Department of Electronic & Telecommunication Engineering,
University of Moratuwa,
Sri Lanka.

Design Report
Handheld RFID Reader

Group Members

Index No	Name
220235V	Ilukkumbura I.M.E.I.B.
220212A	Hapuarachchi H.A.D.N.D.
220162T	Fernando C.S.R.
220221B	Hathurusingha HA.R.
220420J	Nawarathne MA.A.K.
220700T	Wickramasinghe S.D.
220089B	Cooray M.S.T.
220163X	Fernando D.S.

Submitted in partial fulfillment of the requirements for the module
EN 2161 Electronic Design Realization

5/21/2025

Contents

1 Introduction

Handheld RFID readers are portable electronic devices that wirelessly identify and track RFID tags attached to objects. These readers enable users to scan tags while on the move, making them highly suitable for applications that require mobility, such as inventory audits, asset tracking, supply chain management, and field service operations. Handheld readers can support various RFID frequency bands, including low-frequency (LF), high-frequency (HF), and ultra-high-frequency (UHF), each catering to different use cases and read range requirements.

High-frequency (HF) handheld RFID readers operate at 13.56 MHz and are typically used for applications that demand short to medium read ranges, usually up to 1 meter. They are well-suited for environments requiring secure and accurate data exchange, such as library management, ticketing, contactless payment systems, and access control. HF readers often support ISO standards like ISO 14443 and ISO 15693, enabling compatibility with a broad range of RFID tags and smart cards. With their balance of portability and reliability, HF handheld RFID readers offer an effective solution for controlled, close-range scanning tasks. This report steps we followed up to now to design an HF RFID reader, with detailed information.

2 Review Process

2.1 Existing Products in the Market

Some handheld readers in the current market, similar to what we are developing, are given below.

2.1.1 RFD8500 Series by Zebra Technologies

Figure 1: RFD8500 UHF RFID Reader

The Zebra RFD8500 is a high-performance UHF RFID and barcode reader that easily adds advanced data capture capabilities to modern mobile devices. Compatible with Android™, iOS™, and Windows™, it supports a wide range of smartphones, tablets, and handhelds, offering flexible deployment as a Bluetooth-enabled sled that can be attached or used standalone. Designed for fast and accurate tag reading, the RFD8500 features a patent-pending antenna, AutoMac[1] technology, and a high-capacity battery for full-shift operation. Its support for both real-time Bluetooth connectivity and offline batch mode ensures efficient data collection in any environment. Easy to deploy with Quad Lock mounts and custom adapters, it provides a versatile, future-ready solution for enhancing mobile inventory management and workforce productivity.

2.1.2 FRD9750 HF Handheld Reader by FAREAD

Figure 2: FRD9750 HF Handheld Reader

This industrial-grade handheld RFID mobile terminal combines high-performance data capture with rugged mobility, which is ideal for demanding environments. Running Android 11.0 and equipped with a powerful 8-core processor, it features 4G, Wi-Fi, and Bluetooth connectivity, a 6.21-inch glove-friendly touchscreen, and dual batteries (6700mAh removable + 4500mAh built-in) for continuous, high-intensity operation. Its HF RFID reader supports ISO 15693 and ISO 18000-3M1 protocols, offering read distances up to 32 cm and a speed of 50 tags per second, thanks to its sensitive antenna and anti-collision algorithm. Designed for applications such as asset tracking, book management, and warehouse inspection, the device also supports 1D/2D barcode scanning, has a rugged IP67-rated build, expandable storage up to 1TB, and comes with a free SDK for easy integration into custom Android applications.

2.1.3 CS108 RFID Sled Handheld Reader by Covergence Systems Limited

Figure 3: CS108 RFID Sled Handheld Reader

Figure 4: CS108 RFID Reader - Charging cradle

The CS108 RFID Sled Handheld Reader is a high-performance UHF RFID reader and barcode scanner designed to seamlessly integrate smartphones and tablets via Bluetooth. Developed by Convergence Systems Limited (CSL), the CS108 delivers exceptional read range and tag reading speed, making it one of most powerful RFID sled readers in its class. It supports Android and iOS platforms, transforming standard mobile devices into enterprise-grade data collection tools. With its ergonomic design, robust build quality, and long battery life, the CS108 is ideal for inventory

management, asset tracking, supply chain, retail, and field service applications. Its advanced RFID engine ensures reliable performance in even the most demanding environments, making it a preferred choice for businesses seeking mobility without compromising capability.

We initially selected RFD8500 as our reference design, then transitioned to the FRD9750 reader because it uses HF technology. Finally, we decided to move on with the CS108 RFID Sled Handheld Reader.

2.2 Industrial Applications

1. **Inventory Management:** Track and manage stock levels in warehouses, manufacturing plants, and retail environments.
2. **Asset Tracking:** Locate and monitor high-value equipment, tools, and machinery in construction, oil, and gas industries.
3. **Warehouse and Logistics:** Streamline receiving, picking, packing, and shipping through real-time item identification.
4. **Manufacturing:** Monitor parts and components throughout production for proper assembly and traceability.
5. **Supply Chain Management:** Improve visibility and traceability of goods from raw materials to finished products.
6. **Field Service and Maintenance:** Identify equipment, log maintenance tasks, and verify service records on-site.
7. **Retail:** Increase inventory accuracy, optimize restocking, and support omnichannel strategies.
8. **Transportation and Logistics:** Monitor shipments, verify cargo, and manage fleet operations effectively.

3 Stakeholder Mapping

The stakeholders of handheld RFID readers can be mainly divided into 4 parts.

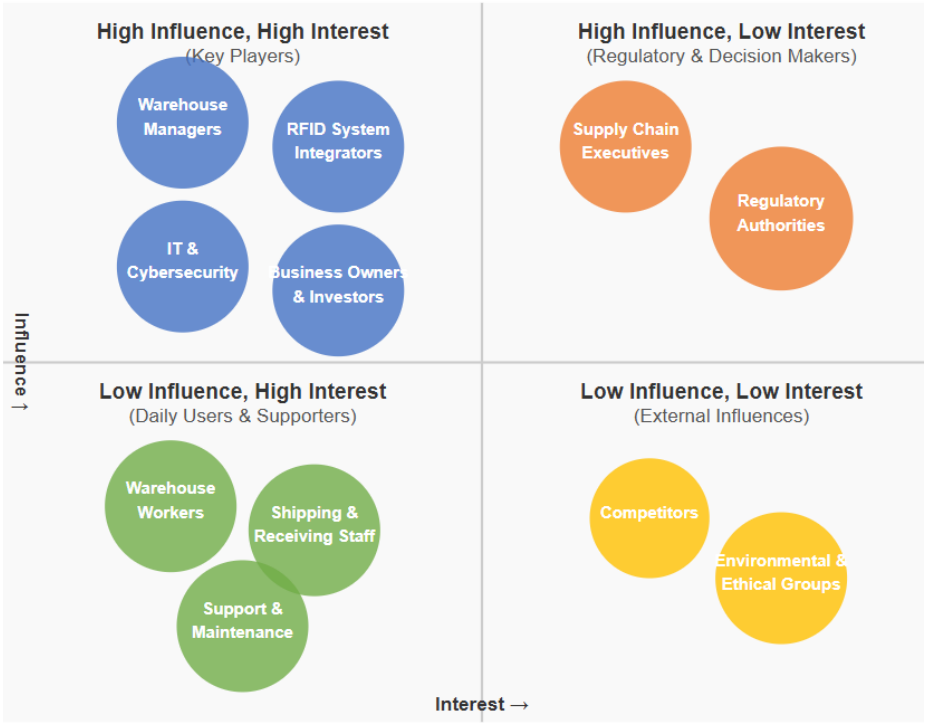


Figure 5: Stakeholder Map

4 Observing Users

4.1 Industry Visit

We got the chance to visit Hayleys Free Zone Ltd in Biyagama. They use barcode readers there, but not RFID readers. They have a database management system, but they do not use it. We realized we have a potential market in companies like that if we develop our product well. We observed charging cradles used to charge barcode readers. That is where we got the idea to design a charging cradle. The field visit was highly insightful to us in many ways.

Figure 6: Industry Visit to Hayleys

Figure 7: A Barcode Reader Hayleys Uses with the Charging Cradle

5 User Need Analysis

Category	Stakeholders, Roles, and Needs
High Influence, High Interest	**Warehouse Managers** – Oversee inventory management, ensure efficiency, and make purchase decisions. Need accurate and real-time tracking, seamless integration with WMS, and user-friendly devices. **RFID System Integrators** – Ensure smooth integration with warehouse management systems (WMS). Need compatibility with various systems, strong technical support, and minimal downtime. **IT and Cybersecurity Teams** – Maintain data security, prevent breaches, and ensure system reliability. Need strong encryption, secure data transmission, and compliance with security standards. **Business Owners and Investors** – Fund technology adoption and expect ROI through improved efficiency. Need cost-effective solutions, scalability, and measurable impact on productivity.
High Influence, Low Interest	**Supply Chain Executives** – Focus on overall logistics optimization and compliance. Need data-driven insights, standardized processes, and regulatory compliance. **Regulatory Authorities** – Ensure RFID devices comply with global frequency and data standards. Need adherence to legal and operational standards, proper testing, and documentation.
Low Influence, High Interest	**Warehouse Workers and Inventory Handlers** – Use handheld RFID devices for scanning and tracking goods. Lightweight, ergonomic, and easy-to-use devices with minimal training are required. **Shipping Receiving Staff** – Ensure accurate inbound and outbound logistics. Need fast scanning capabilities, real-time inventory updates, and error-free operations. **Support and Maintenance Teams** – Provide technical support, troubleshoot, and update firmware/software. Need accessible troubleshooting tools, detailed documentation, and reliable spare parts.
Low Influence, Low Interest	**Competitors** – Influence industry trends and push for innovation. Need awareness of technological advancements, market trends, and differentiation strategies. **Environmental and Ethical Groups** – Monitor sustainability and data privacy concerns related to RFID use. Need eco-friendly device options, responsible data handling, and compliance with sustainability policies.

Table 1: Roles and Needs of Stakeholders

6 Idea Simulation

When we initially started discussing, our idea was to design a UHF RFID reader because they are used in industry mostly, and have a long reading range. After brainstorming, we finalized our conceptual design: a gun-shaped handheld device with an

easy-to-hold handle, a strip to prevent falling, a 12-hour rechargeable battery, a mobile phone as the user interface connected to the device via WiFi, and a data database via Bluetooth.

Then we got the chance to visit Hayleys Free Zone Ltd in Biyagama. They use barcode readers there, but not RFID readers. That is where we got the idea to design a charging cradle.

After researching the circuit designs of RFID readers, we realized that it is hard to implement a circuit for a UHF RFID reader at our level. Then we decided to design an HF RFID reader, though it has a short reading range. We also decided to buy an in-built antenna module rather than designing it ourselves because it is too hard.

7 Conceptual Design

7.1 Enclosure

Below are some of the initial sketches created by our team members.

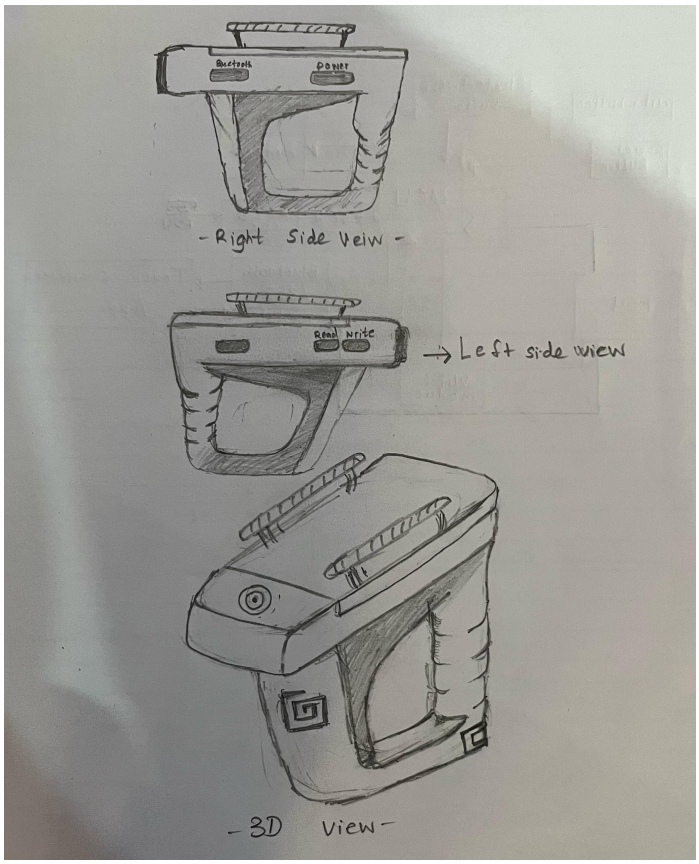


Figure 8: An Initial Sketch

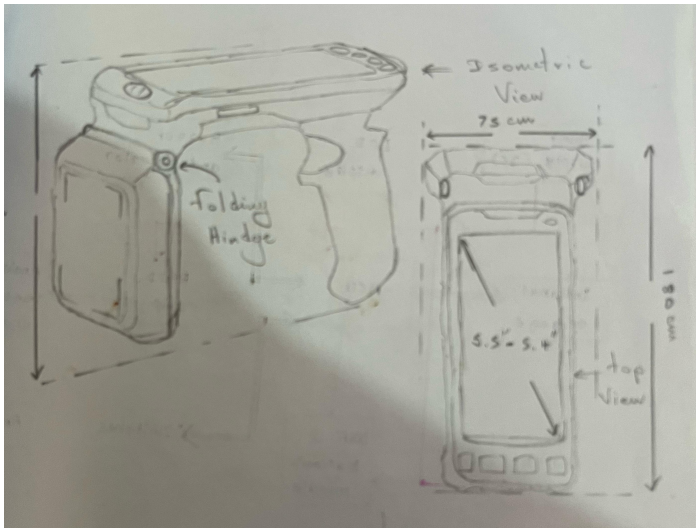


Figure 9: An Initial Sketch

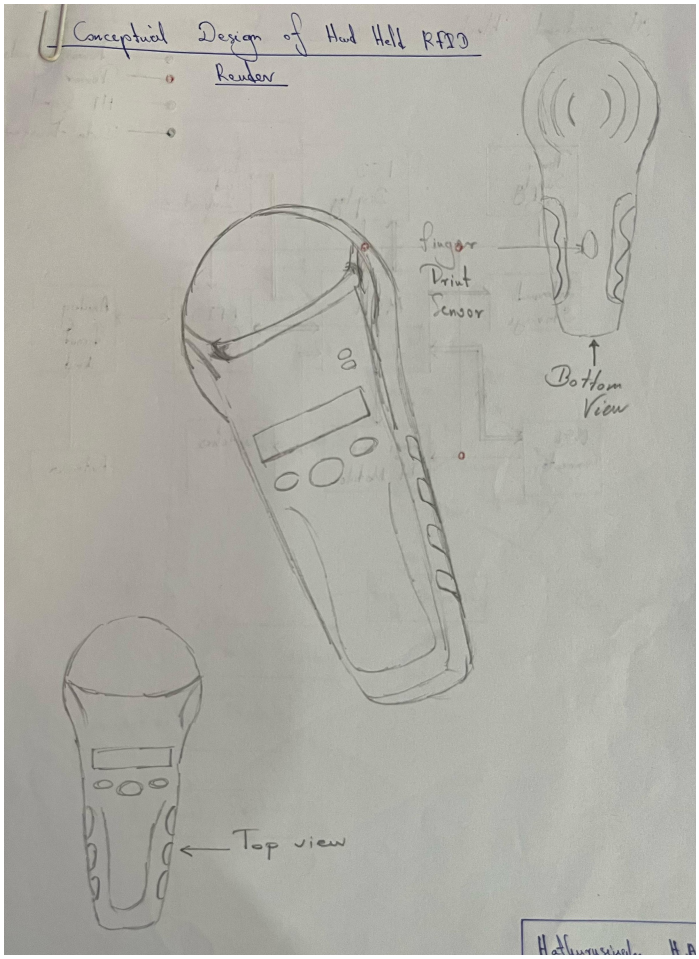


Figure 10: An Initial Sketch

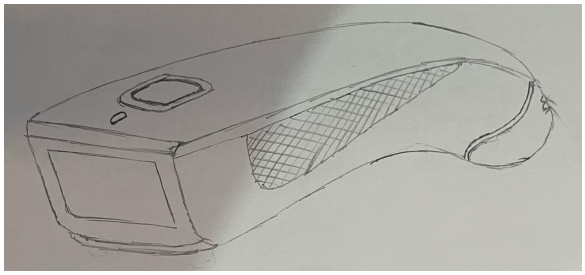


Figure 11: An Initial Sketch

7.2 Block Diagram

Given below are some of the initial block diagrams drawn by our team members.

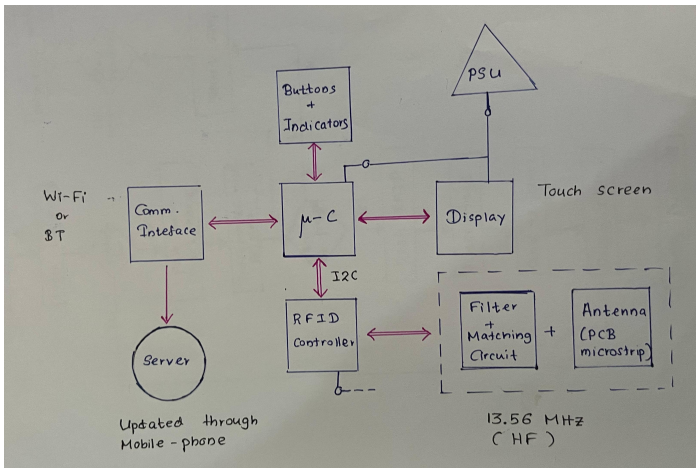


Figure 12: An Initial Block Diagram

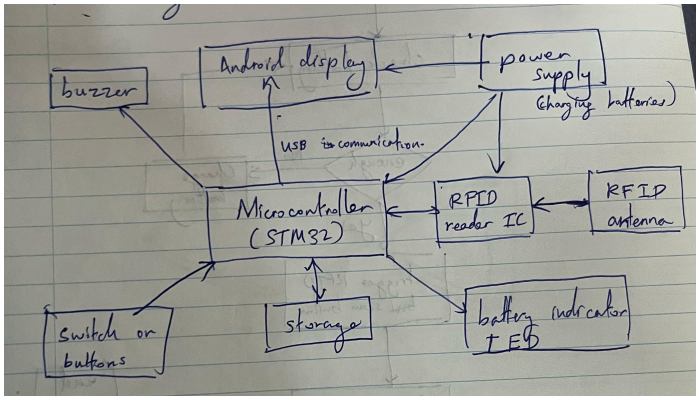


Figure 13: An Initial Block Diagram

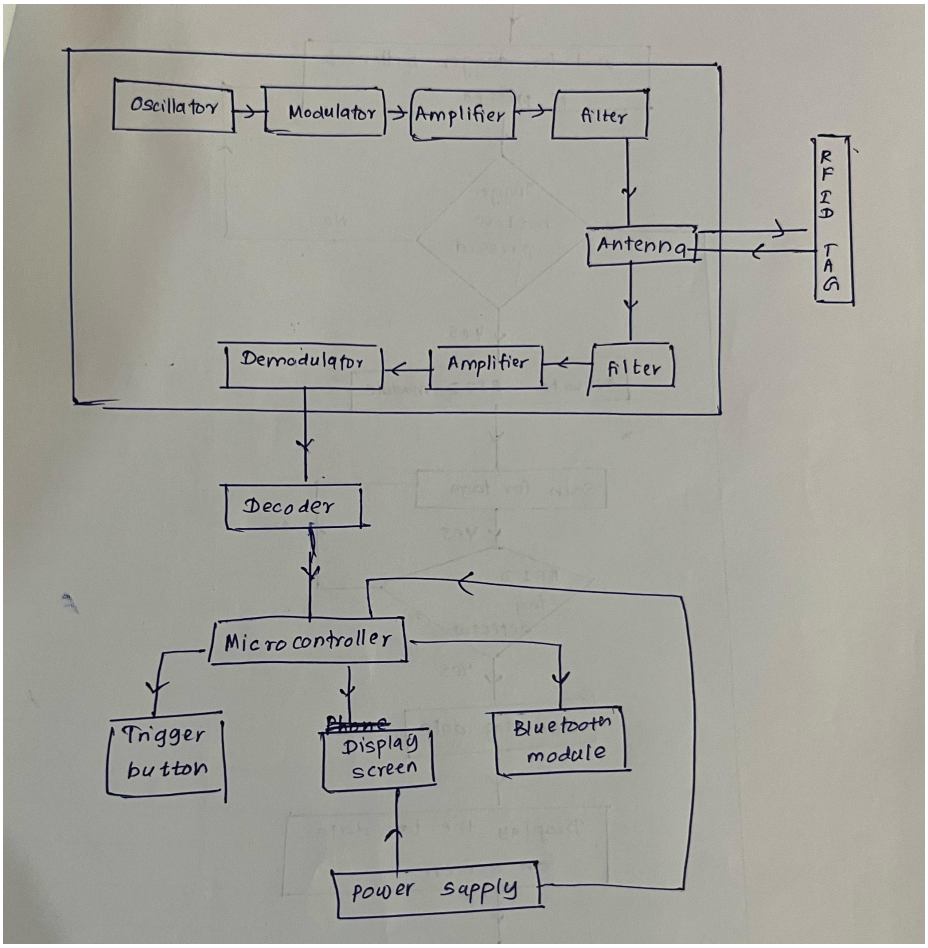


Figure 14: An Initial Block Diagram

7.3 Flow Chart

Below are some of the initial flow charts drawn by our team members.

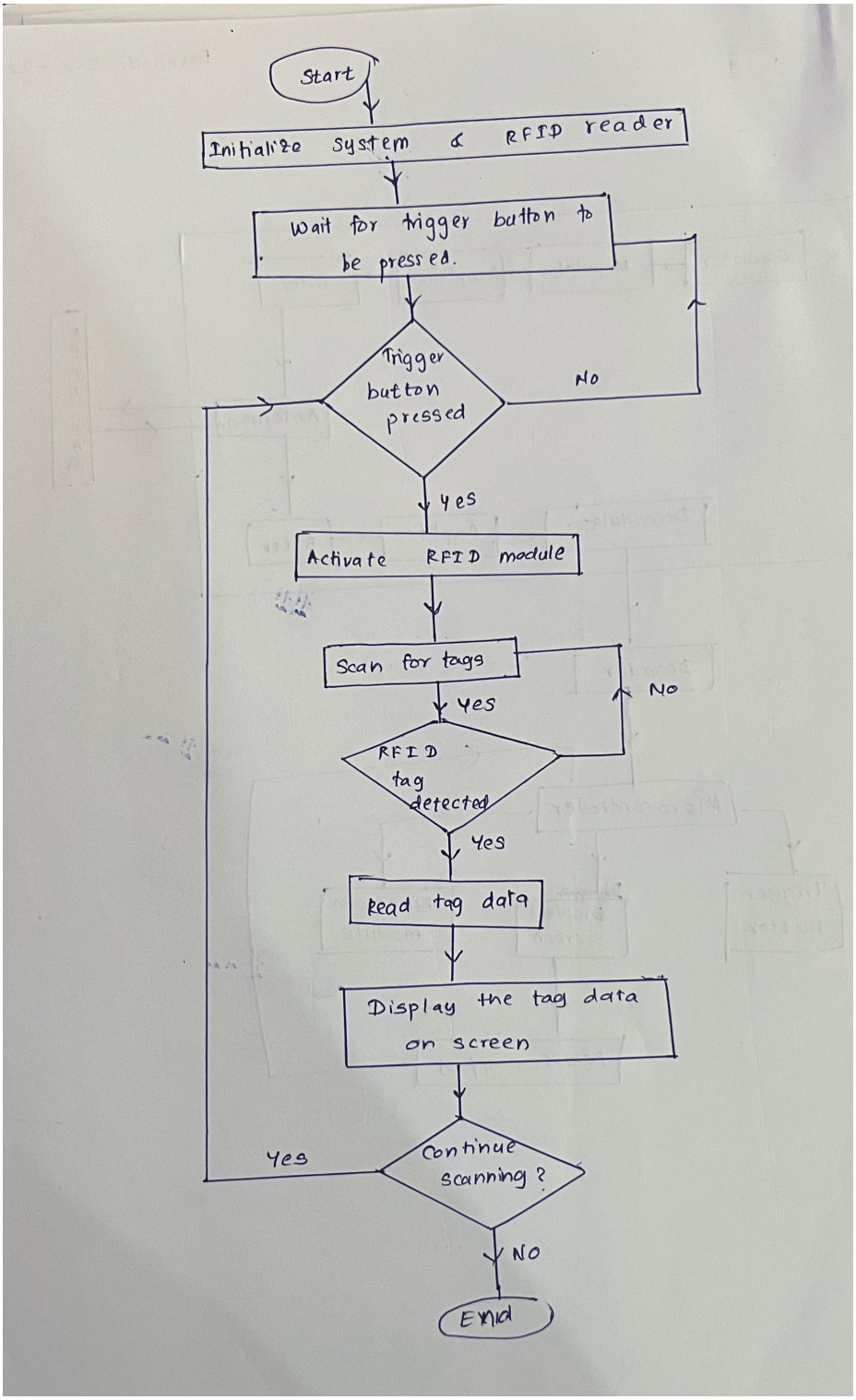


Figure 15: An Initial Flow Chart

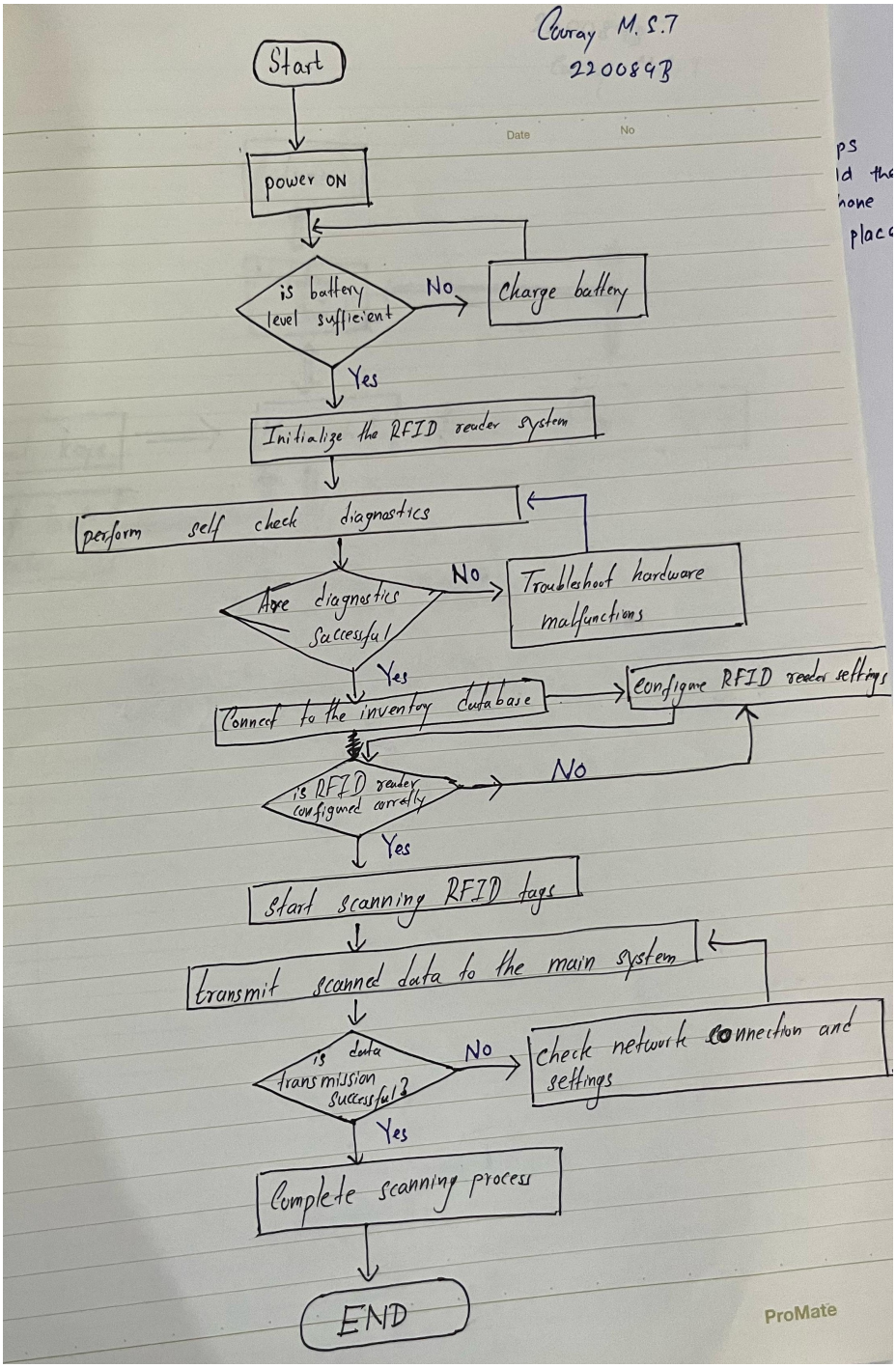


Figure 16: An Initial Flow Chart

7.4 Evaluation

We evaluated our enclosure designs, block diagrams, and flowcharts according to the users' needs.

