take the username and password and call the Login API.

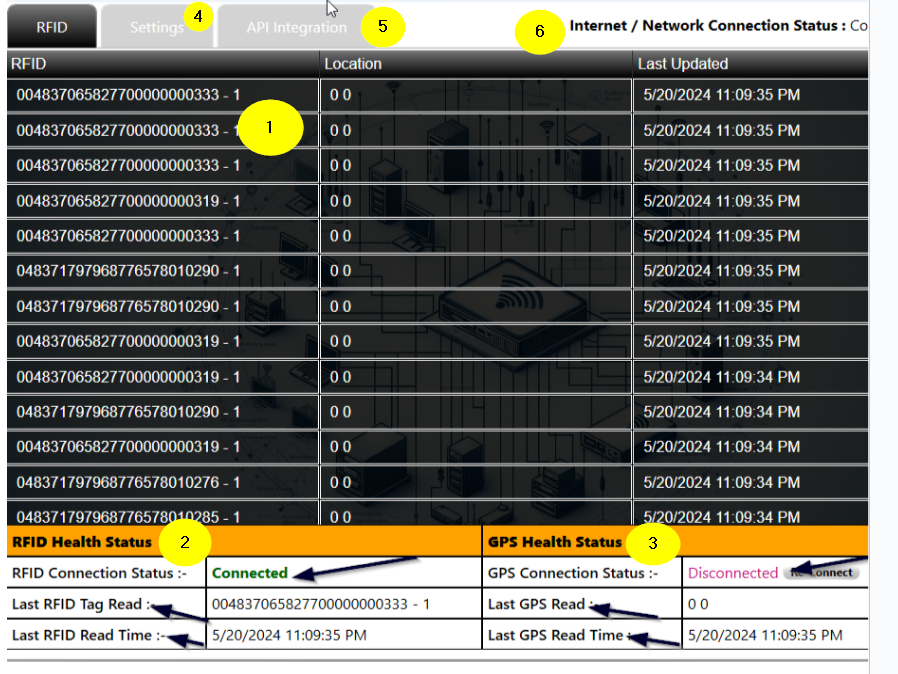
Upon successful login - We will get the token and also all other settings like GPS, RFID(given below) and Environment URL from API server and store into local DB.

1. Based on the environment radio button selection - it will connect to that environment.

****

**\*\*\*\*\*\*\* Home Page -**

* **1. RFID Scan Data Table -**
* Add two more columns to the list below - RSSI and Antenna
* So we will have RFID Tag, Antenna, RSSI, Lan & Long, Read Time.
* This screen will show the live reads from reader, it will show only recent 5-8 rows(based on the screen size)



**2. RFID Health Status -**

* It should be showing the current status of reader connectivity.
* If the reader disconnects somehow, it should keep trying to reconnect by itself (use Keep live event from LLRP).
* Show the last read tag and time

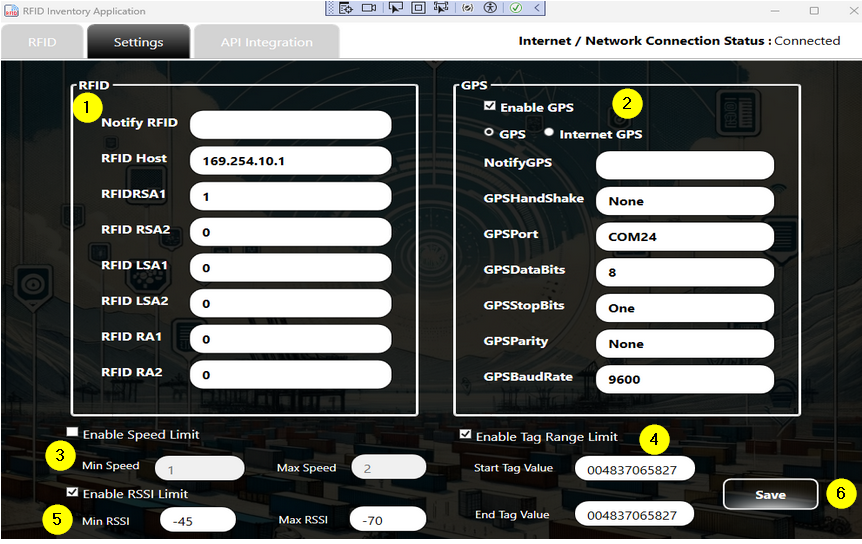
1. **GPS Health Status -**

* It should be showing the current status of reader connectivity.
* If the reader disconnects somehow, it should keep trying to reconnect by itself.
* Show the last GPS read and time

1. **Setting -**

Setting values are mainly for the Reader and GPS configuration. These values can be set up from the Reader Application itself or can also be retrieved from an API server application using the REST API.

After successfully login on RFID Inventory application, in the login API response you will get all the values, if the login API response has these values then bind them to below controls and store them in the local DB.