Category	User needs	Nuwan	Induwara	Sanugi	Agra	Rivikula	Achintha	Dahami	Sahan	Combined Disgn 1	Combined Design 2	Combined Design 3
Enclosure	Screen solution (Phone on device, On screen) -8	4	6	8	8	6	6	8	6	Mobile phone	Screen	No screen
	Ergonomic Design (Lightweight, Easy to grip)	3	7	6	7	6	7	7	7	Gun type	Gun Type	Wand Type
	Robustness	9	7	6.5	7	6.5	7	7	7	Rugged Design	Rugged Design	Rugged design
	Input keys (For power on, reading)	7	6	8	8	8	8	4	6	3 keys (Power, read,bluetooth)	2 Buttons (Powe, Tigger)	3 keys (Power, read,bluetooth)
	Hand attachment (Straps, rough handle)	10	10	0	10	0	0	0	0	Strap present	Strap present	Strap Present
	Battery level display (LEDs, On screen display)	10	7	6	6	6	6	6	7	LED Battery level display	Screen Displays Bttery Level	LED Battery Level
Electronics (Block Diagram)	Battery (Rechargeable, Replacable)	9	4	4	4	4	4	4	4	Rechargable type c (12 hours)	Reacharchable	Rechargable Type C
	Feedback	8	8	0	0	4	4	0	6	Buzzer Sound Only	Buzzer Sound Only	Sound+ Heptic feedback
Softwear (Block Diagram)	Integration with mobile	7	8	9	9	8	8	9	8	(RFID-Mobile-Database)9	Device - Cloud Update	Device - Cloud updated
	User Expereince	4	8	9	9	8	8	9	8			
Overall		71	71	56.5	68	56.5	58	54	59			

Figure 17: Conceptual Design Evaluation

7.5 Final Selection

Here are the finalized enclosure, block diagram, and flow chart.

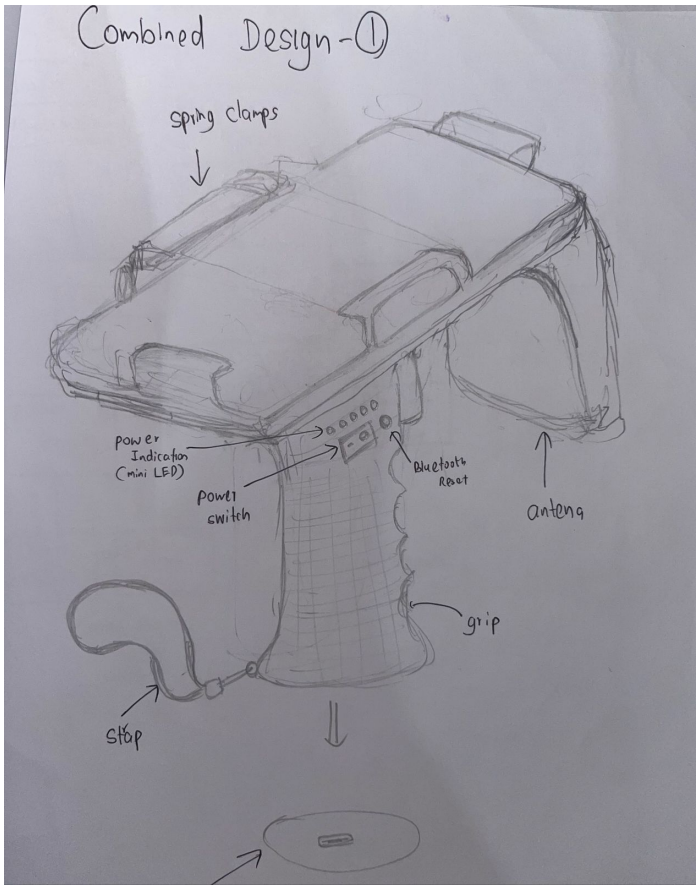


Figure 18: Finalized Enclosure Design

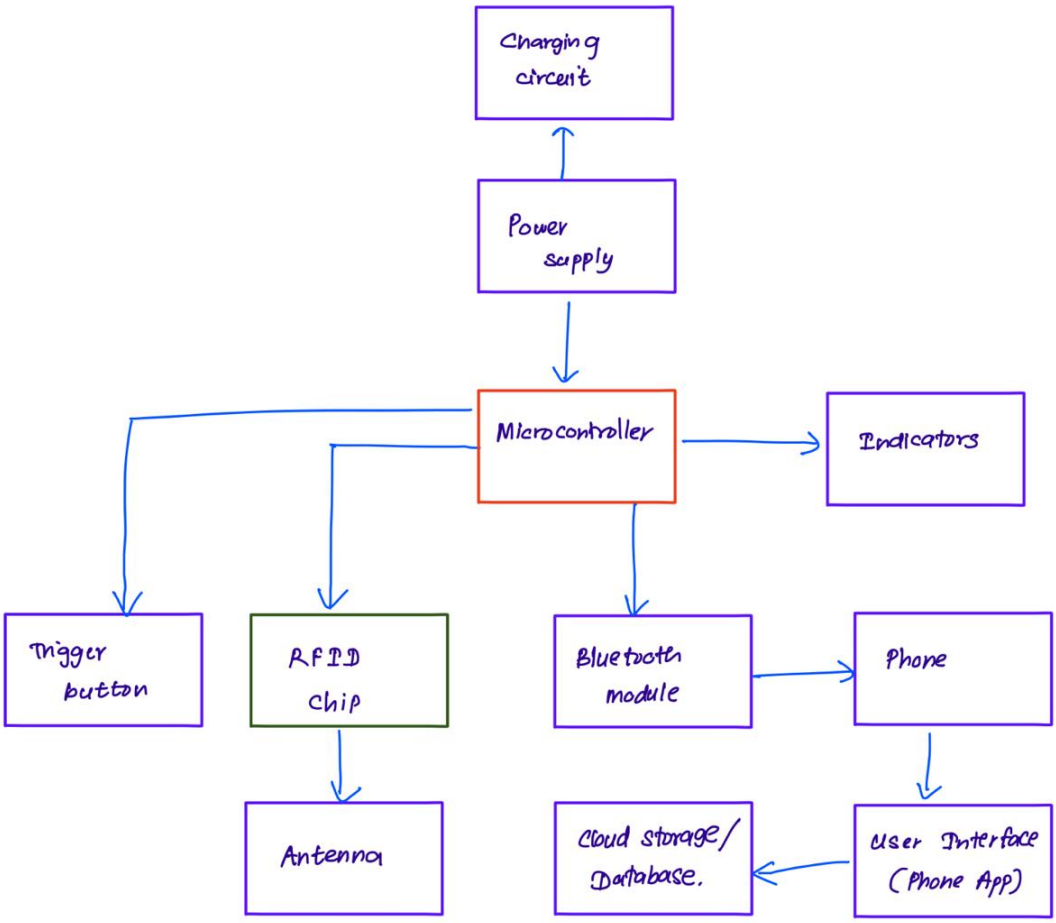


Figure 19: Finalized Block Diagram

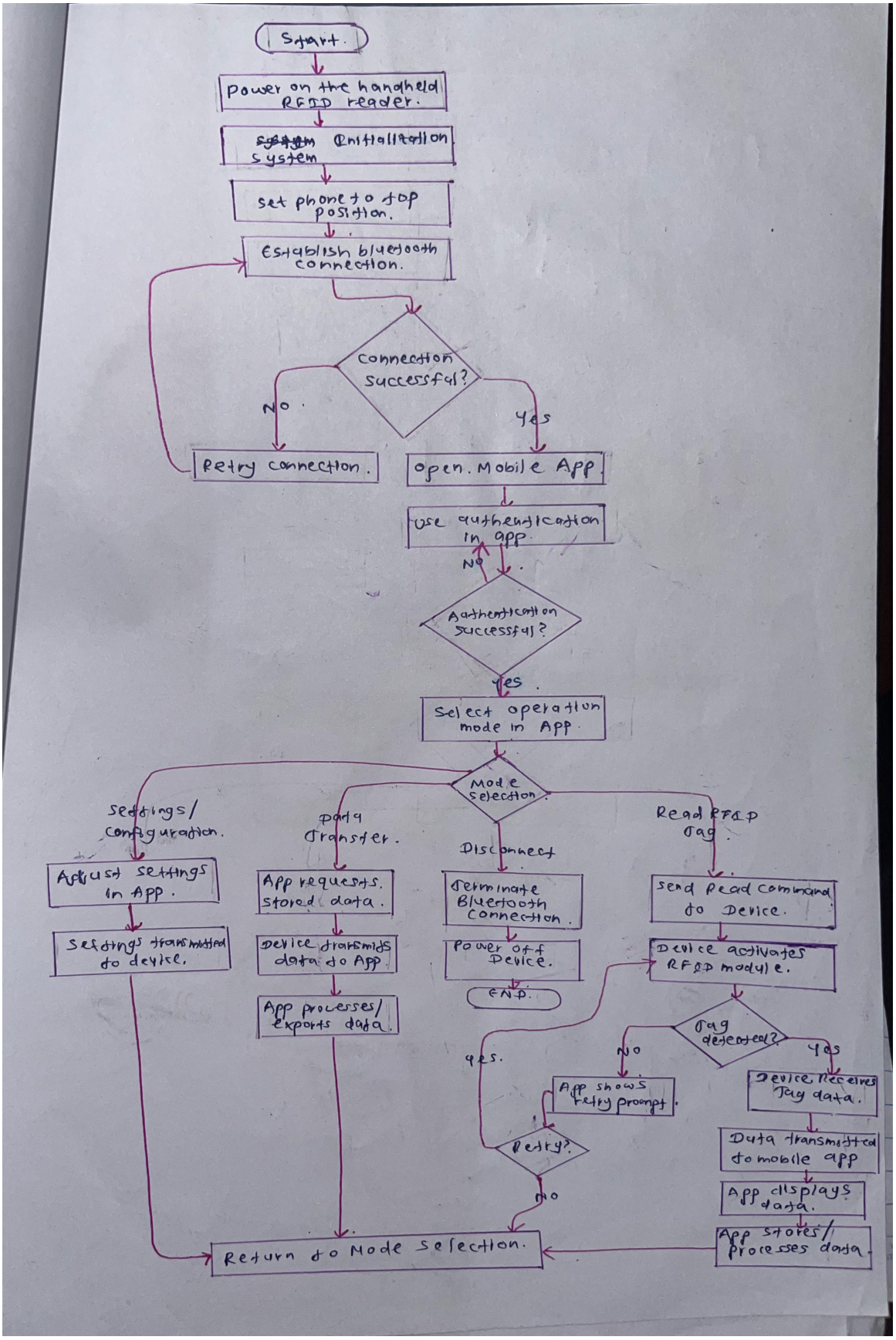


Figure 20: Finalized Flow Chart

Stage 1

8 Component Selection

8.1 RFID Module

Our requirements for a suitable HF RFID module are as follows:

Parameter	**Requirement**
Frequency	13.56 MHz
Power Consumption	Medium power consumption
Supply Voltage	Must align with the power we can give
Read Range	Should be high
Data Transfer Rate	Higher data transfer rate
Memory Capacity	Should be high
Temperature Range	Should operate in the typical room temperature range
Standards Compliance	Should support ISO15693
External Communication Interfaces	SPI, I^2C
Cost	$< \$10$
Package	Should be small

Table 2: Requirements for HF RFID Module

We selected a few RFID chips and evaluated them.

RFID Chip	Frequency	Read range	Data transfer rat	Standards compliance	External communicatio n interfaces	Cost	Resource Availability	Total	
NXP CLRC663 NFC	10	4	10	10	10	8	8	60	Alternative
TI TRF7970A	10	2	10	10	10	9	8	59	
STCR95HF	10	2	10	10	10	10	8	60	
ST25R3916	10	8	10	10	10	8	10	66	
ST25R3916B	10	8	10	10	10	10	2	60	
ST25R3911B	10	5	5	10	4	3	10	47	
PN532/C1	10	3	5	3	10	10	8	49	
PN5180A0xx/C3,C4	10	5	10	10	10	8	8	61	Selected

Figure 21: Evaluation Table - MCU

According to the table, we selected PN5180 as our RFID module.

8.2 Micro-controller

The following choices were compared to select a better MCU

MCU	Clock Speed	Flash Memory	I/O Pins	USB Support	Peripherals	Power Consumption	Market Availability	Programmability	Hobbyist/ Professional	Total Score
ATmega32U4	16 MHz (8)	32 KB (8)	26 (10)	Native (10)	UART/SPI/I2C (8)	Low (8)	Common (9)	easy (10)	Professional (8)	8.86
ATmega328P	16 MHz (8)	32 KB (8)	23 (8)	None (5)	UART/SPI/I2C (8)	Low (9)	Very common (10)	easy (10)	Mostly hobby (4)	7.97
SAMC21G18A	48 MHz (10)	256 KB (10)	38 (10)	Native (10)	Multiple SERCOM	Low (9)	Moderate (7)	moderate (7)	Professional (10)	8.83
AT90USB1286	16 MHz (8)	128 KB (9)	32 (9)	Native (10)	UART/SPI/I2C (8)	Moderate (7)	Less common (6)	moderate (8)	Decent (7)	7.92
ESP32	240 MHz (10)	4 MB (8)	34 (9)	Native (10)	3 UART/2 SPI/2 I2	High (5)	Common (9)	easy (10)	Mostly hobby (4)	7.88
STM32F103C8T6	72 MHz (9)	64 KB (8)	37 (9)	Native (10)	3 UART/2 SPI/2 I2	Moderate (7)	Common (8)	moderate (8)	Less professional (6	7.87
Notes:										
Scores: 1 (poor) to 10 (excellent), based on our requirments.										
Weighted Total: Calculated as (Clock × 0.09) + (Flash × 0.09) + (I/O × 0.18) + (USB × 0.09) + (Peripherals × 0.09) + (Power × 0.135) + (Availability × 0.09) + (Programmability × 0.135) + (Hobby/Pro × 0.10).										
Sources: Data compiled from manufacturer datasheets (Microchip, Espressif, STMicroelectronics), Arduino documentation, and community resources.										

Figure 22: Evaluation Table - Bluetooth Module

After careful selection, Atmega32U4 was selected as the MCU

8.3 Bluetooth Module

The evaluation table for the Bluetooth module is given below.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Chip	BLE Support (Score)	Power Consumption (Score)	Interface (S	PIO (Score	of Development (S	Cost (Score)	Size (Score)	urces Flash/RAM (Sc	ist/Professional (S	Score (Weighted)	
2	nRF52805	5.2 (9)	Low (9)	Yes (8)	10 (8)	High (9)	~$2–$3 (9)	Very small (10)	192 KB / 24 KB (10)	Professional (10)	9.05	
3	DA14531	5.1 (8)	Ultra-low (10)	Yes (8)	12 (9)	Moderate (6)	~$1–$2 (10)	Very small (10)	144 KB / 32 KB (10)	Professional (9)	8.8	
4	CC2640R2F	5.1 (8)	Low (7)	Yes (8)	31 (4)	Moderate (6)	~$2–$4 (8)	Small (9)	128 KB / 28 KB (10)	Professional (9)	7.5	
5	EFR32BG22	5.2 (9)	Low (9)	Yes (8)	26 (5)	Moderate (7)	~$3–$5 (7)	Small (9)	352 KB / 32 KB (8)	Professional (10)	7.95	
6	nRF52820	5.2 (9)	Low (8)	Yes (8)	18 (8)	High (8)	~$3–$4 (7)	Small (9)	256 KB / 32 KB (8)	Professional (10)	8.25	
7	ESP32-C3-MINI	5.0 (7)	Moderate (4)	Yes (8)	22 (7)	Very High (10)	~$2–$3 (9)	Small (8)	400 KB / 8 KB (7)	Mostly Hobby (4)	7.3	
8	nRF52840	5.0 (7)	Moderate (6)	Yes (8)	48 (4)	Very High (9)	~$5–$7 (6)	Small (7)	1 MB / 256 KB (6)	Professional (10)	7.25	
9	**Notes:**											
10	**Scores: 1 (poor) to 10 (excellent), For our requirments**											
11	**Weighted Total: Calculated as (Power × 0.25) + (Cost × 0.2) + (Ease × 0.2) + (Hobby/Pro × 0.15) + (Size × 0.1) + (BLE + UART + GPIO + Resources) × 0.025 each.**											
12	**Sources: Data compiled from manufacturer datasheets.**											
13												

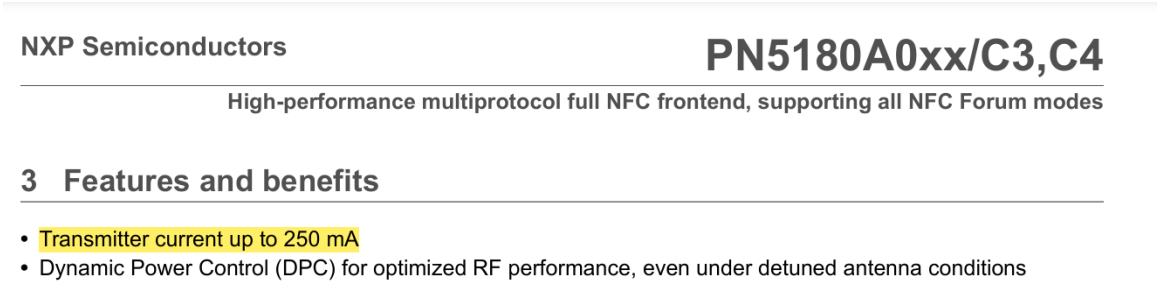
Figure 23: Evaluation Table - Bluetooth Module

After careful selection, NRF52805 was selected as the Bluetooth module.

- Highly affordable, making it ideal for budget-sensitive projects such as student or prototype developments.
- Features built-in 802.11 b/g/n Wi-Fi capability, supporting station and access point modes without additional components.
- Fully compatible with the Arduino IDE and UART communication, simplifying programming and integration with microcontrollers like the ATmega32u4.
- Small form factor is suitable for handheld or embedded systems, conserving space in compact enclosures.
- Offers multiple power-saving modes, making it suitable for battery-powered devices.
- Supports MQTT, HTTP, and TCP/IP protocols, enabling reliable communication with cloud services or local networks.
- Backed by a large developer community with abundant tutorials, libraries, and troubleshooting resources.

8.4 Battery

Calculations were done considering the maximum current drawn from the components.



NXP Semiconductors

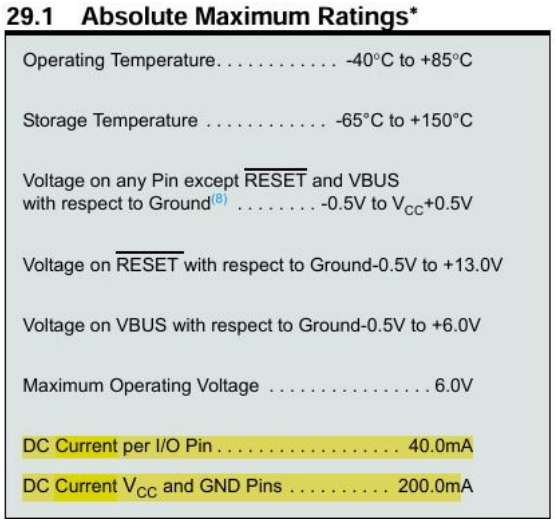
PN5180A0xx/C3,C4

High-performance multiprotocol full NFC frontend, supporting all NFC Forum modes

3 Features and benefits

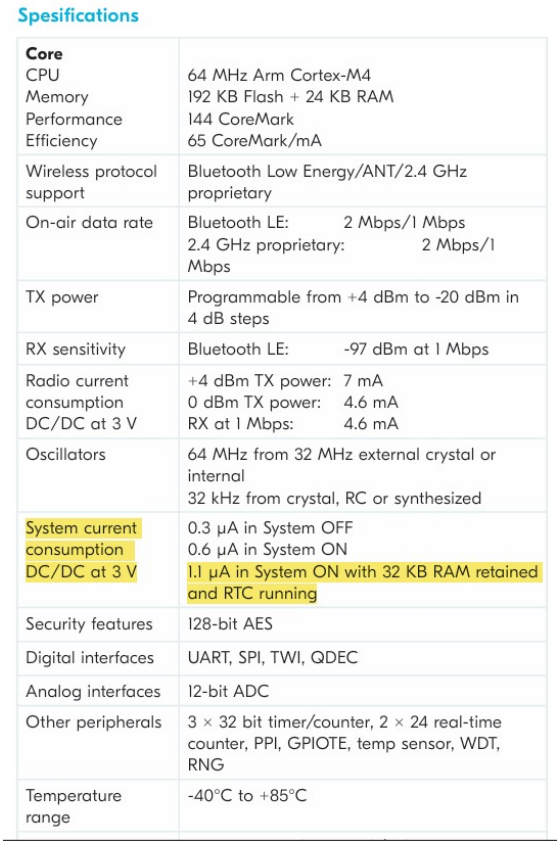
- Transmitter current up to 250 mA
- Dynamic Power Control (DPC) for optimized RF performance, even under detuned antenna conditions

Figure 24: Max current drawn from the RFID chip



29.1 Absolute Maximum Ratings*

Operating Temperature	-40°C to +85°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ and VBUS with respect to Ground[8]	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Voltage on VBUS with respect to Ground	-0.5V to +6.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0mA
DC Current V_{CC} and GND Pins	200.0mA



Spesifications

Core CPU Memory Performance Efficiency	64 MHz Arm Cortex-M4 192 KB Flash + 24 KB RAM 144 CoreMark 65 CoreMark/mA
Wireless protocol support	Bluetooth Low Energy/ANT/2.4 GHz proprietary
On-air data rate	Bluetooth LE: 2 Mbps/1 Mbps 2.4 GHz proprietary: 2 Mbps/1 Mbps
TX power	Programmable from +4 dBm to -20 dBm in 4 dB steps
RX sensitivity	Bluetooth LE: -97 dBm at 1 Mbps
Radio current consumption DC/DC at 3 V	+4 dBm TX power: 7 mA 0 dBm TX power: 4.6 mA RX at 1 Mbps: 4.6 mA
Oscillators	64 MHz from 32 MHz external crystal or internal 32 kHz from crystal, RC or synthesized
System current consumption DC/DC at 3 V	0.3 µA in System OFF 0.6 µA in System ON 1.1 µA in System ON with 32 KB RAM retained and RTC running
Security features	128-bit AES
Digital interfaces	UART, SPI, TWI, QDEC
Analog interfaces	12-bit ADC
Other peripherals	3 × 32 bit timer/counter, 2 × 24 real-time counter, PPI, GPIOTE, temp sensor, WDT, RNG
Temperature range	-40°C to +85°C

Figure 25: Current drawn from the Atmega and Bluetooth Chip

ATMEGA16U4 - ATMEL = 200 mA

PN5180 - RFID chip = 250 mA

nRF52805 - Bluetooth chip = 1.1 mA

Working hours = 8 hours

Capacity needed = 450 mA x 8 h
= 3600 mAh

Figure 26: Calculation

After that, a comparison of batteries was conducted to select a suitable battery.

Battery selecetion

Table1

Column 1	Lithium-Polymer (LiPo) Batteries	Lithium-Ion (18650)	Lithium Iron Phosphate (LiFePO4)
Availability	2(Shipping restircted)	10	5
Cycle Life	5 (300–500 cycles)	6 (500–1000 cycles)	9(2000–5000 cycles)
Safety	5 (prone to swelling/fire)	6 (moderate risk)	8(very safe, stable)1456
Form Factor Flexibility	10 (custom shapes/sizes)	5 (fixed cylindrical)	5 (fixed cylindrical/prismatic)4
Discharge Rate	9 (high, good for bursts)	7 (good)	4 (moderate)24
Temperature Tolerance	6 (sensitive)	7 (moderate)	8(excellent range)16
Weight	9 (very light)	7 (light)	7 (light)
Cost	4 (good ones are pricey)	8 (low/moderate)	8
Total	50	56	54

Figure 27: Battery Comparison

After careful selection, Li-Ion 18650 was selected as the battery.

9 Schematic Design

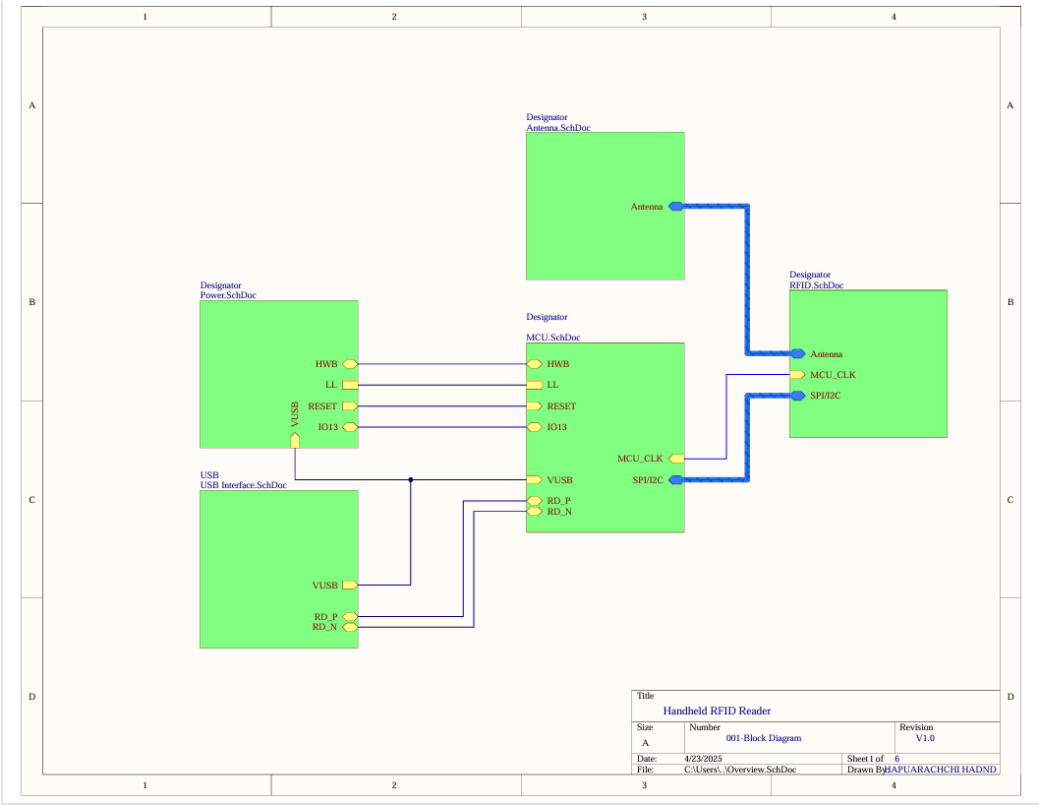


Figure 28: Overview

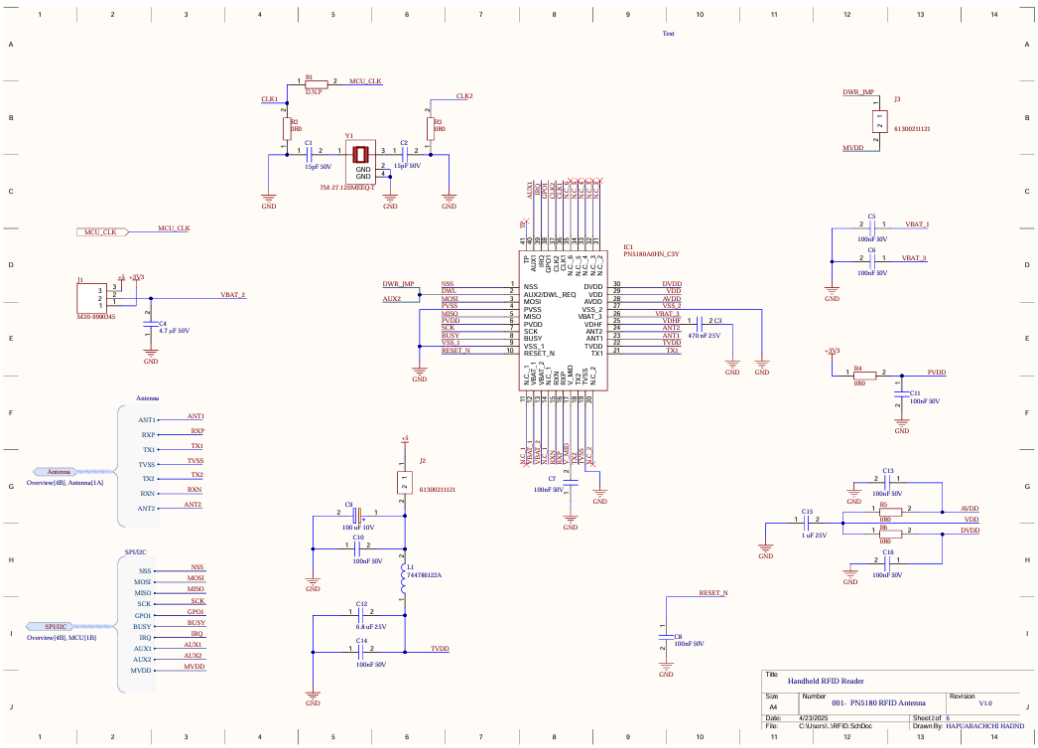


Figure 29: RFIF Reader

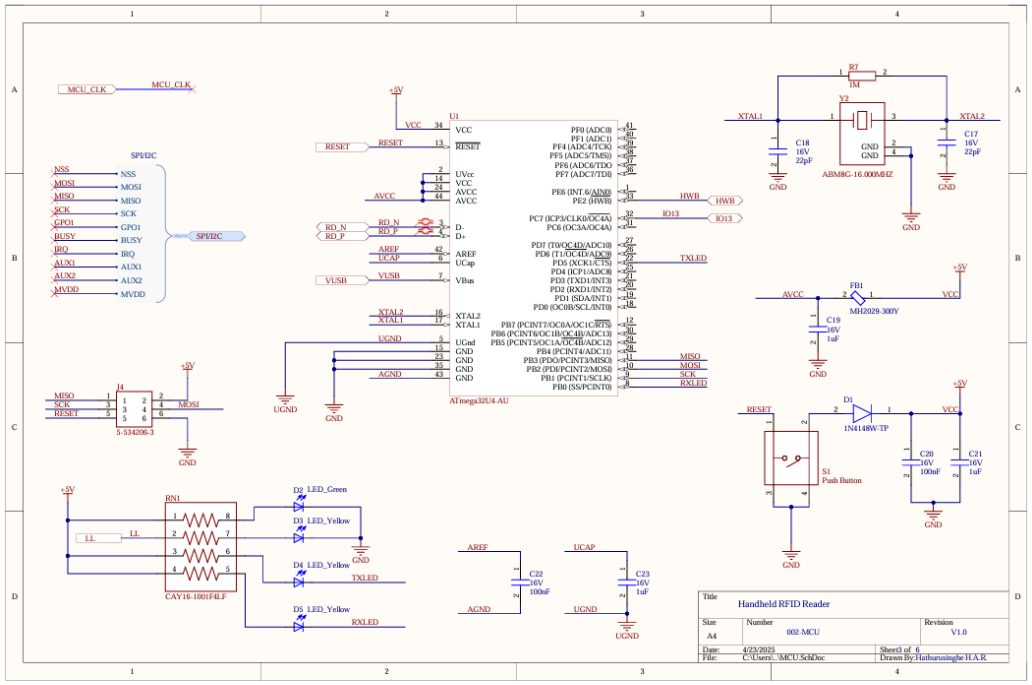


Figure 30: MCU

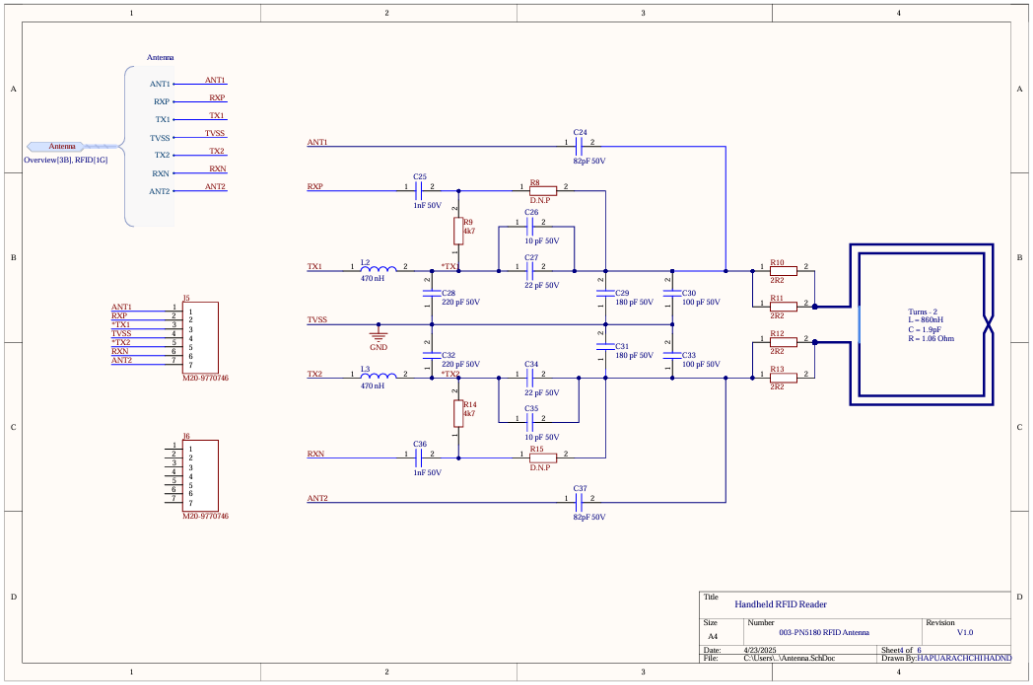


Figure 31: Antenna

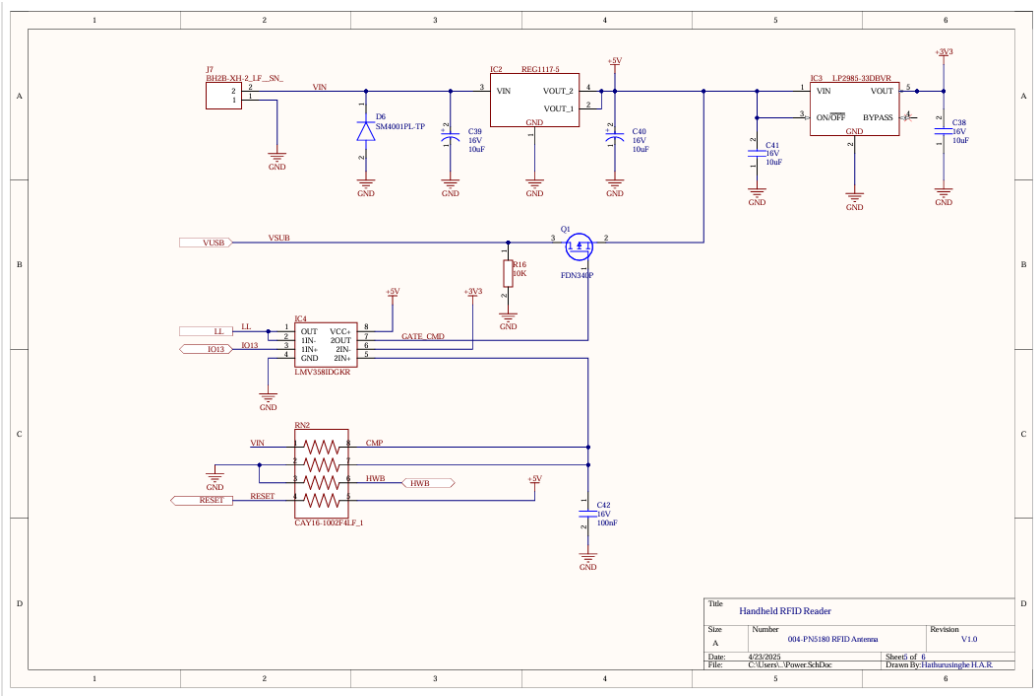


Figure 32: Antenna

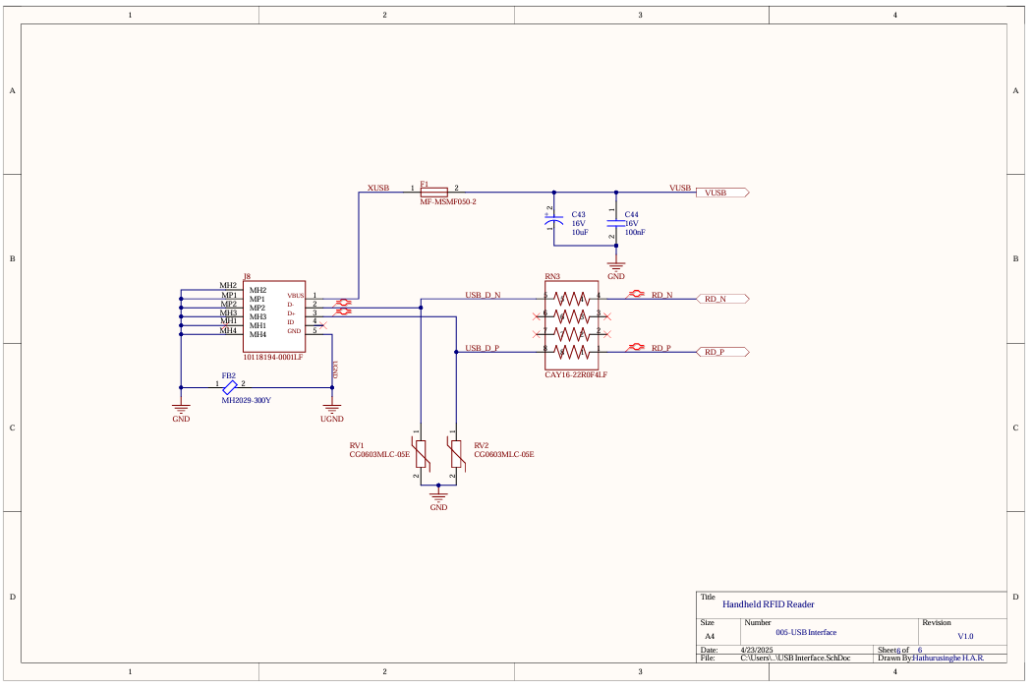


Figure 33: USB Interface

10 Antenna Design

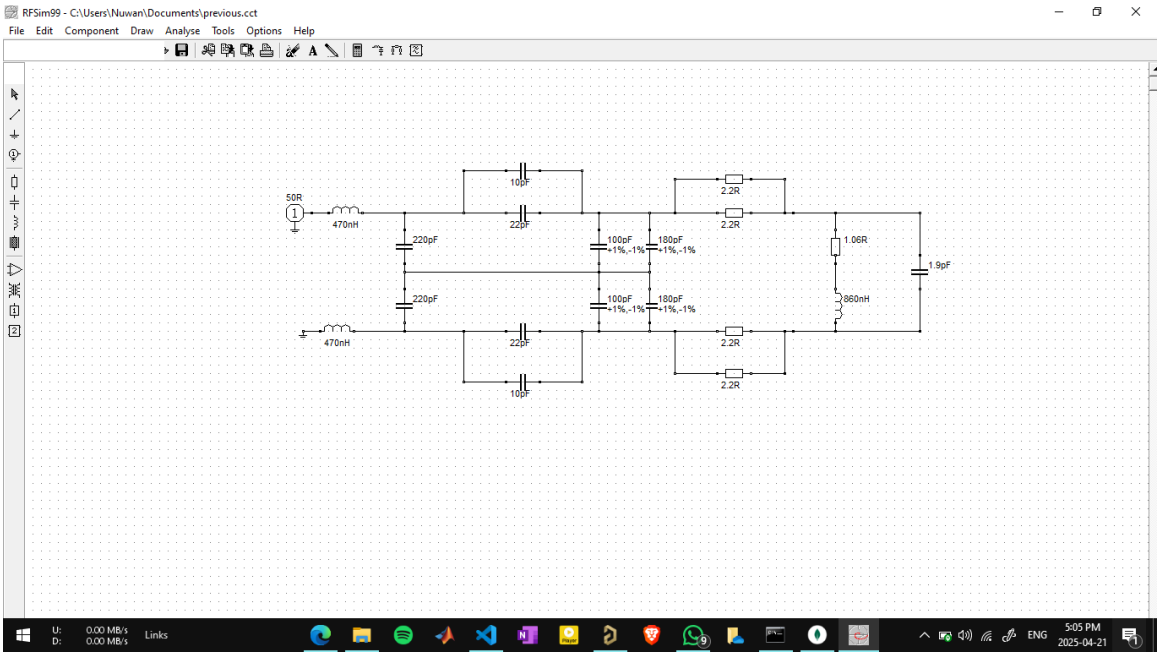


Figure 34: Antenna Circuit

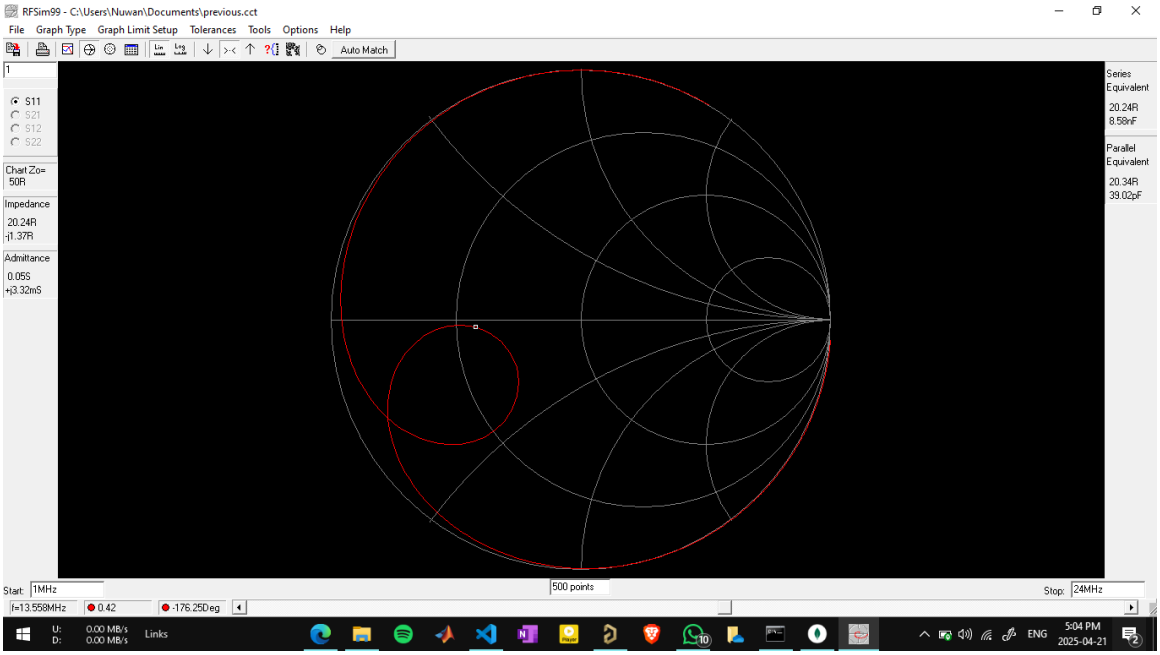


Figure 35: Simulation Results

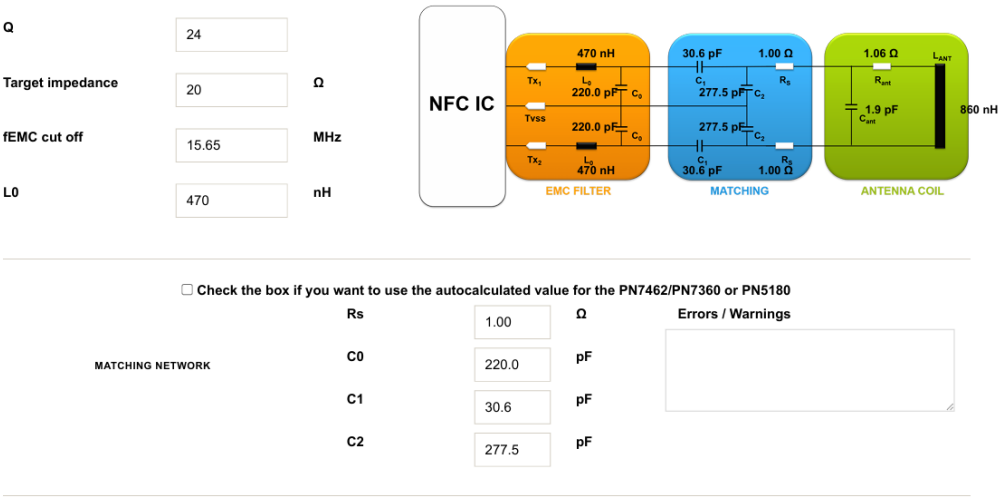


Figure 36: Calculation for the Antenna

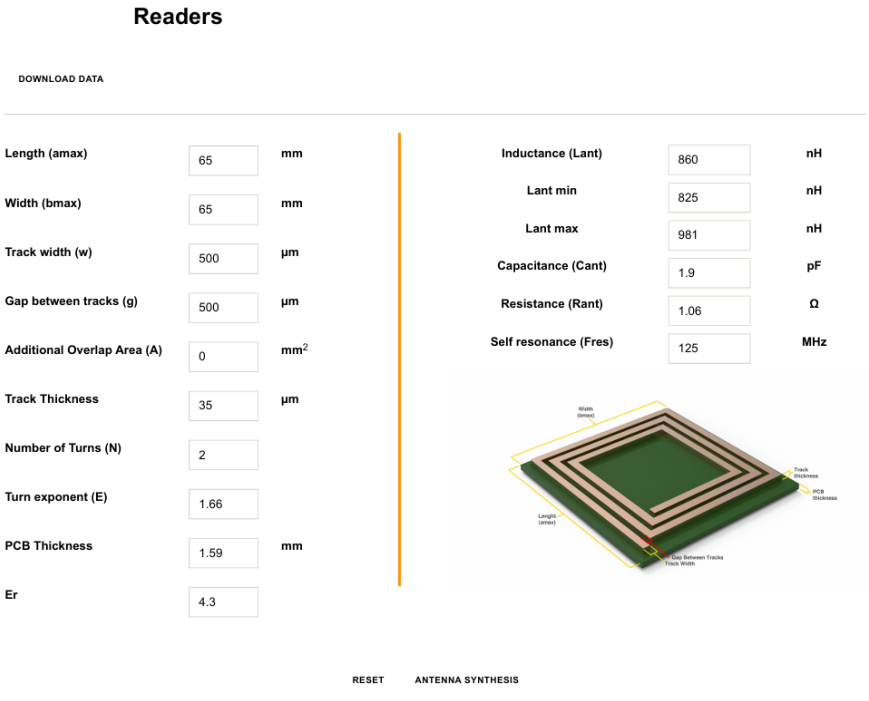


Figure 37: Calculation for the Antenna

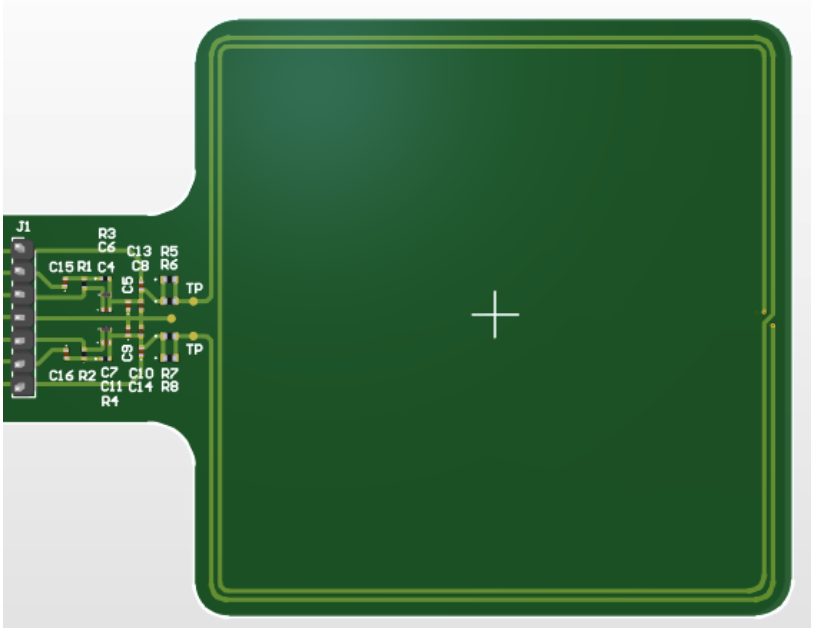


Figure 38: PCB Design of the Antenna

11 Enclosure Design

11.1 RFID Reader

Our designed enclosure for the device is given below

Figure 39: Enclosure Design - RFID Reader

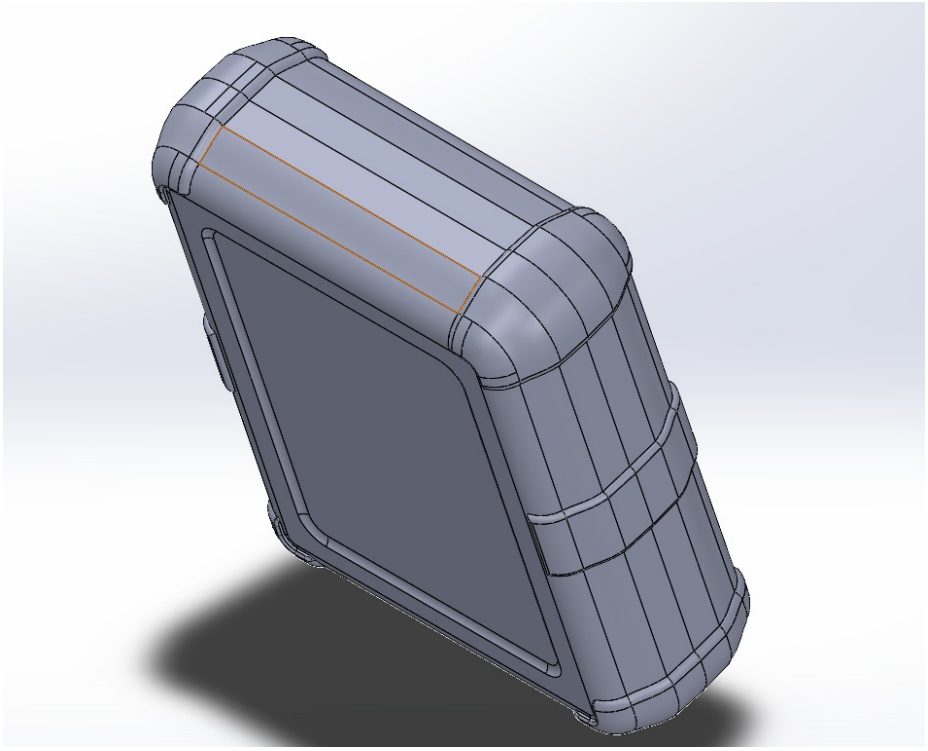


Figure 40: Enclosure Design - Antenna Holder

11.2 Charging Cradle

Our designed enclosure for the charging cradle is given below.