* 
* **1. RFID**
* *If the notify* value is 1 then the system should produce a beep sound on the RFID Reader connect and disconnect event.(Default value - 1)
* *RFID Host* - IP address of the reader (Default value - 169.254.10.1)
* *RFIDRSA1 to RFID RA* are nothing but the antenna setting (by default all the antennas are setup)
* **2. GPS**
* Enable GPS - If this checkbox is checked then the system should be sending the RFID tags with GPS coordinates, speed to the server.
* **GPS or Intername GPS (Radio Button) -**
* If “GPS” (This external GPS) checkbox is checked then the system should be searching the external GPS connected on the COM port or USB (Raspberry and Linux) system and connected automatically and filled these values. Also give an option to the user to edit below values

Notify GPS - Beep sound on connect and disconnect

HandShake

GPS Port

DataBits

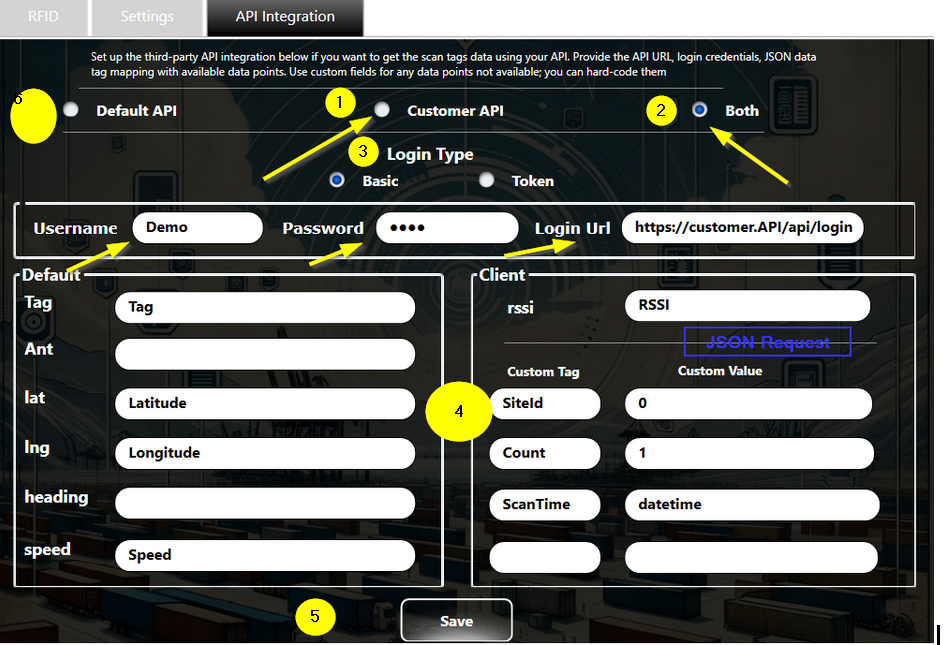
StopBit

Parity

BaudRate

* **3. Enable Speed Limit**
* This is the vehicle speed given by GPS on which the application is running. If this setting is enabled and the vehicle's speed is outside of Min and Max speed value then do not send the Tag reads and GPS data to API server, but it will still be displayed on the Home page.
* **4. Enable Tag Range Limit**
* If this setting is enabled and the readed tag is outside of the given start value and end tag value then ignore tag reads and do not send the data to API server but it will still be displayed on the Home page.
* **5. Enable RSSI Limit**
* This is the signal strength given by Reader on which the application is running. If this setting is enabled and the RSSI is outside of Min and Max value then do not send the Tag reads and GPS data to API server, but it will still be displayed on the Home page.
* **6. SAVE Button**
* *On click of this button all the settings will be saved on the local DB of the application and reader and GPS setting be refreshed as per the new values.*

**3. API Integration-**

****

This section defines the API server integrations.

* **1. Default API**
* This is our company Server API (default) on which we receive the RFID Tag scanned data. If this option is checked then this screen will not show any other options.
* **2. Customer API**
* If this option is checked then we will need to show the 4,5 and 6 the number setting.
* We will need to send the scanned data to only the customer API.
* **3. Both -**
* If this option is selected then scanned data will be sent to customer and our APIs
* **5. Both -**
* Select the Basic or Token -
* For Basic - show username and password with the URL
* For Token - only show one textbox for JWT token
* Default - These are the JSON values we have available in our RFID application.
* Client - We will need to be given an option to map their API values.
* **6. SAVE -**
* *On click of this button all the settings will be saved on the local DB of the application and reader and GPS setting be refreshed as per the new values.*

**Here is a formatted version of your points for clarity:**

Optimize Tag Reading Process: The tag reading process should be faster and aligned with the RFID reader hardware to avoid delays.

Temporary Database Storage: Store all scanned data in a temporary database. Implement a separate process to filter out duplicates and sync the filtered data with the server.

Isolate RFID Reader Library: Separate or isolate the RFID reader library to ensure full control over the code.

Real-Time RFID Reader Status Check: Continuously check the RFID reader's connection status in real-time and implement automatic reconnection if it disconnects for any reason.

GPS Configuration and Auto-Connect: Ensure GPS settings are configurable, and the auto-connect function works as intended.

Data Synchronization: Ensure that both RFID scanned data and device health data are synced with the server without errors.

Settings Functionality: Verify that all sections of the settings are working as designed, as there were previous issues with certain sections not functioning properly.

Multi-Platform Compatibility: The application should be fully operational on Windows, Raspberry Pi/Ubuntu, and Mac platforms.

Scalability and Responsiveness: The application should be scalable and responsive across all screen resolutions, from 5-inch to 32-inch displays.

Build and Installer Creation: Define and implement the process for building and creating installers for Windows, Raspberry Pi/Ubuntu, and Mac.

Auto Start on computer power on: - Once an application is installed on a device we will need to give an option to start the application automatically on system reboot or restart. Application should be reading the setting data from local storage.

Error Logging: Implement proper error logging for all application events, including RFID connection/disconnection, GPS connection/disconnection, and data sync processes.

Email Alerts for Connectivity Issues: Implement email alerts and notifications for prolonged disconnection events (details to be provided later).