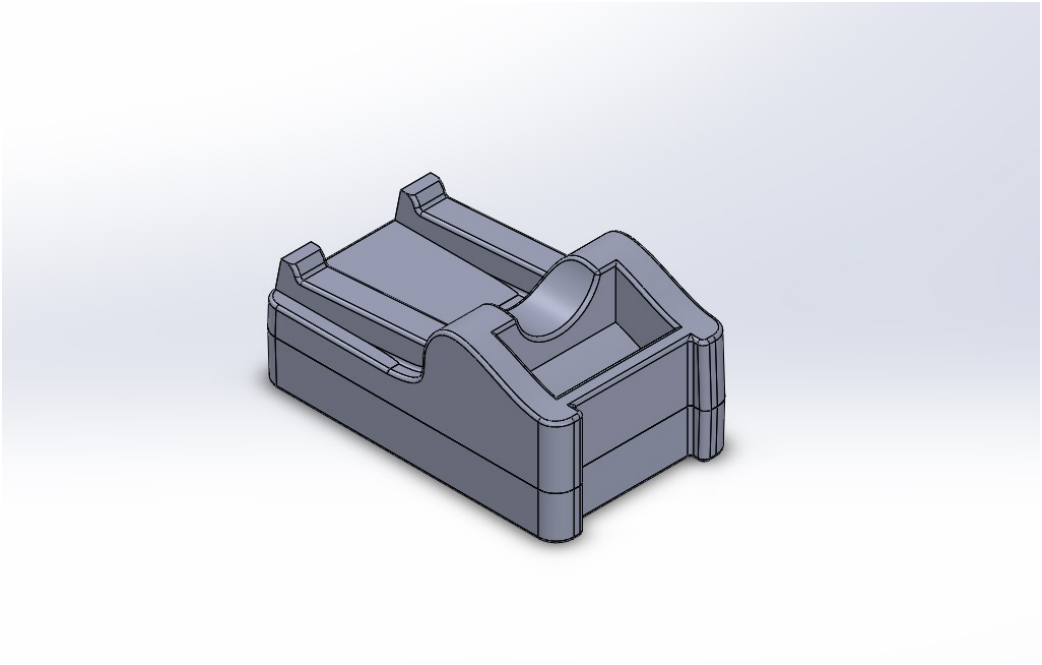


Figure 41: Enclosure Design - Charging Cradle

12 Web app development

A database and a web application are also being developed in the initial stages to support the RFID reader by managing and visualizing the collected data.

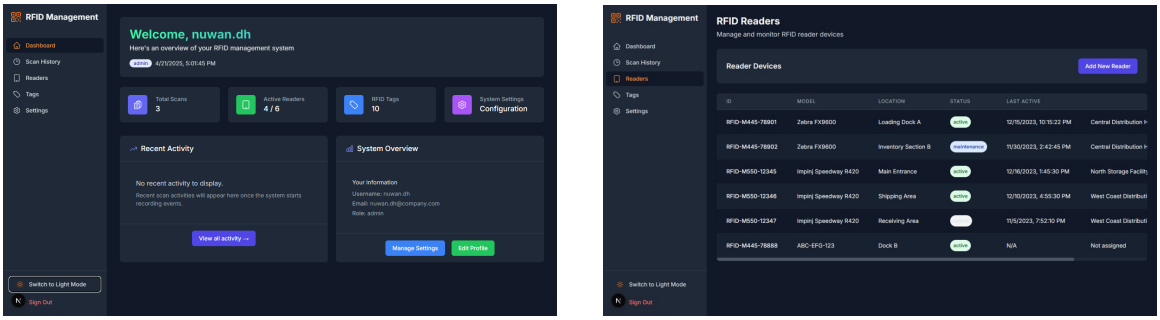


Figure 42: Web application interface – Home page and RFID tags page

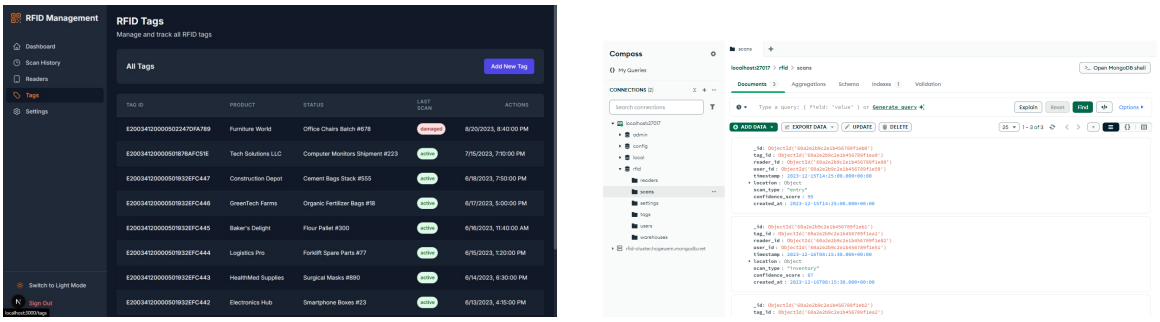


Figure 43: Web application interface – Reader devices page and Database

Although the project initially explored UHF technology, the complexity and antenna design constraints led to a well-justified pivot toward HF (13.56 MHz) RFID technology. This shift allowed for a more manageable and reliable implementation path, aligning with our timeline and available resources. The adoption of the PN532 module, integration of a microcontroller, and ergonomic considerations for handheld use were key milestones in refining the system.

Moving forward, our focus will shift to PCB layout, firmware development, and mechanical prototyping, where testing and iterative refinement will be crucial. Ultimately, this project strengthened our understanding of embedded system design and RFID technology and underscored the value of structured collaboration and industrial feedback in creating viable engineering solutions.

Stage 2

13 Change from Bluetooth to WiFi

Feature	Wi-Fi (ESP8266)	Bluetooth (HC-05 / BLE)
Range	30–100 meters	10–15 meters (Classic), up to 50 m (BLE)
Internet/Cloud Access	Direct connection to internet/cloud servers	Requires smartphone to forward data via Wi-Fi
Data Throughput	High (up to 72 Mbps)	Low (1–2 Mbps for Classic, 1 Mbps for BLE)
Smartphone Requirement	Only Wi-Fi needed (e.g., mobile hotspot)	Requires both Bluetooth and Wi-Fi ON
Protocol Support	Native support for MQTT, HTTP, TCP/IP	Requires custom handling for server communication
Multi-Device Communication	Supports multiple clients via standard IP stack	Mostly point-to-point
Power Consumption (Module)	70–200 mA (TX/RX)	30–40 mA (Classic BT), 10–20 mA (BLE)
Phone Power Usage	Only Wi-Fi active — more efficient	Wi-Fi + Bluetooth ON — higher battery drain
Integration Complexity	Easy with standard IoT libraries	Requires Bluetooth pairing and protocol bridging

Table 3: Comparison of Wi-Fi (ESP8266) vs Bluetooth for RFID Reader Communication

After comparing the two modes, we have decided to use WiFi rather than Bluetooth.

13.1 WiFi Chip

We have selected ESP8266 as our WiFi module.

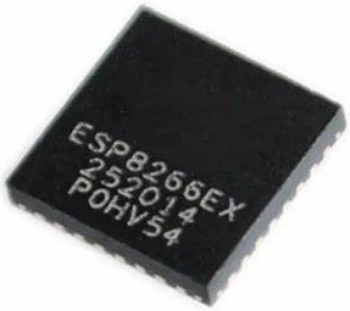


Figure 44: ESP8266 Chip

14 Finalized Block Diagram

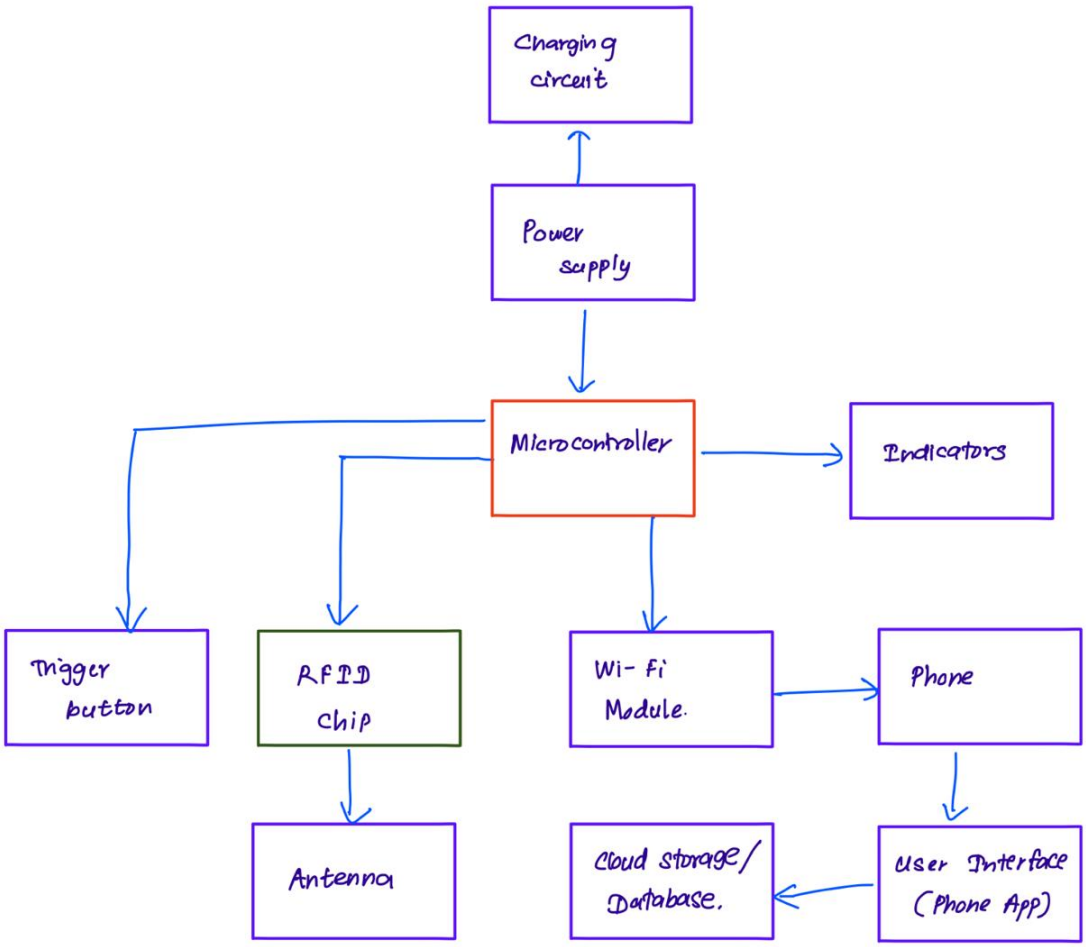


Figure 45: Finalized Block Diagram

15 Schematic Design

Minor changes were made to the previously designed schematics. Finalized ones are included below.

15.1 Antenna

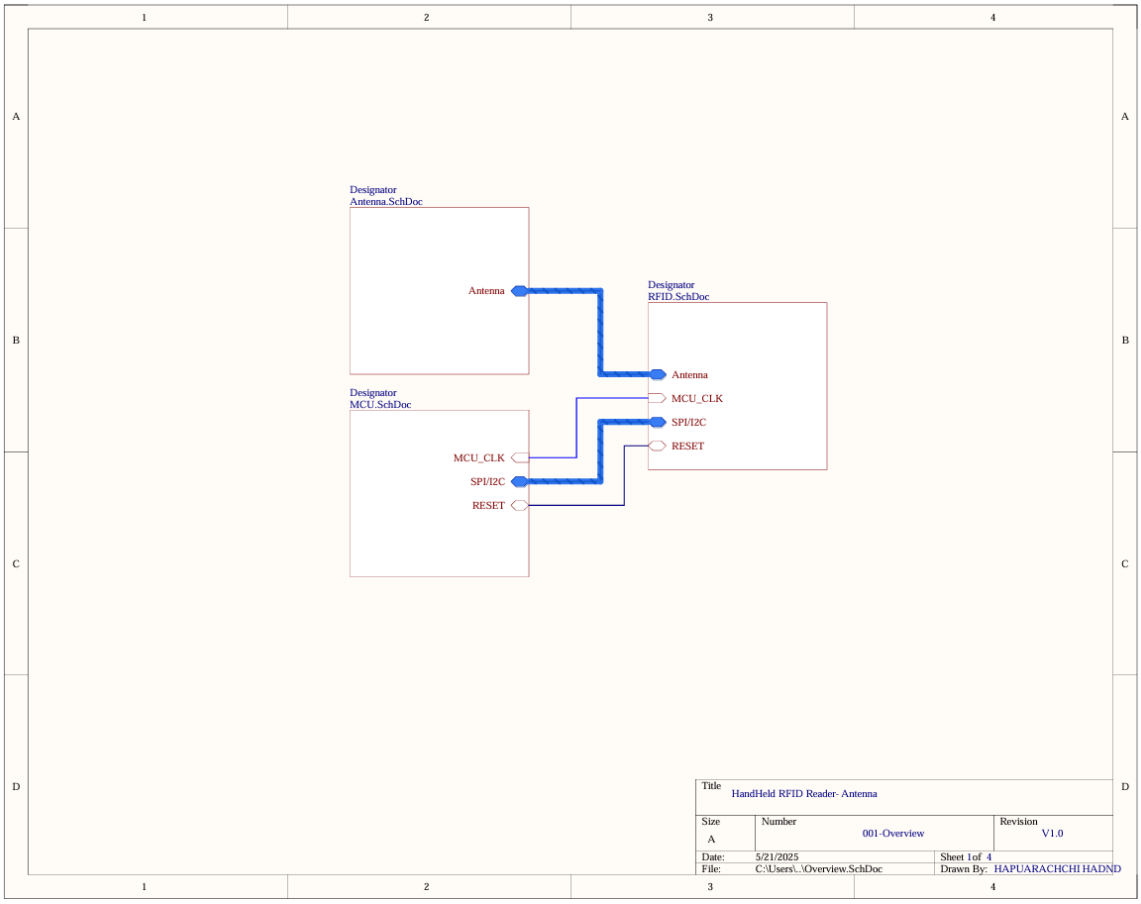


Figure 46: Overview

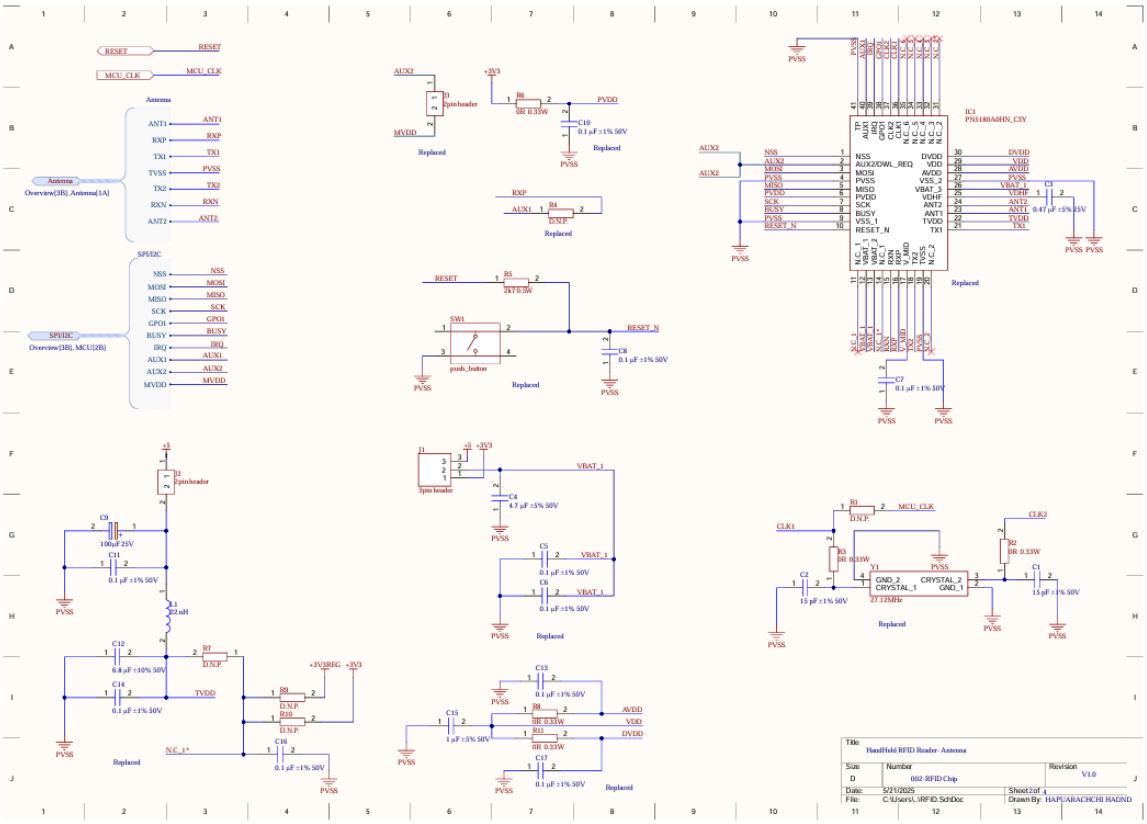


Figure 47: PN5180

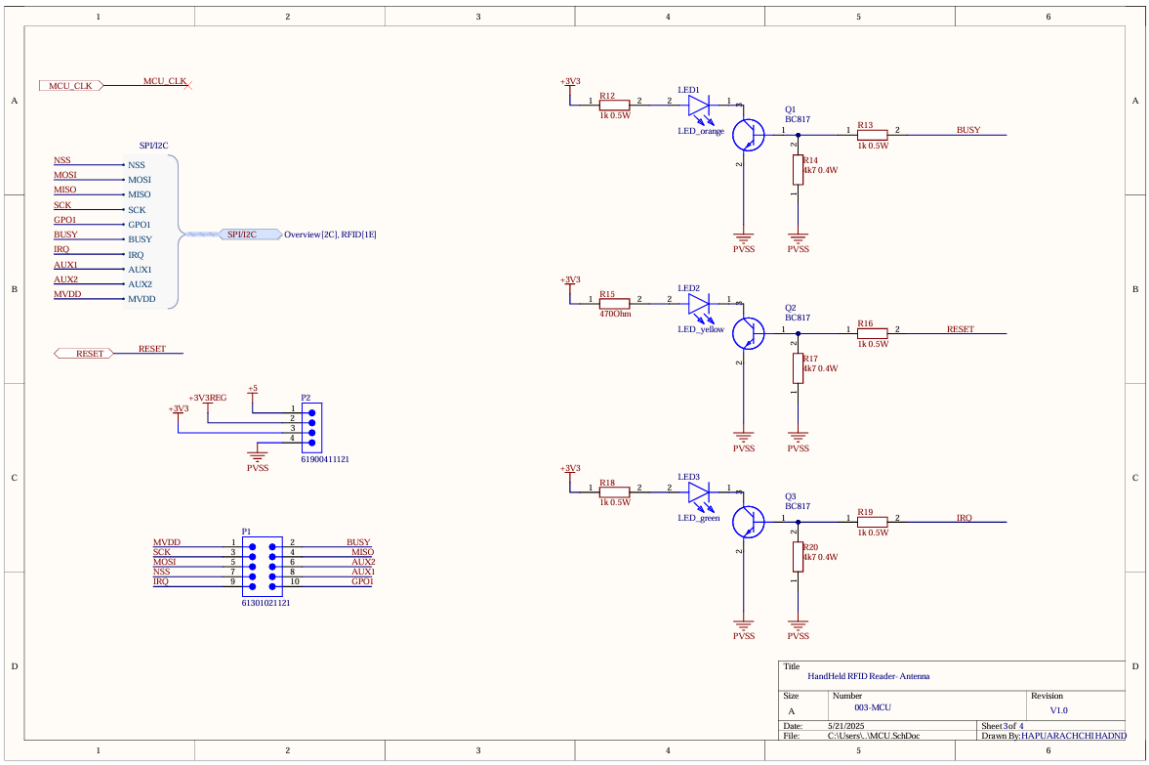


Figure 48: Communication with the MCU

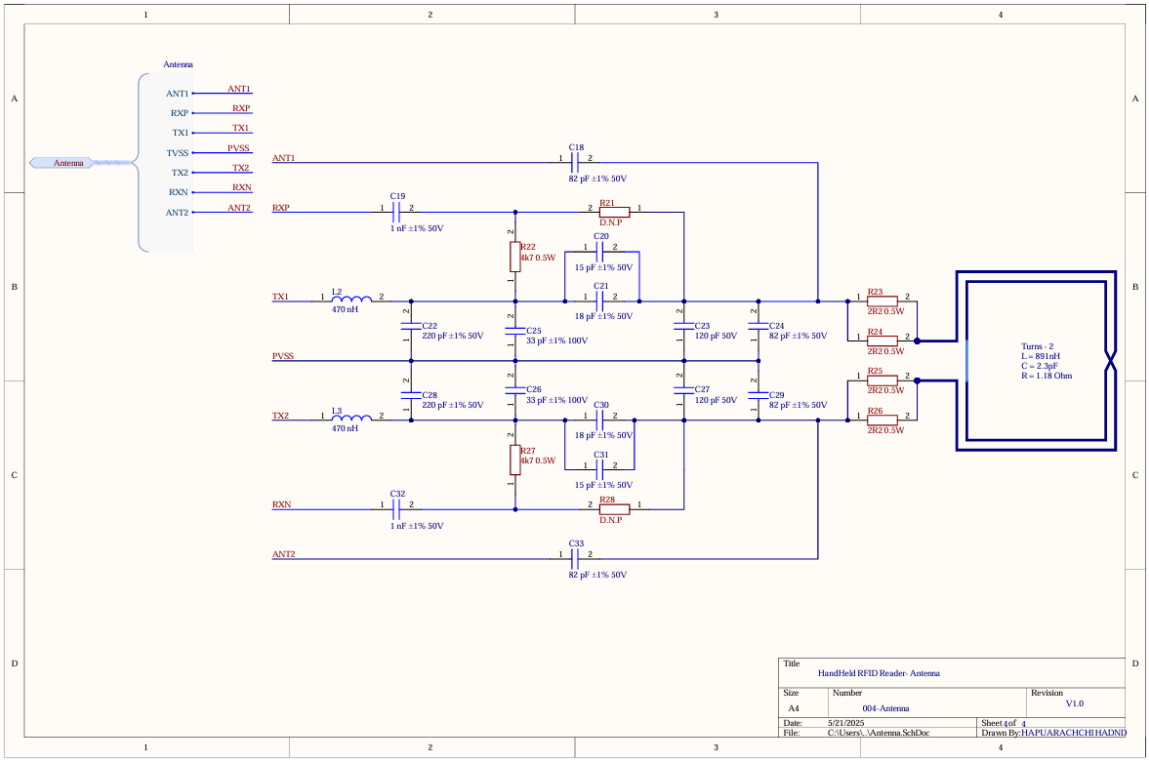


Figure 49: Antenna

15.2 MCU

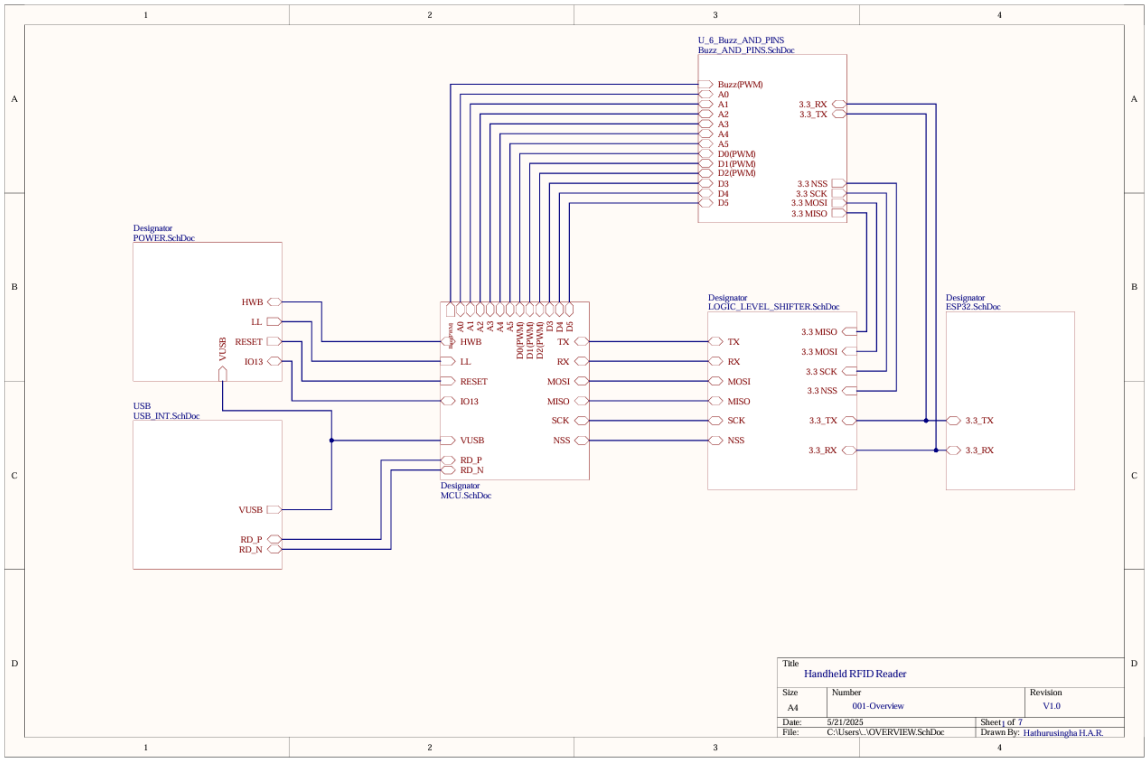


Figure 50: Overview

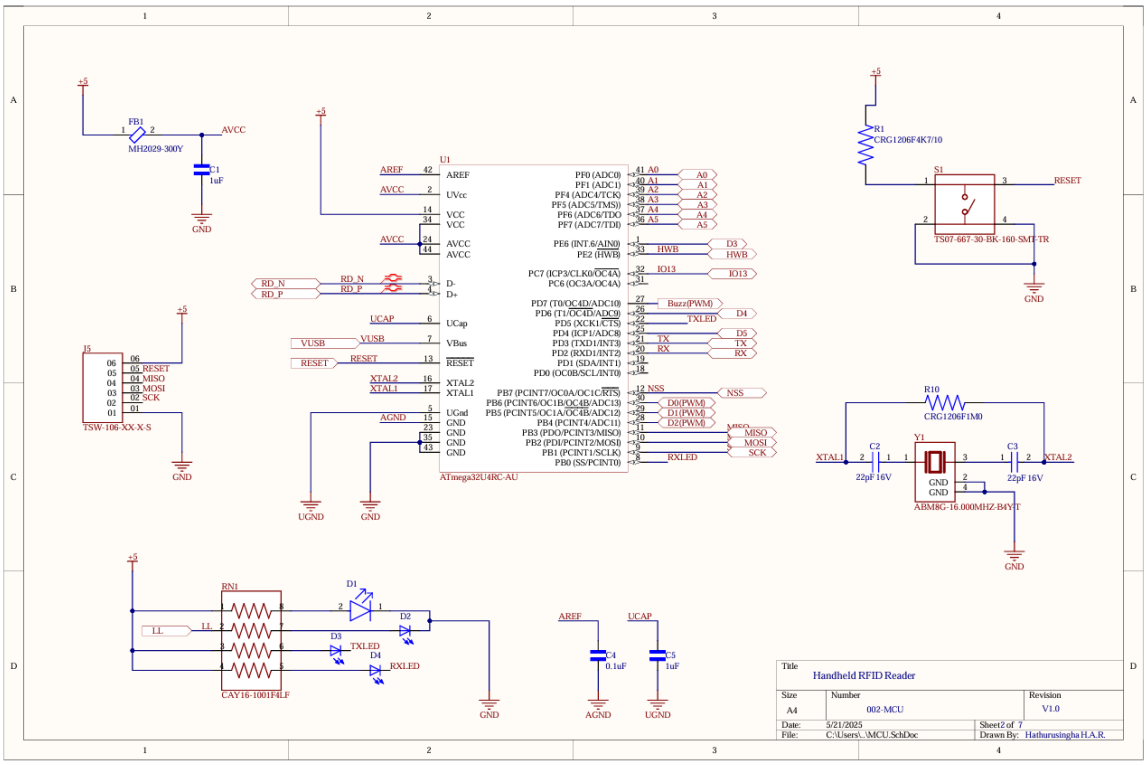


Figure 51: MCU

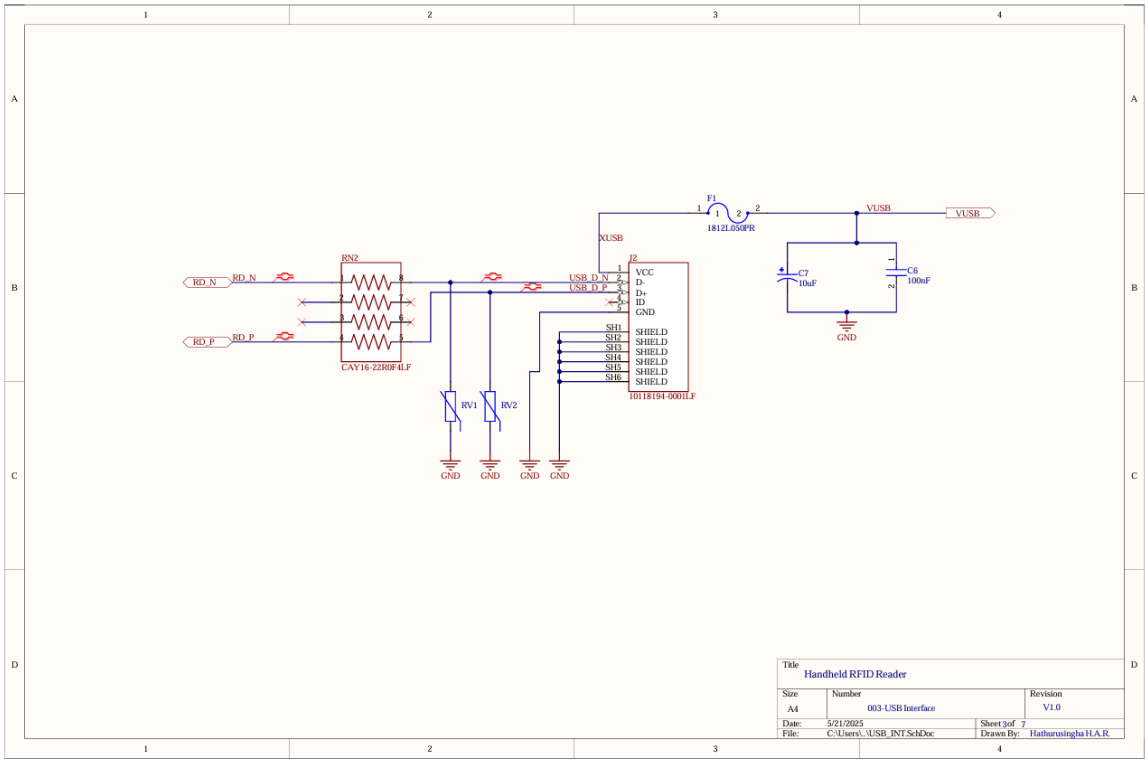


Figure 52: USB Interface

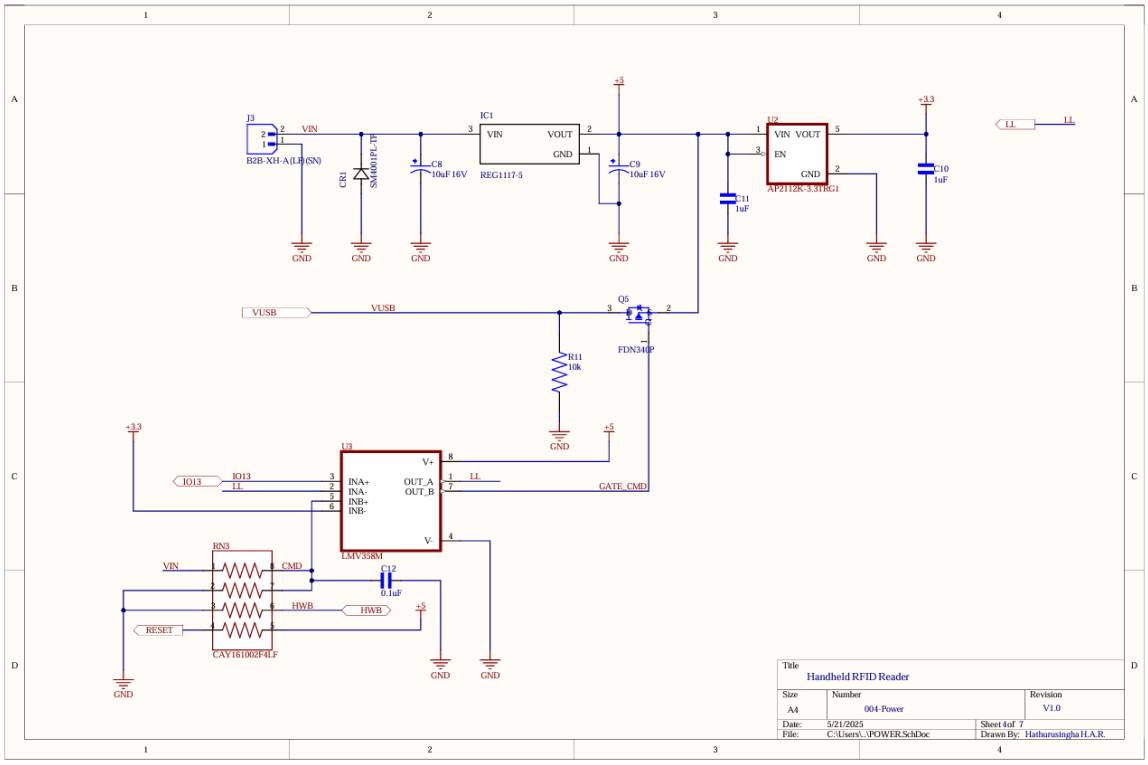


Figure 53: Power

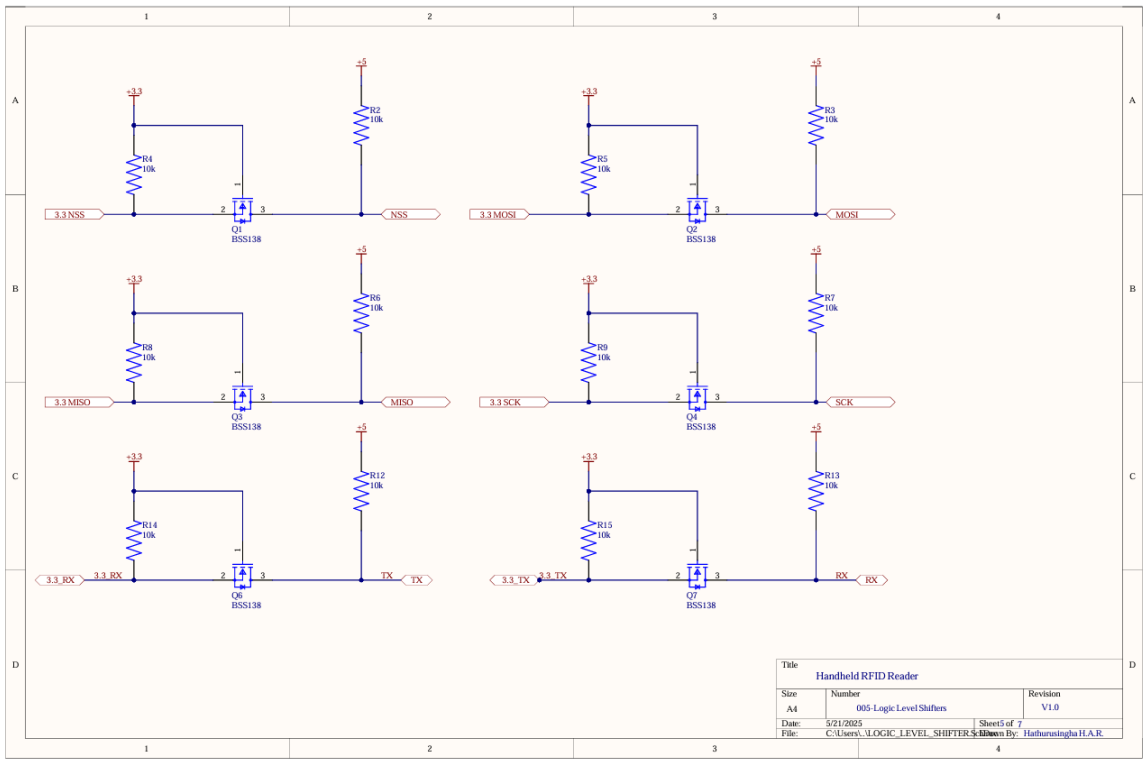


Figure 54: Logic level Shifters

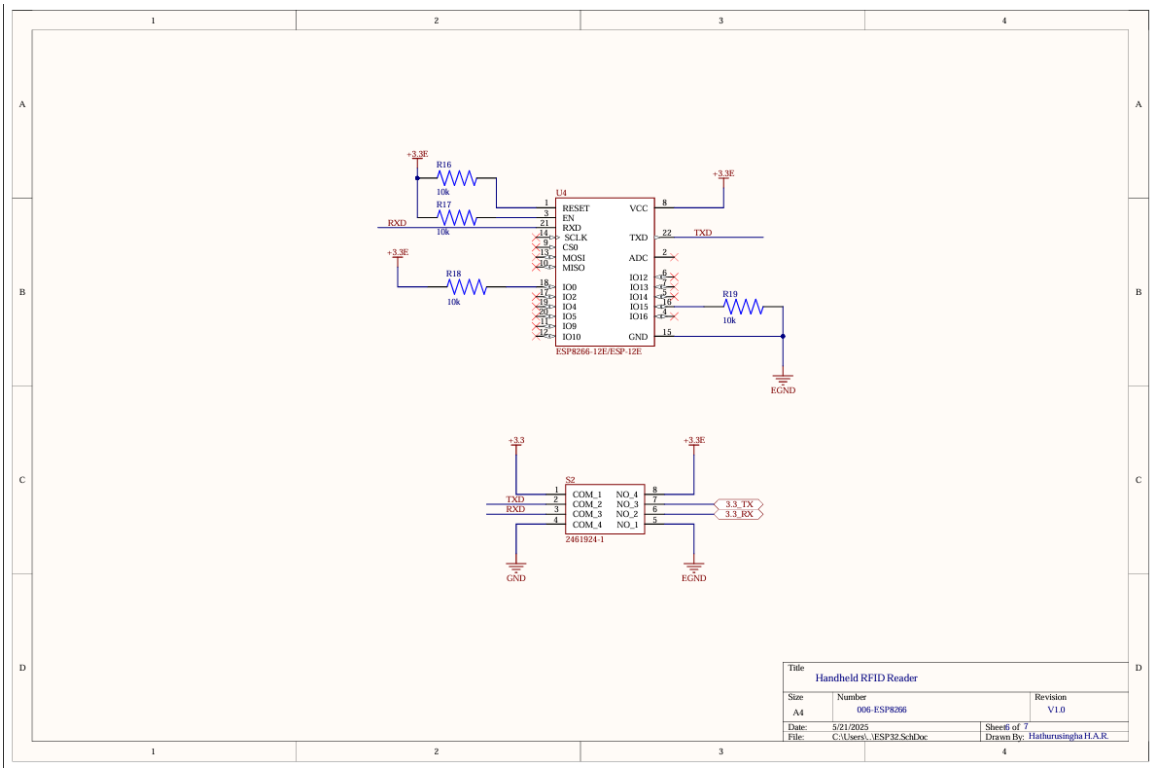


Figure 55: ESP 8266

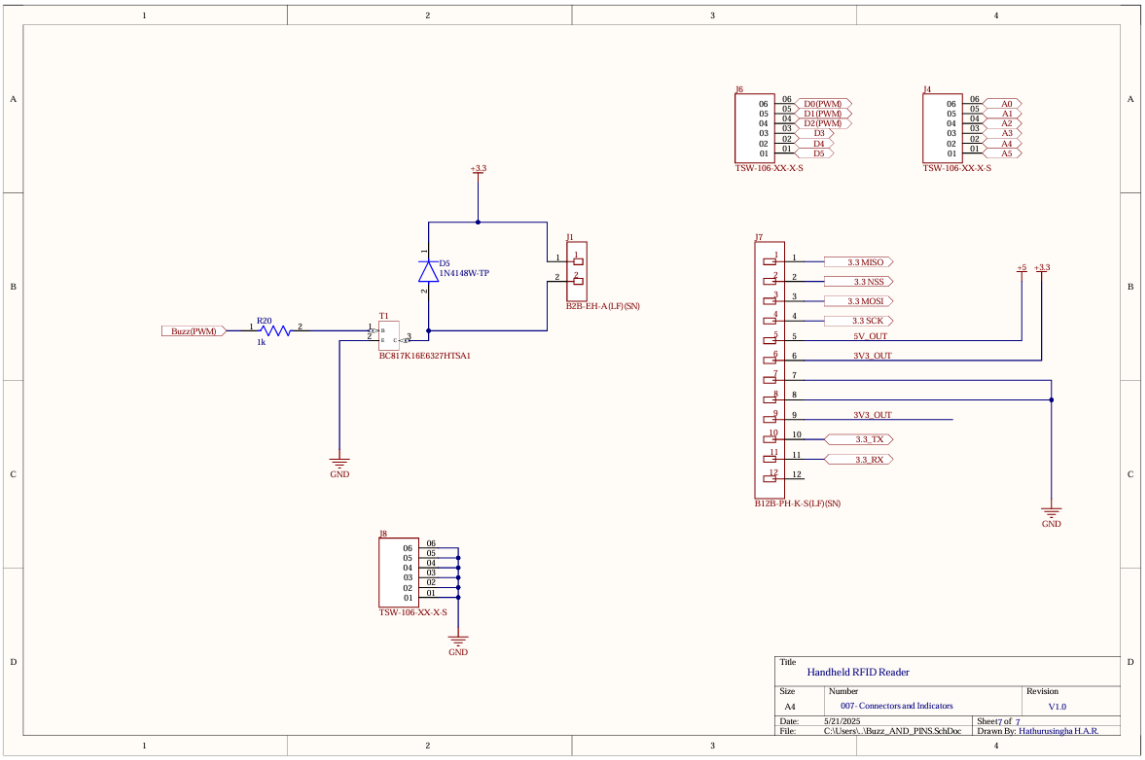


Figure 56: Connectors and Indicators

15.3 Power

The power circuit of the RFID reader is designed to efficiently manage both external power input and battery operation. A 12V supply is delivered through pogo pins when the device is docked in the charging cradle. Automatic power source switching is handled using a P-type MOSFET, which prevents current from flowing to the circuit during charging, ensuring that external power is used solely for charging purposes. When not connected to the cradle, the device operates on battery power supplied by two 18650 lithium-ion cells connected in series. To ensure safe and balanced charging of both cells, a Battery Management System (BMS) board is integrated into the design, improving reliability, battery life, and system safety.

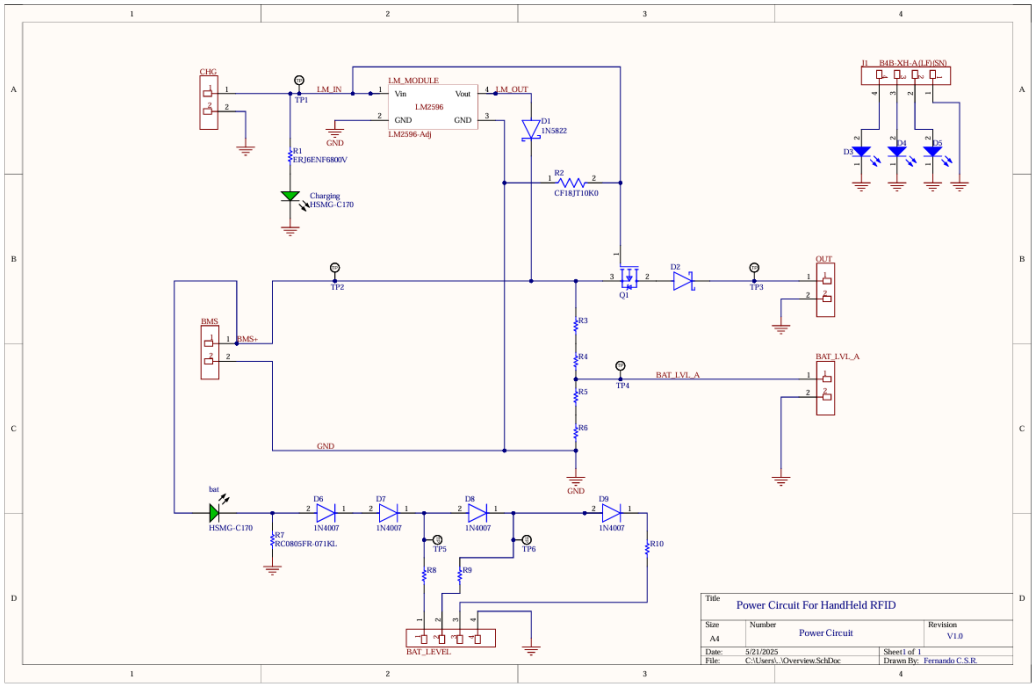


Figure 57: Power Circuit

16 PCB Design

Initially, the handheld RFID reader was designed with a 4-layer PCB to ensure optimal signal integrity and compact integration of both the microcontroller unit (MCU) and the RFID antenna. However, it became evident that a 4-layer design introduced unnecessary complexity in fabrication and significantly increased production costs, especially for low-volume manufacturing. Additionally, since our design uses HF (13.56 MHz) RFID rather than UHF, the high-frequency signal handling requirements are more forgiving, reducing the need for advanced layer stacking and controlled impedance. As a result, we adopted a more practical solution by separating the system into two 2-layer PCBs—one for the MCU and control circuitry, and another for the RFID antenna. This modular approach preserved design flexibility, simplified testing, and reduced manufacturing costs without compromising performance.

The final design of the handheld RFID reader consists of three separate 2-layer PCBs, each dedicated to a specific function: the antenna, the microcontroller unit (MCU), and the power management circuitry. This modular architecture was chosen to simplify the overall layout, reduce fabrication complexity, and improve ease of debugging. By separating the antenna from digital and power components, we reduced potential interference and improved signal performance. Separating the power module allowed for more efficient thermal and power distribution design, enhancing system reliability.

16.1 Antenna

PCB layout

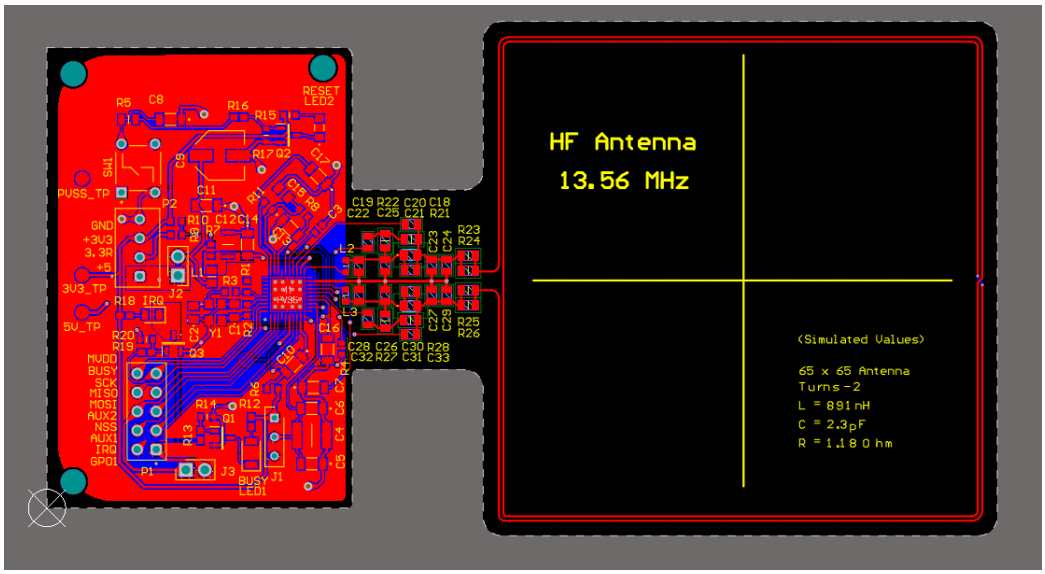


Figure 58: Top Layer

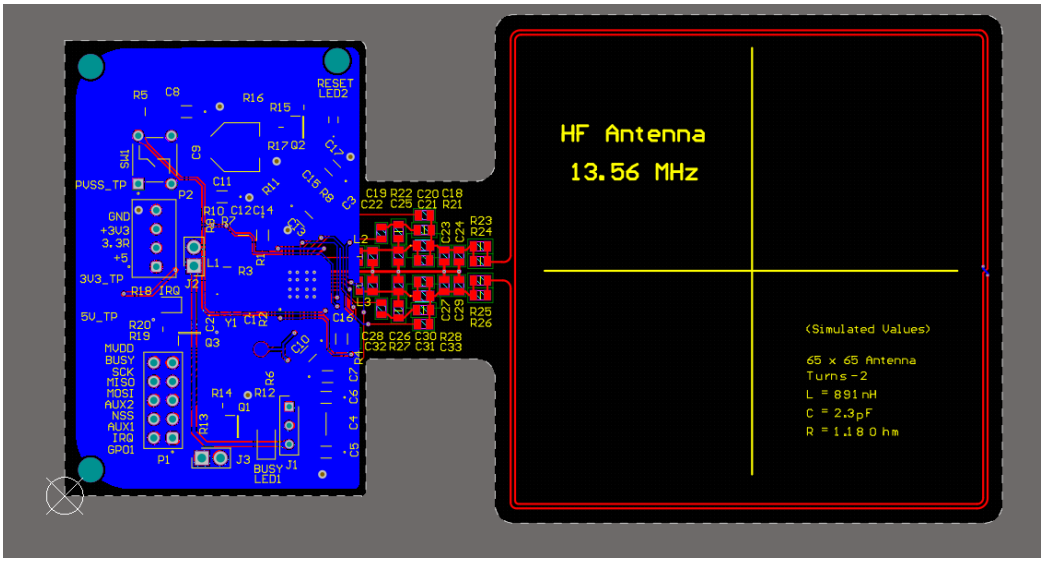


Figure 59: Bottom Layer

3D View

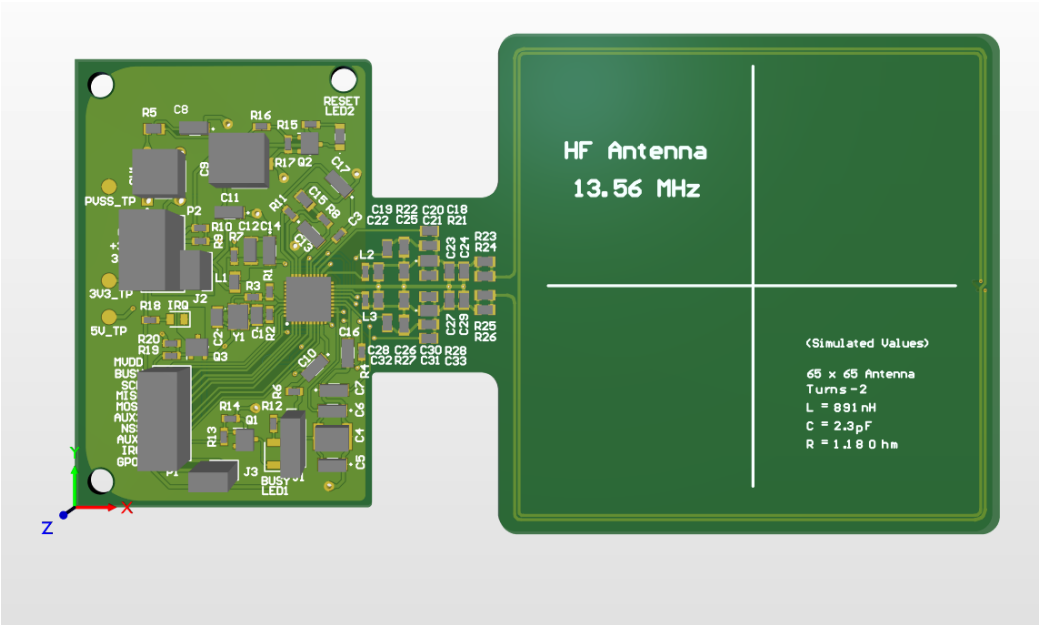


Figure 60: 3D View

16.2 MCU

PCB layout

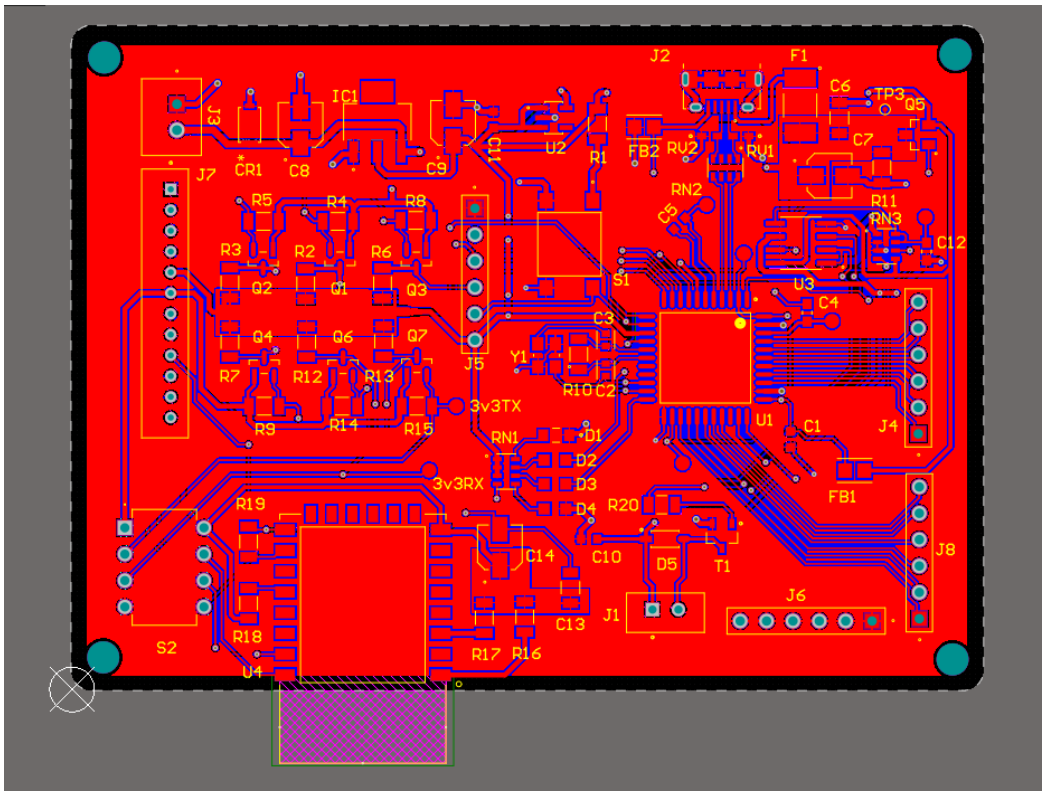


Figure 61: Top layer

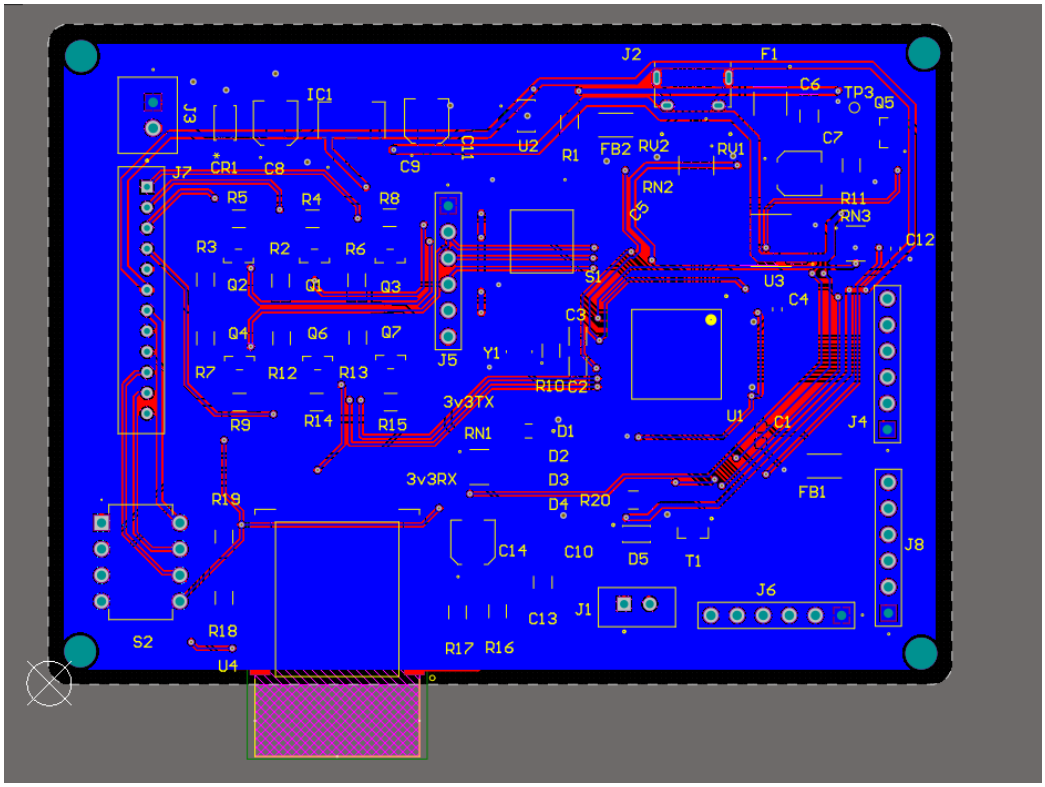


Figure 62: Bottom layer

3D View

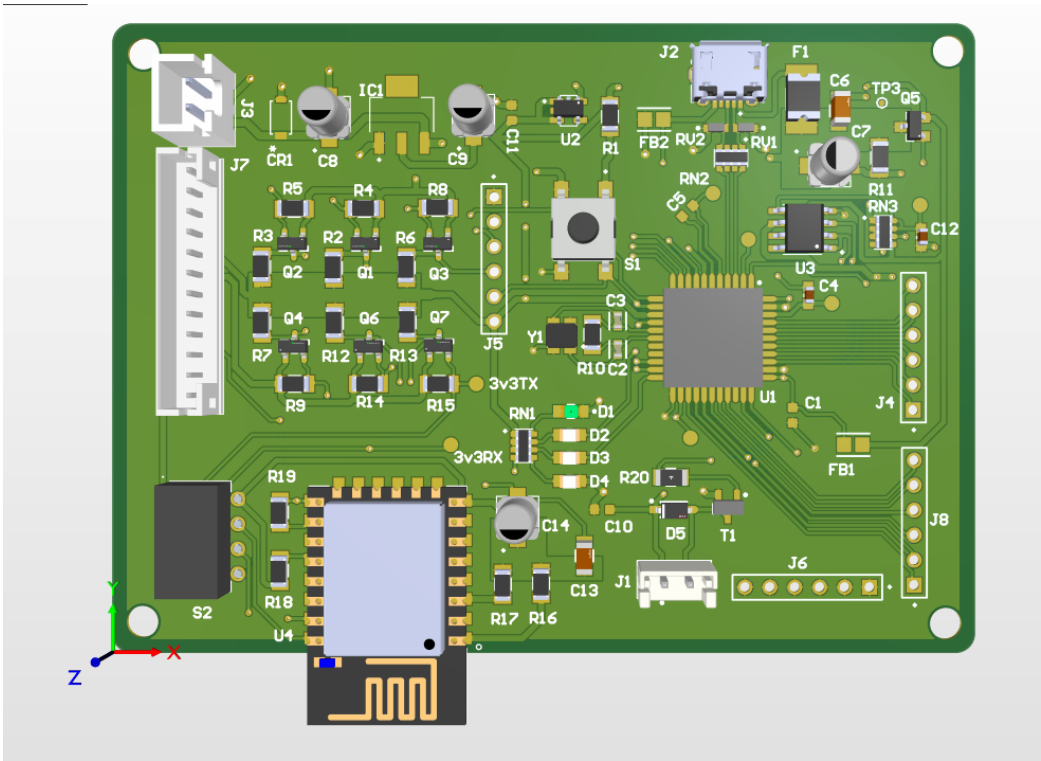


Figure 63: 3D View

16.3 Power

PCB Layout

The PCB layout was designed to allow the battery level indicator module to be physically separated from the main PCB. This enables flexible placement within the enclosure, allowing the indicators to be positioned for optimal visibility.

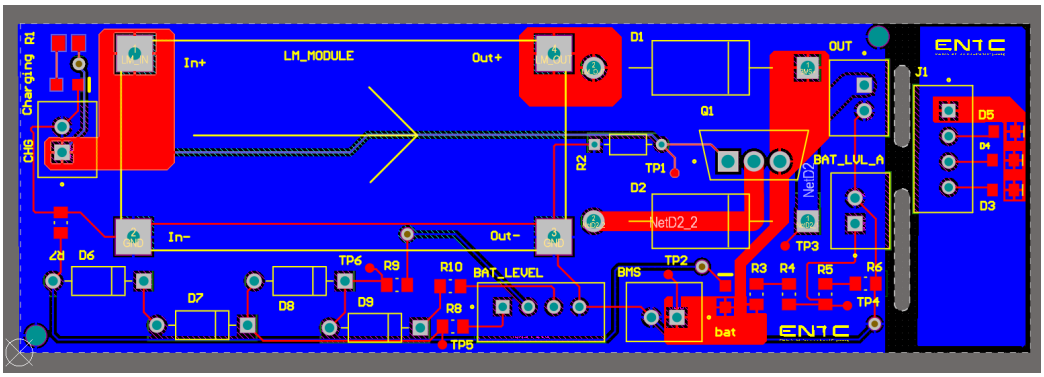


Figure 64: Top layer

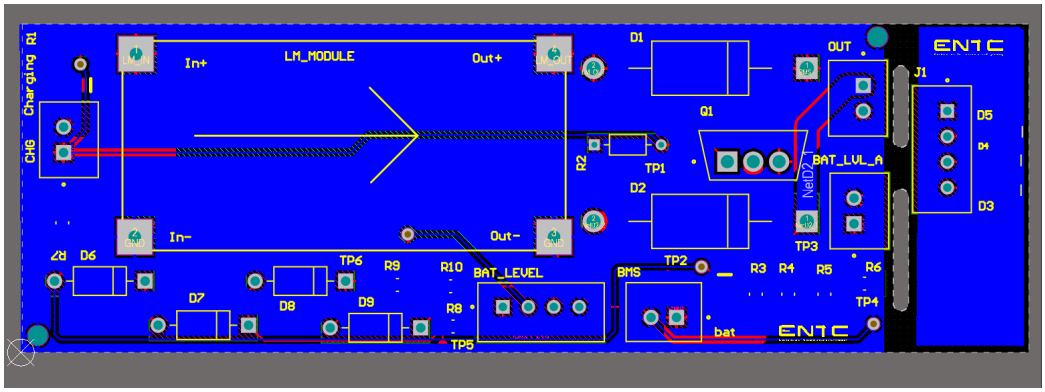


Figure 65: Bottom layer

3D View

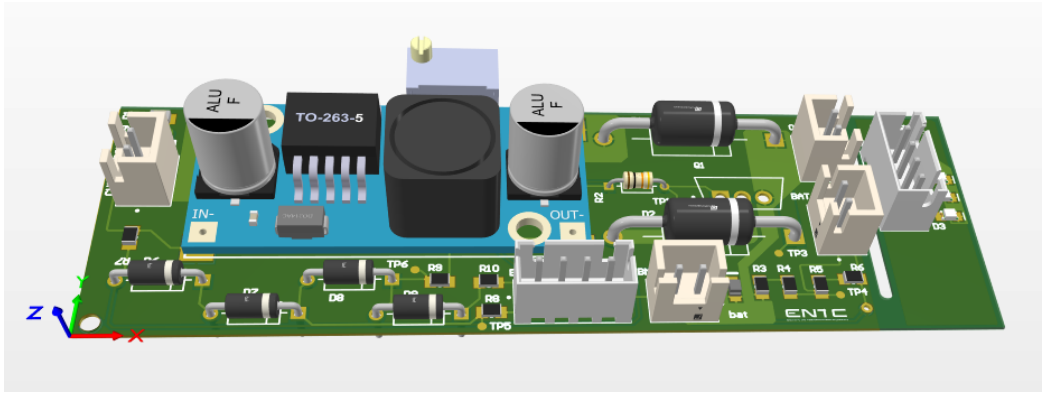


Figure 66: MCU Overview

17 Enclosure Design

An additional round of enclosure design was undertaken to address integration, usability, and alignment requirements. Both the RFID reader and the power cradle were redesigned to accommodate the updated PCB layout, modular components, and improved mechanical fit. The finalized enclosures are shown in the images below.

17.1 SolidWorks Design

17.1.1 RFID reader

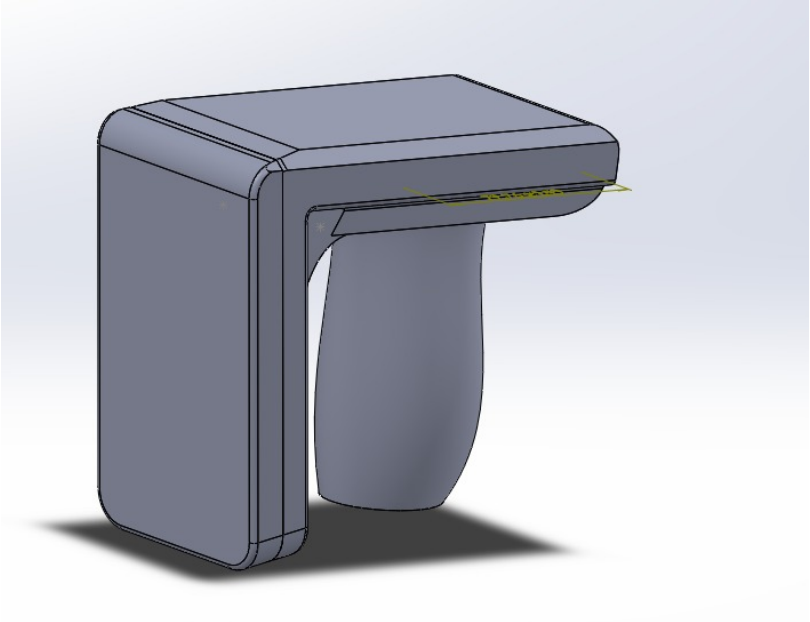


Figure 67: Enclosure - Left View

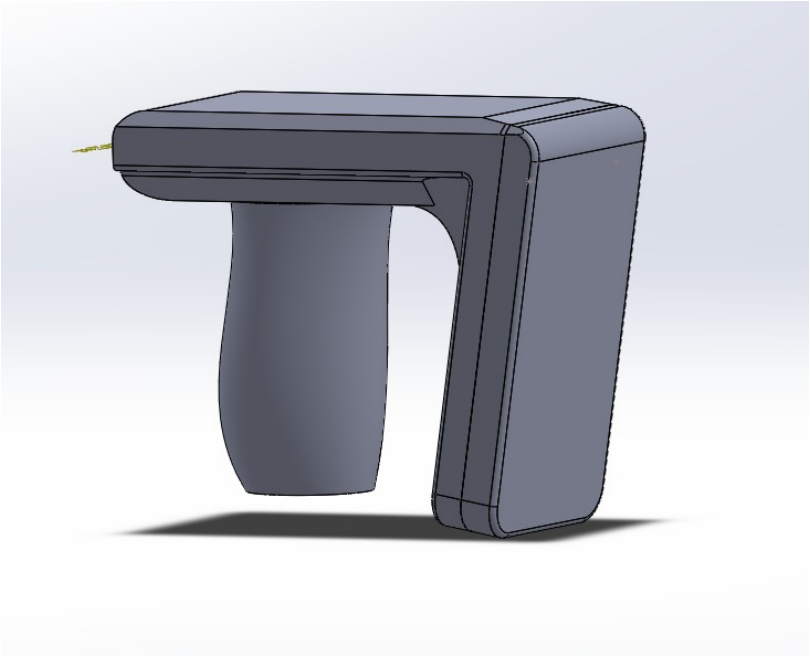


Figure 68: Enclosure - Right View

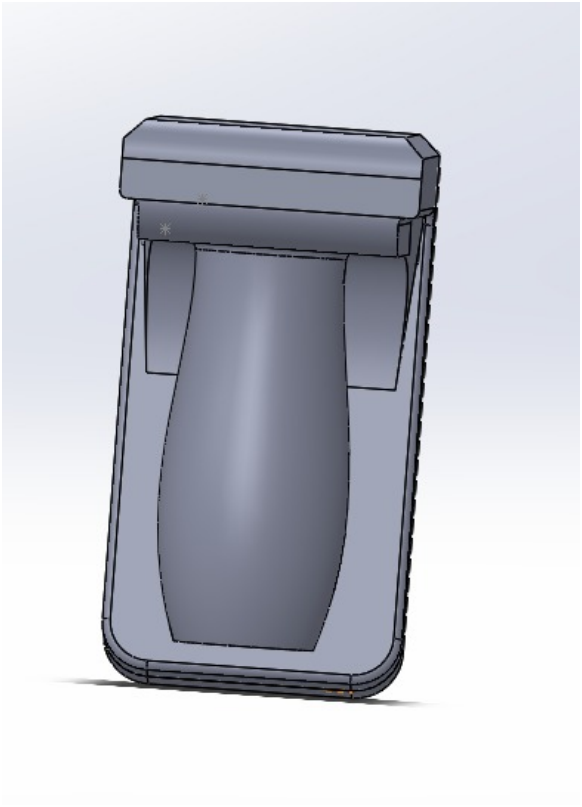


Figure 69: Enclosure - Back View

17.1.2 Power cradle

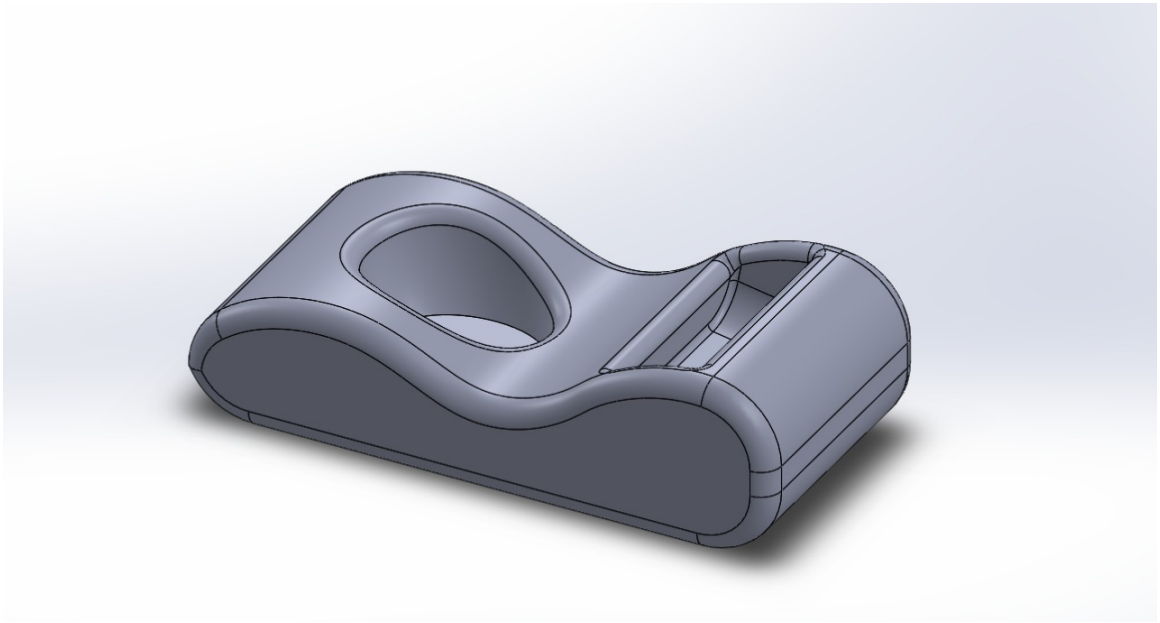


Figure 70: Power Cradle

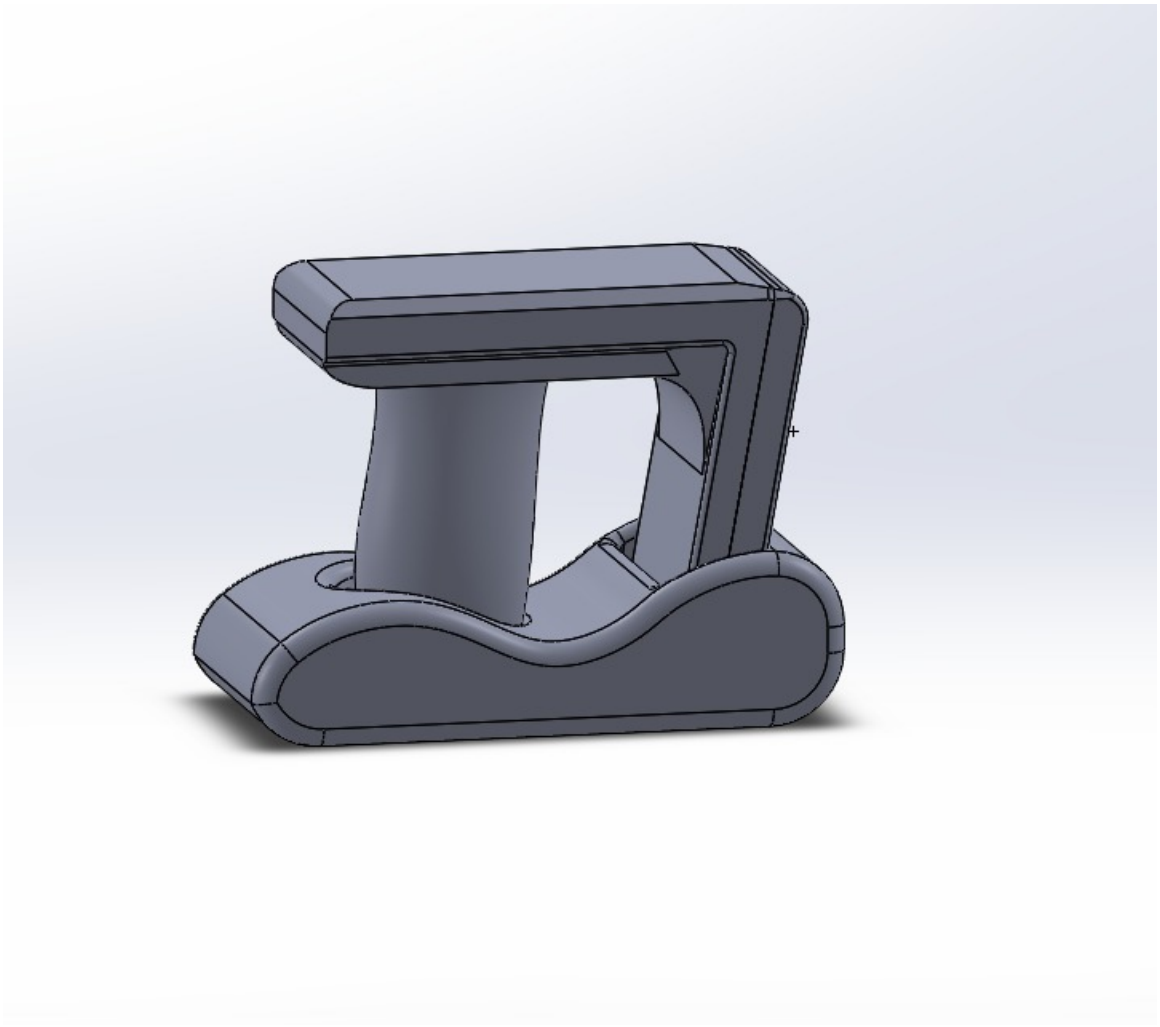


Figure 71: Enclosure - Final Assembly

17.2 Prototype Stage

A prototype of the handle was 3D printed using PLA material to evaluate the ergonomics, fit, and overall design before final fabrication.

Figure 72: Handle- Prototype

17.3 Printed Enclosure

The final enclosure was 3D printed using ABS material to meet industry-level standards for strength, durability, and heat resistance.

Figure 73: Handheld reader

Figure 74: Cradle

18 Firmware Implementation

Module	Functionality	Programming Approach
PN5180 (RFID Reader)	High-frequency RFID front-end that reads ISO 15693 tags. Communicates with the microcontroller via SPI.	Controlled by the ATmega32U4 via SPI interface. Uses NXP's PN5180 driver.
ATmega32U4 (Microcontroller)	Central control unit. Interfaces with the PN5180 via SPI and with the ESP8266 via UART. Handles RFID tag reading and forwards data.	Programmed in Atmel Studio using register-level C. Programmed via USB interface.
ESP8266 (WiFi Module)	Transmits RFID data to a remote server or mobile device via WiFi. Acts as a communication bridge.	Programmed using the ESP8266 RTOS SDK with Xtensa GCC. Flashed using `esptool.py` over UART. Receives data via UART from ATmega32U4 and sends it over WiFi.

Table 4: Functionality of the Modules and their Programming Approach

18.1 Initial Testing

We conducted tag readability testing using the PN5180 RFID module and successfully read the data stored on the provided HF RFID tag. To simulate the trigger-based reading functionality of our final product, we used a push button to initiate the tag scans, effectively replicating the intended user interaction. All testing was conducted using an Arduino Leonardo board, with SPI communication used to interface with the PN5180 module. Additionally, a NodeMCU was successfully connected to the Leonardo board via a UART interface, confirming the viability of wireless communication in the final design architecture.

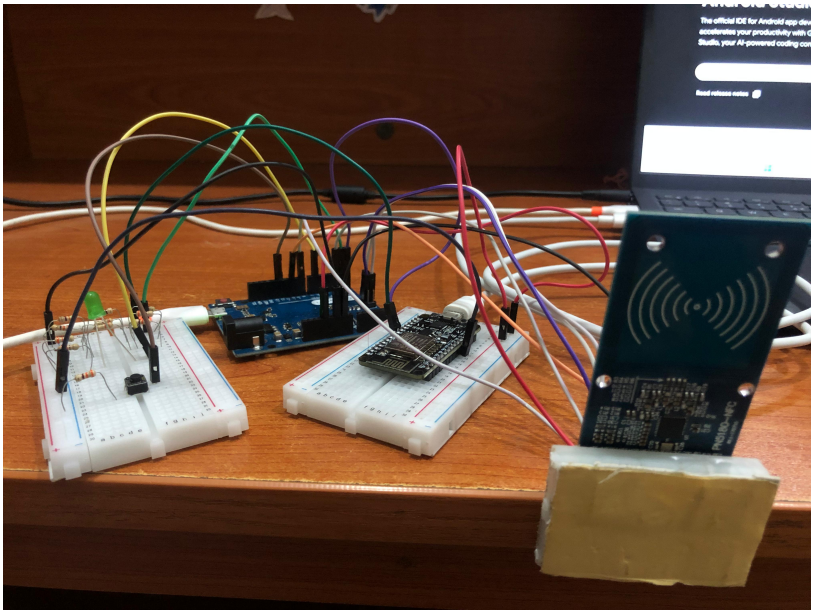


Figure 75: Module Testing - Breadboard Implementation

19 Software Development

19.1 Web App Development

We have finalized our web app, which will be used to connect to the database and access the RFID tag details. It is developed using Node.js.

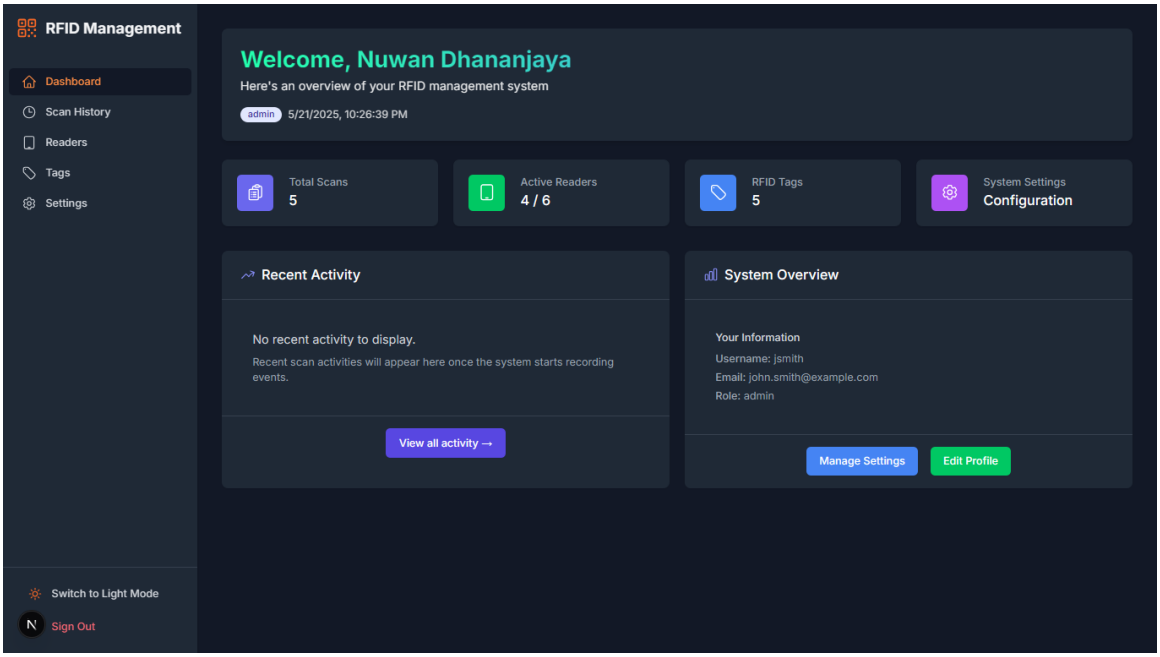


Figure 76: Web App - Dashboard

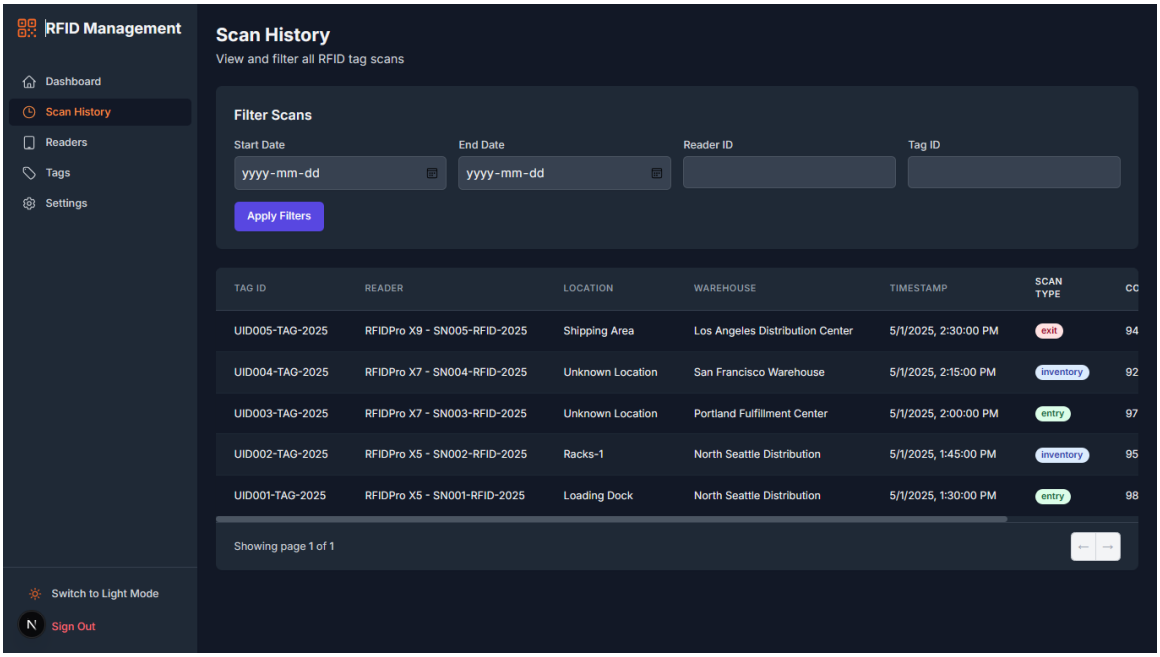


Figure 77: Web App - Scan History

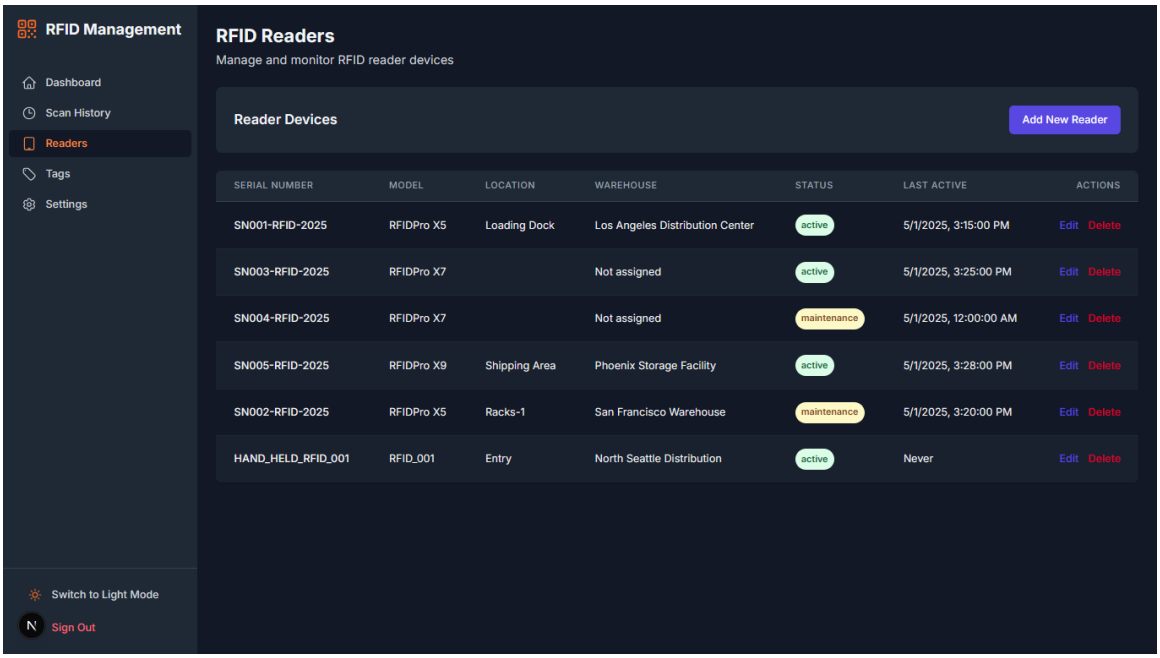


Figure 78: Web App - Tags

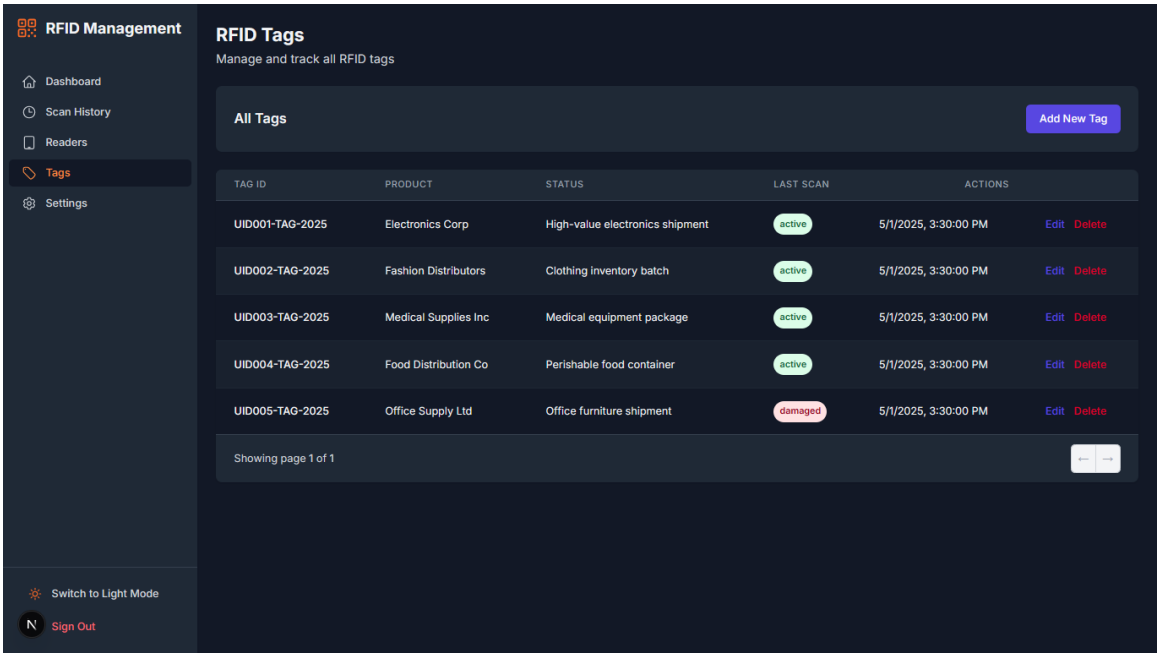


Figure 79: Web App - Dashboard

19.2 Mobile App Development

The mobile app is developed using Flutter.

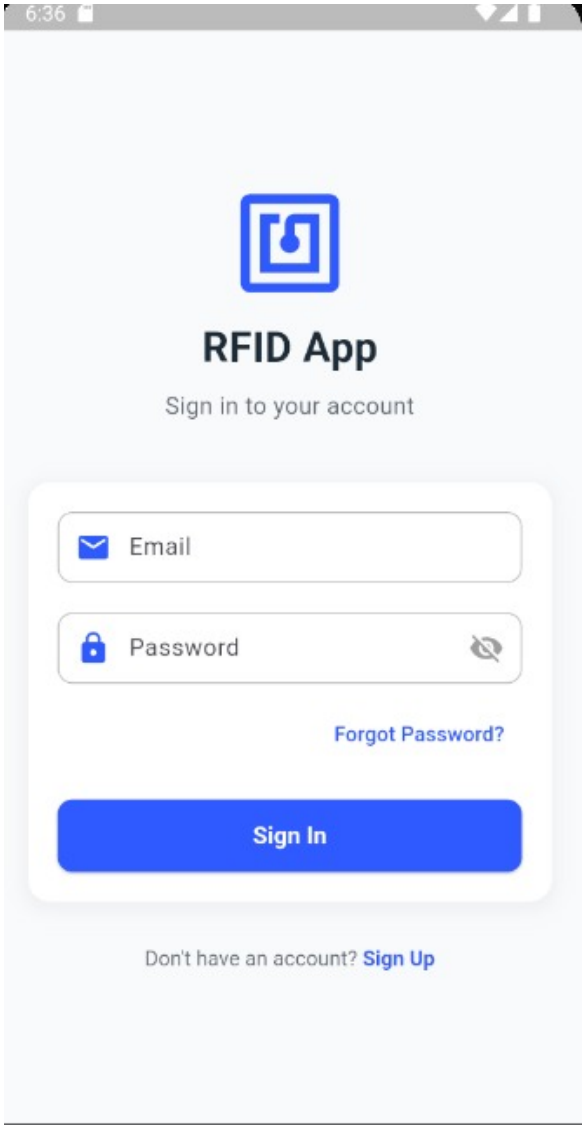


Figure 80: Mobile App - Login Page

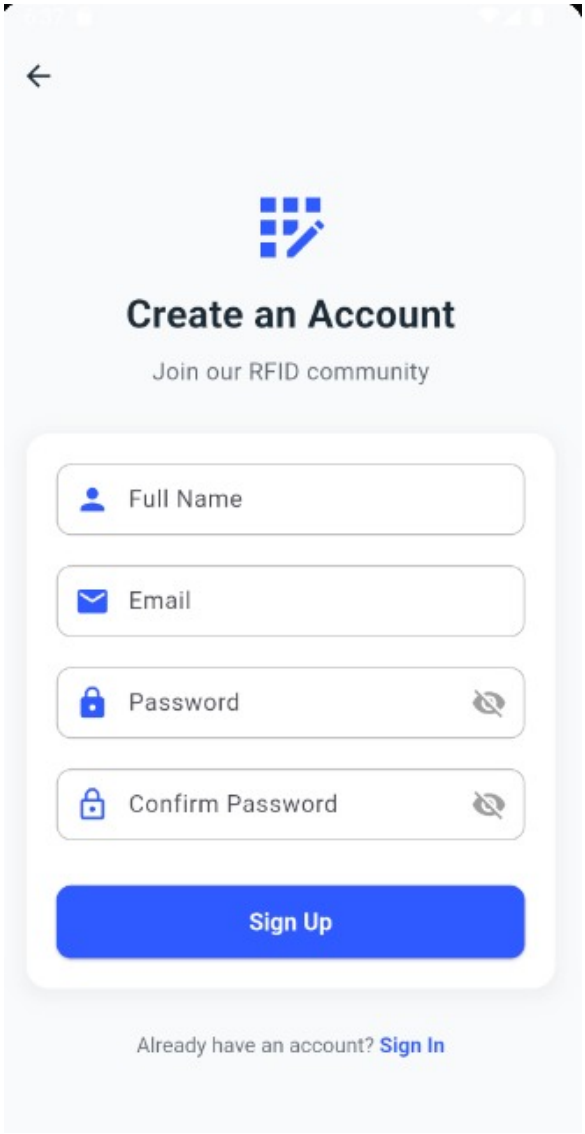


Figure 81: Mobile App - Sign up Page

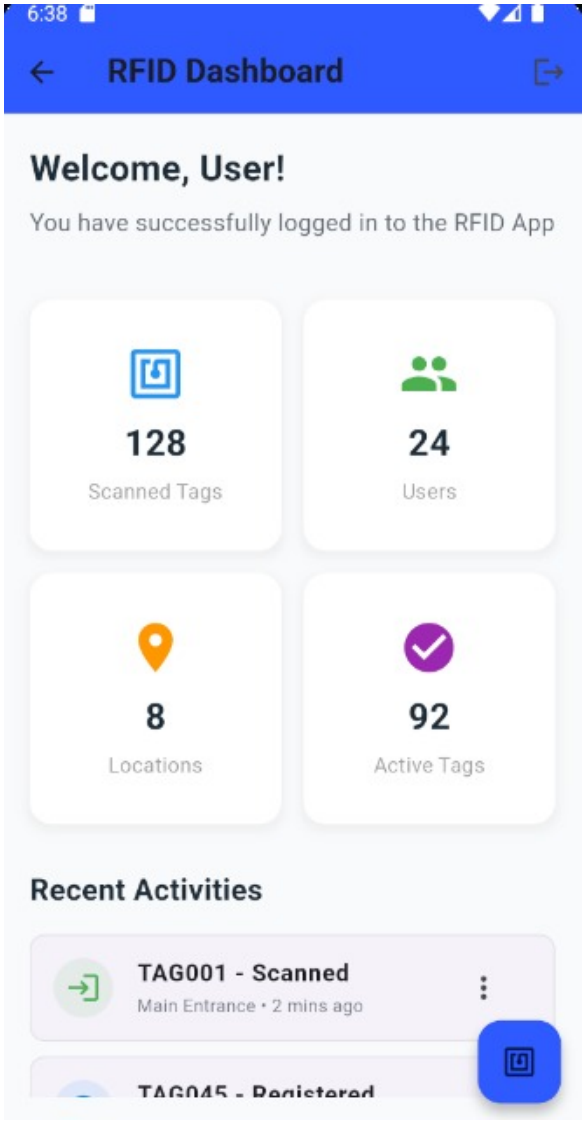


Figure 82: Mobile App - Dashboard

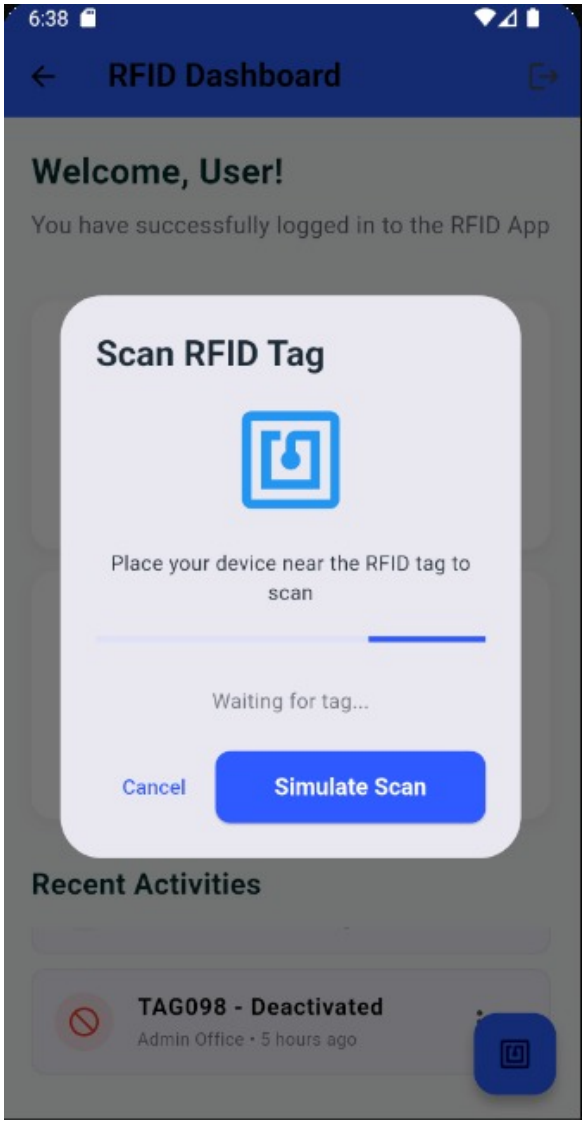


Figure 83: Mobile App - Scan RFID Tag

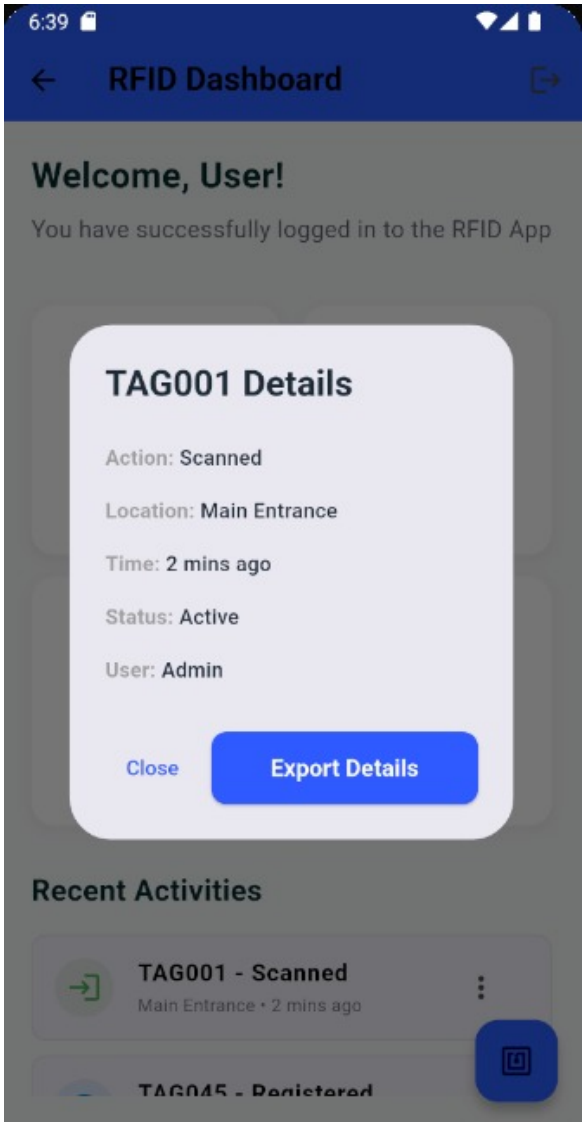


Figure 84: Mobile App - Tag Details

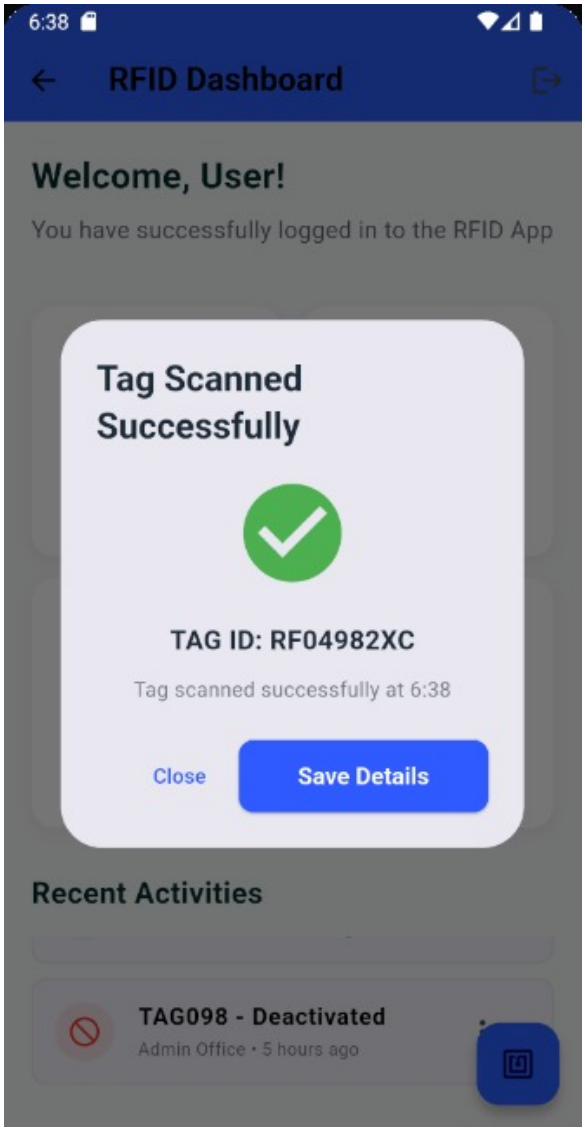


Figure 85: Mobile App - Tag Scanned Successfully

19.3 Database Integration

The database is implemented using Supabase(PostgreSQL).

Stage 3

20 Enclosure Design

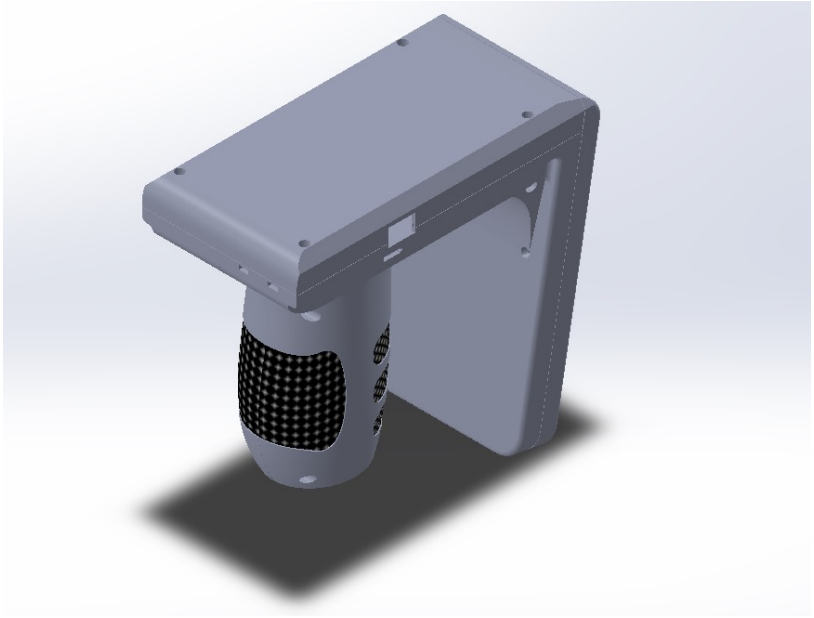


Figure 86: RFID Reader - Back View

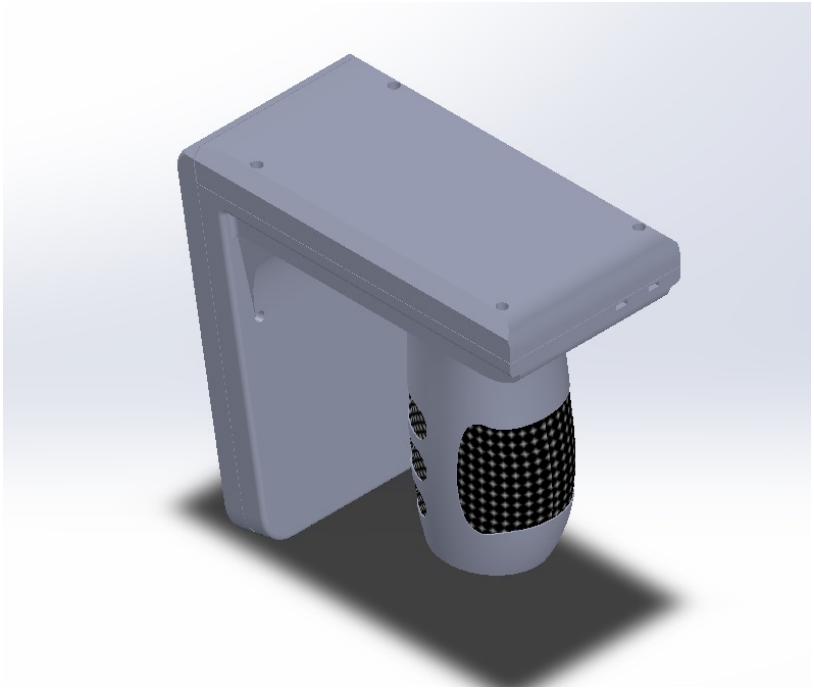


Figure 87: RFID Reader - Left View

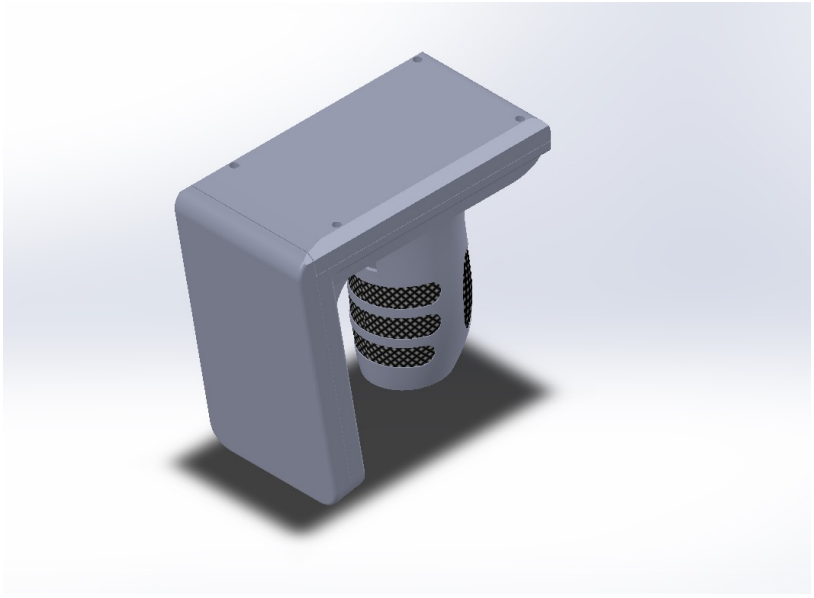


Figure 88: RFID Reader - Front View

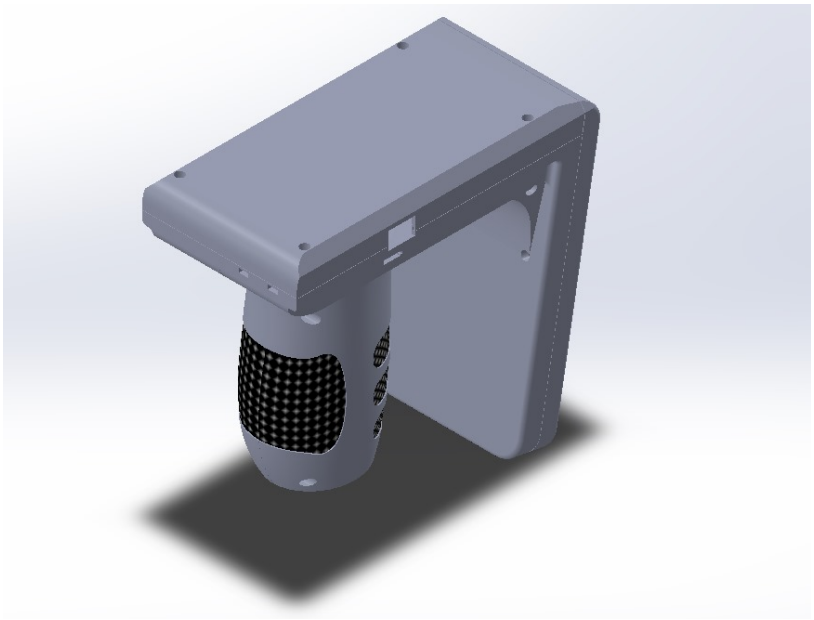


Figure 89: RFID Reader - Left View

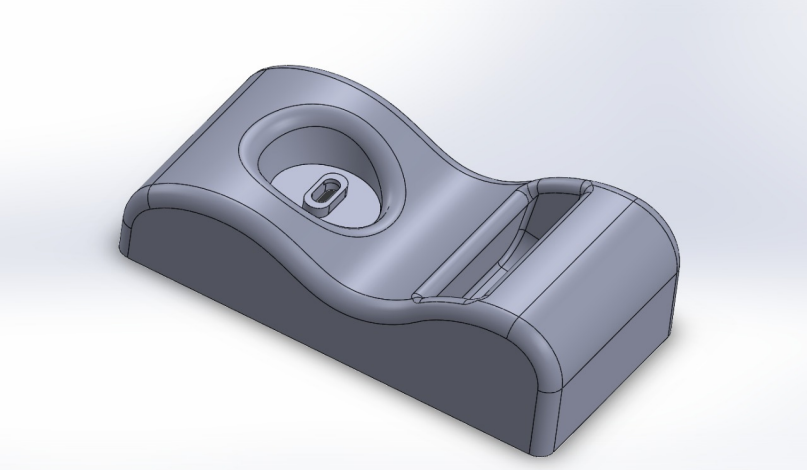


Figure 90: Charging Cradle

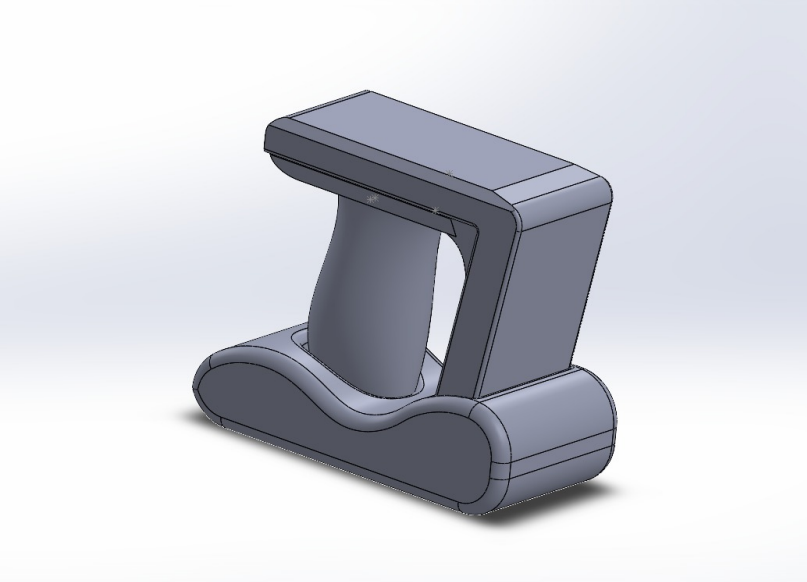


Figure 91: RFID Reader and Charging Cradle

21 Firmware Implementation

21.1 PN5180

21.1.1 Header File

```
#ifndef PN5180_H
#define PN5180_H

#include <SPI.h>

// https://www.nxp.com/docs/en/data-sheet/PN5180A0XX-C1-C2.pdf
#define PN5180_WRITE_REGISTER           (0x00)
#define PN5180_WRITE_REGISTER_OR_MASK   (0x01)
```

```
#define PN5180_WRITE_REGISTER_AND_MASK  (0x02)
#define WRITE_REGISTER_MULTIPLE                 (0x03)
#define PN5180_READ_REGISTER            (0x04)
#define READ_REGISTER_MULTIPLE                  (0x05)
#define PN5180_WRITE_EEPROM                             (0x06)
#define PN5180_READ_EEPROM              (0x07)
#define WRITE_TX_DATA                           (0x08)
#define PN5180_SEND_DATA                (0x09)
#define PN5180_READ_DATA                (0x0A)
#define PN5180_SWITCH_MODE              (0x0B)
#define MIFARE_AUTHENTICATE                     (0x0C)
#define EPC_INVENTORY                           (0x0D)
#define EPC_RESUME_INVENTORY                    (0x0E)
#define EPC_RETRIEVE_INVENTORY_RESULT_SIZE (0x0F)
#define EPC_RETRIEVE_INVENTORY_RESULT   (0x10)
#define PN5180_LOAD_RF_CONFIG           (0x11)
#define UPDATE_RF_CONFIG                        (0x12)
#define RETRIEVE_RF_CONFIG_SIZE                 (0x13)
#define RETRIEVE_RF_CONFIG                      (0x14)
// Reserved for future use                      (0x15)
#define PN5180_RF_ON                    (0x16)
#define PN5180_RF_OFF                   (0x17)

// PN5180 Registers
#define SYSTEM_CONFIG       (0x00)
#define IRQ_ENABLE          (0x01)
#define IRQ_STATUS          (0x02)
#define IRQ_CLEAR           (0x03)
#define TRANSCEIVE_CONTROL  (0x04)
#define TIMER1_RELOAD       (0x0c)
#define TIMER1_CONFIG       (0x0f)
#define RX_WAIT_CONFIG      (0x11)
#define CRC_RX_CONFIG       (0x12)
#define RX_STATUS           (0x13)
#define TX_WAIT_CONFIG      (0x17)
#define TX_CONFIG                       (0x18)
#define CRC_TX_CONFIG       (0x19)
#define RF_STATUS           (0x1d)
#define SYSTEM_STATUS       (0x24)
#define TEMP_CONTROL        (0x25)
#define AGC_REF_CONFIG          (0x26)

// PN5180 EEPROM Addresses
#define DIE_IDENTIFIER      (0x00)
#define PRODUCT_VERSION     (0x10)
#define FIRMWARE_VERSION    (0x12)
#define EEPROM_VERSION      (0x14)
#define IRQ_PIN_CONFIG      (0x1A)

enum PN5180TransceiveStat {
  PN5180_TS_Idle = 0,
  PN5180_TS_WaitTransmit = 1,
  PN5180_TS_Transmitting = 2,
  PN5180_TS_WaitReceive = 3,
  PN5180_TS_WaitForData = 4,
  PN5180_TS_Receiving = 5,
  PN5180_TS_LoopBack = 6,
  PN5180_TS_RESERVED = 7
};
```

```
68
69  // PN5180 IRQ_STATUS
70  #define RX_IRQ_STAT             (1<<0)  // End of RF receiption IRQ
71  #define TX_IRQ_STAT             (1<<1)  // End of RF transmission IRQ
72  #define IDLE_IRQ_STAT           (1<<2)  // IDLE IRQ
73  #define RFOFF_DET_IRQ_STAT      (1<<6)  // RF Field OFF detection IRQ
74  #define RFON_DET_IRQ_STAT       (1<<7)  // RF Field ON detection IRQ
75  #define TX_RFOFF_IRQ_STAT       (1<<8)  // RF Field OFF in PCD IRQ
76  #define TX_RFON_IRQ_STAT        (1<<9)  // RF Field ON in PCD IRQ
77  #define RX_SOF_DET_IRQ_STAT     (1<<14) // RF SOF Detection IRQ
78  #define GENERAL_ERROR_IRQ_STAT  (1<<17) // General error IRQ
79  #define LPCD_IRQ_STAT                   (1<<19) // LPCD Detection IRQ
80
81  class PN5180 {
82  private:
83    uint8_t PN5180_NSS;   // active low
84    uint8_t PN5180_BUSY;
85    uint8_t PN5180_RST;
86    SPIClass& PN5180_SPI;
87
88    SPISettings SPI_SETTINGS;
89    static uint8_t readBuffer[508];
90
91  public:
92    PN5180(uint8_t SSpin, uint8_t BUSYpin, uint8_t RSTpin, SPIClass& spi=SPI);
93
94    void begin();
95    void end();
96
97    /*
98     * PN5180 direct commands with host interface
99     */
100 public:
101   /* cmd 0x00 */
102   bool writeRegister(uint8_t reg, uint32_t value);
103   /* cmd 0x01 */
104   bool writeRegisterWithOrMask(uint8_t addr, uint32_t mask);
105   /* cmd 0x02 */
106   bool writeRegisterWithAndMask(uint8_t addr, uint32_t mask);
107
108   /* cmd 0x04 */
109   bool readRegister(uint8_t reg, uint32_t *value);
110
111   /* cmd 0x06 */
112   bool writeEEprom(uint8_t addr, uint8_t *buffer, uint8_t len);
113   /* cmd 0x07 */
114   bool readEEprom(uint8_t addr, uint8_t *buffer, int len);
115
116   /* cmd 0x09 */
117   bool sendData(uint8_t *data, int len, uint8_t validBits = 0);
118   /* cmd 0x0a */
119   uint8_t * readData(int len);
120   bool readData(uint8_t len, uint8_t *buffer);
121   /* prepare LPCD registers */
122   bool prepareLPCD();
123   /* cmd 0x0B */
124   bool switchToLPCD(uint16_t wakeupCounterInMs);
125   /* cmd 0x11 */
126   bool loadRFConfig(uint8_t txConf, uint8_t rxConf);
```

```
127
128   /* cmd 0x16 */
129   bool setRF_on();
130   /* cmd 0x17 */
131   bool setRF_off();
132
133   /*
134    * Helper functions
135    */
136 public:
137   void reset();
138
139   uint8_t commandTimeout = 250;
140   uint32_t getIRQStatus();
141   bool clearIRQStatus(uint32_t irqMask);
142
143   PN5180TransceiveStat getTransceiveState();
144
145
146   /*
147    * Private methods, called within an SPI transaction
148    */
149 private:
150   bool transceiveCommand(uint8_t *sendBuffer, size_t sendBufferLen, uint8_t
        ↪ *recvBuffer = 0, size_t recvBufferLen = 0);
151
152 };
153
154 #endif /* PN5180_H */
```

21.1.2 C++ Code

```
1  #include "PN5180.h"
2  #include <util/delay.h>
3  #include <string.h>
4
5  uint8_t PN5180::readBuffer[508];
6
7  PN5180::PN5180(uint8_t SSpin, uint8_t BUSYpin, uint8_t RSTpin) :
8    PN5180_NSS(SSpin),
9    PN5180_BUSY(BUSYpin),
10   PN5180_RST(RSTpin) {}
11
12 void PN5180::begin() {
13   // NSS, RST as output; BUSY as input
14   setOutput(PN5180_NSS);
15   setOutput(PN5180_RST);
16   setInput(PN5180_BUSY);
17   setHigh(PN5180_NSS);
18   setHigh(PN5180_RST);
19   SPI_init();
20 }
21
22 void PN5180::end() {
23   setHigh(PN5180_NSS);
24 }
25
26 bool PN5180::writeRegister(uint8_t reg, uint32_t value) {
```

```
27    uint8_t *p = (uint8_t*)&value;
28    uint8_t buf[6] = { PN5180_WRITE_REGISTER, reg, p[0], p[1], p[2], p[3] };
29    return transceiveCommand(buf, 6);
30  }
31
32  bool PN5180::writeRegisterWithOrMask(uint8_t reg, uint32_t mask) {
33    uint8_t *p = (uint8_t*)&mask;
34    uint8_t buf[6] = { PN5180_WRITE_REGISTER_OR_MASK, reg, p[0], p[1], p[2], p
        ↪ [3] };
35    return transceiveCommand(buf, 6);
36  }
37
38  bool PN5180::writeRegisterWithAndMask(uint8_t reg, uint32_t mask) {
39    uint8_t *p = (uint8_t*)&mask;
40    uint8_t buf[6] = { PN5180_WRITE_REGISTER_AND_MASK, reg, p[0], p[1], p[2],
        ↪ p[3] };
41    return transceiveCommand(buf, 6);
42  }
43
44  bool PN5180::readRegister(uint8_t reg, uint32_t *value) {
45    uint8_t cmd[2] = { PN5180_READ_REGISTER, reg };
46    return transceiveCommand(cmd, 2, (uint8_t*)value, 4);
47  }
48
49  bool PN5180::writeEEprom(uint8_t addr, uint8_t *buffer, uint8_t len) {
50    uint8_t cmd[len + 2];
51    cmd[0] = PN5180_WRITE_EEPROM;
52    cmd[1] = addr;
53    for (int i = 0; i < len; i++) cmd[2 + i] = buffer[i];
54    return transceiveCommand(cmd, len + 2);
55  }
56
57  bool PN5180::readEEprom(uint8_t addr, uint8_t *buffer, int len) {
58    if ((addr > 254) || ((addr+len) > 254)) return false;
59    uint8_t cmd[3] = { PN5180_READ_EEPROM, addr, (uint8_t)len };
60    return transceiveCommand(cmd, 3, buffer, len);
61  }
62
63  bool PN5180::sendData(uint8_t *data, int len, uint8_t validBits) {
64    if (len > 260) return false;
65    uint8_t buf[len + 2];
66    buf[0] = PN5180_SEND_DATA;
67    buf[1] = validBits;
68    for (int i = 0; i < len; i++) buf[2+i] = data[i];
69
70    writeRegisterWithAndMask(SYSTEM_CONFIG, 0xFFFFFFF8);
71    writeRegisterWithOrMask(SYSTEM_CONFIG, 0x00000003);
72
73    if (getTransceiveState() != PN5180_TS_WaitTransmit) return false;
74    return transceiveCommand(buf, len + 2);
75  }
76
77  bool PN5180::readData(uint8_t len, uint8_t *buffer) {
78    if (len > 508) return false;
79    uint8_t cmd[2] = { PN5180_READ_DATA, 0x00 };
80    return transceiveCommand(cmd, 2, buffer, len);
81  }
82
83  bool PN5180::loadRFConfig(uint8_t txConf, uint8_t rxConf) {
```

```
  uint8_t cmd[3] = { PN5180_LOAD_RF_CONFIG, txConf, rxConf };
  return transceiveCommand(cmd, 3);
}

bool PN5180::setRF_on() {
  uint8_t cmd[2] = { PN5180_RF_ON, 0x00 };
  if (!transceiveCommand(cmd, 2)) return false;
  return waitIRQ(TX_RFON_IRQ_STAT);
}

bool PN5180::setRF_off() {
  uint8_t cmd[2] = { PN5180_RF_OFF, 0x00 };
  if (!transceiveCommand(cmd, 2)) return false;
  return waitIRQ(TX_RFOFF_IRQ_STAT);
}

void PN5180::reset() {
  setLow(PN5180_RST);
  _delay_ms(1);
  setHigh(PN5180_RST);
  _delay_ms(5);
  waitIRQ(IDLE_IRQ_STAT);
}

uint32_t PN5180::getIRQStatus() {
  uint32_t irq;
  readRegister(IRQ_STATUS, &irq);
  return irq;
}

bool PN5180::clearIRQStatus(uint32_t mask) {
  return writeRegister(IRQ_CLEAR, mask);
}

PN5180TransceiveStat PN5180::getTransceiveState() {
  uint32_t rf;
  if (!readRegister(RF_STATUS, &rf)) return PN5180TransceiveStat(0);
  return PN5180TransceiveStat((rf >> 24) & 0x07);
}

bool PN5180::waitIRQ(uint32_t flag) {
  uint32_t start = millis();
  while (!(getIRQStatus() & flag)) {
    if ((millis() - start) > 500) return false;
  }
  clearIRQStatus(flag);
  return true;
}
```

21.2 PN5180ISO1443

21.2.1 Header File

```
#ifndef PN5180ISO14443_H
#define PN5180ISO14443_H

#include "PN5180.h"

```

```
6
7  class PN5180ISO14443 : public PN5180 {
8
9  public:
10   PN5180ISO14443(uint8_t SSpin, uint8_t BUSYpin, uint8_t RSTpin, SPIClass&
         ↪ spi=SPI);
11
12 private:
13   uint16_t rxBytesReceived();
14   uint32_t GetNumberOfBytesReceivedAndValidBits();
15 public:
16   // Mifare TypeA
17   int8_t activateTypeA(uint8_t *buffer, uint8_t kind);
18   bool mifareBlockRead(uint8_t blockno,uint8_t *buffer);
19   uint8_t mifareBlockWrite16(uint8_t blockno, uint8_t *buffer);
20   bool mifareHalt();
21   /*
22    * Helper functions
23    */
24 public:
25   bool setupRF();
26   int8_t readCardSerial(uint8_t *buffer);
27   bool isCardPresent();
28 };
29
30 #endif /* PN5180ISO14443_H */
```

21.2.2 C++ Code

```
1  #include <avr/io.h>
2  #include <util/delay.h>
3  #include "PN5180ISO14443.h"
4  #include "PN5180.h"
5  #include "Debug.h"
6
7  PN5180ISO14443::PN5180ISO14443(uint8_t SSpin, uint8_t BUSYpin, uint8_t
       ↪ RSTpin, SPIClass& spi)
8    : PN5180(SSpin, BUSYpin, RSTpin, spi) {}
9
10 bool PN5180ISO14443::setupRF() {
11   if (!loadRFConfig(0x00, 0x80)) return false;
12   return setRF_on();
13 }
14
15 uint16_t PN5180ISO14443::rxBytesReceived() {
16   uint32_t rxStatus;
17   readRegister(RX_STATUS, &rxStatus);
18   return (uint16_t)(rxStatus & 0x1FF);
19 }
20
21 int8_t PN5180ISO14443::activateTypeA(uint8_t* buffer, uint8_t kind) {
22   uint8_t cmd[7], uidLength = 0;
23   if (!loadRFConfig(0x00, 0x80)) return -1;
24   setRF_on();
25   _delay_ms(10);
26   if (!writeRegisterWithAndMask(SYSTEM_CONFIG, 0xFFFFFFBF)) return -1;
27   if (!writeRegisterWithAndMask(CRC_RX_CONFIG, 0xFFFFFFFE)) return -1;
28   if (!writeRegisterWithAndMask(CRC_TX_CONFIG, 0xFFFFFFFE)) return -1;
```

```
  if (!writeRegisterWithAndMask(SYSTEM_CONFIG, 0xFFFFFFF8)) return -1;
  if (!writeRegisterWithOrMask(SYSTEM_CONFIG, 0x03)) return -1;
  if (getTransceiveState() != PN5180_TS_WaitTransmit) return -1;
  clearIRQStatus(0xFFFFFFFF);
  cmd[0] = (kind == 0) ? 0x26 : 0x52;
  if (!sendData(cmd, 1, 0x07)) return 0;
  _delay_ms(10);
  if (!readData(2, buffer)) return 0;

  bool timeout = true;
  for (int i = 0; i < 200; i++) {
    if (getTransceiveState() == PN5180_TS_WaitTransmit) {
      timeout = false;
      break;
    }
    _delay_ms(1);
  }
  if (timeout) return -1;

  clearIRQStatus(0xFFFFFFFF);
  cmd[0] = 0x93; cmd[1] = 0x20;
  if (!sendData(cmd, 2, 0x00)) return -2;
  _delay_ms(5);
  if (rxBytesReceived() != 5) return -2;
  if (!readData(5, cmd + 2)) return -2;
  for (int i = 0; i < 4; i++) buffer[i] = cmd[2 + i];
  if (!writeRegisterWithOrMask(CRC_RX_CONFIG, 0x01)) return -2;
  if (!writeRegisterWithOrMask(CRC_TX_CONFIG, 0x01)) return -2;
  cmd[0] = 0x93; cmd[1] = 0x70;
  if (!sendData(cmd, 7, 0x00)) return 4;
  if (!readData(1, buffer + 2)) return -2;
  if ((buffer[2] & 0x04) == 0) {
    for (int i = 0; i < 4; i++) buffer[3 + i] = cmd[2 + i];
    uidLength = 4;
  } else {
    if (cmd[2] != 0x88) return 0;
    for (int i = 0; i < 3; i++) buffer[3 + i] = cmd[3 + i];
    writeRegisterWithAndMask(CRC_RX_CONFIG, 0xFFFFFFFE);
    writeRegisterWithAndMask(CRC_TX_CONFIG, 0xFFFFFFFE);
    cmd[0] = 0x95; cmd[1] = 0x20;
    if (!sendData(cmd, 2, 0x00)) return -2;
    if (!readData(5, cmd + 2)) return -2;
    for (int i = 0; i < 4; i++) buffer[6 + i] = cmd[2 + i];
    writeRegisterWithOrMask(CRC_RX_CONFIG, 0x01);
    writeRegisterWithOrMask(CRC_TX_CONFIG, 0x01);
    cmd[0] = 0x95; cmd[1] = 0x70;
    if (!sendData(cmd, 7, 0x00)) return -2;
    if (!readData(1, buffer + 2)) return -2;
    uidLength = 7;
  }
  return uidLength;
}

bool PN5180ISO14443::mifareBlockRead(uint8_t blockno, uint8_t* buffer) {
  uint8_t cmd[2] = { 0x30, blockno };
  if (!sendData(cmd, 2, 0x00)) return false;
  _delay_ms(5);
  return (rxBytesReceived() == 16 && readData(16, buffer));
}
```

```
88
89  uint8_t PN5180ISO14443::mifareBlockWrite16(uint8_t blockno, uint8_t* buffer)
        ↪ {
90    uint8_t cmd[2] = { 0xA0, blockno };
91    writeRegisterWithAndMask(CRC_RX_CONFIG, 0xFFFFFFFE);
92    sendData(cmd, 2, 0x00);
93    readData(1, cmd);
94    sendData(buffer, 16, 0x00);
95    _delay_ms(10);
96    readData(1, cmd);
97    writeRegisterWithOrMask(CRC_RX_CONFIG, 0x01);
98    return cmd[0];
99  }
100
101 bool PN5180ISO14443::mifareHalt() {
102   uint8_t cmd[2] = { 0x50, 0x00 };
103   sendData(cmd, 2, 0x00);
104   return true;
105 }
106
107 int8_t PN5180ISO14443::readCardSerial(uint8_t* buffer) {
108   uint8_t response[10] = { 0 };
109   int8_t uidLength = activateTypeA(response, 0);
110   if (uidLength < 4 || (response[0] == 0xFF && response[1] == 0xFF)) return
        ↪ 0;
111   if (response[3] == 0x00 || response[3] == 0xFF) return 0;
112   bool validUID = false;
113   for (int i = 1; i < uidLength; i++) {
114     if (response[i + 3] != 0x00 && response[i + 3] != 0xFF) {
115       validUID = true;
116       break;
117     }
118   }
119   if ((uidLength == 4 && response[3] == 0x88) ||
120       (uidLength == 7 && (response[6] == 0x88 ||
121       (response[6] == 0x00 && response[7] == 0x00 && response[8] == 0x00 &&
            ↪ response[9] == 0x00) ||
122       (response[6] == 0xFF && response[7] == 0xFF && response[8] == 0xFF &&
            ↪ response[9] == 0xFF)))) {
123     validUID = false;
124   }
125   if (validUID) {
126     for (int i = 0; i < uidLength; i++) buffer[i] = response[i + 3];
127     return uidLength;
128   }
129   return 0;
130 }
131
132 bool PN5180ISO14443::isCardPresent() {
133   uint8_t buffer[10];
134   return (readCardSerial(buffer) >= 4);
135 }
```

21.3 SPI Communication

21.3.1 Header File

```h
#ifndef SPI_H
#define SPI_H

#include <stdint.h>
#include <stddef.h>

void SPI_init(void);
void SPI_setConfig(void);
void SPI_transfer(uint8_t *buffer, size_t length);
void SPI_end(void);
void SPI_testCommunication(void);

#endif
```

21.3.2 C++ Code

```cpp
#include <avr/io.h>
#include "mySPI.h"
#include <Arduino.h>

#define F_CPU 16000000UL

void SPI_init(void) {
    // Set MOSI (PB2), SCK (PB1), and NSS (PD1) as outputs; MISO (PB3) as input
    DDRB |= (1 << DDB2) | (1 << DDB1);
    DDRD |= (1 << DDD1);
    DDRB &= ~(1 << DDB3);

    // Configure SPI: Master, Mode 0 (CPOL=0, CPHA=0), 125 kHz (F_CPU/128)
    SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
    SPSR &= ~(1 << SPI2X);
    // Serial.println("intialized");
}

void SPI_setConfig(void) {
    // Reapply SPI settings (for compatibility with PN5180s fixed settings)
    SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
    SPSR &= ~(1 << SPI2X);
    // Serial.println("intialized again");
}

void SPI_transfer(uint8_t *buffer, size_t length) {
    for (size_t i = 0; i < length; i++) {
        SPDR = buffer[i]; // Write byte to SPDR
        while (!(SPSR & (1 << SPIF))); // Wait for transmission complete
        buffer[i] = SPDR; // Read received byte
        // Serial.println(buffer[i]);
    }
}

void SPI_end(void) {
    SPCR &= ~(1 << SPE); // Disable SPI
}


void SPI_testCommunication(void) {
```

```
    SPI_init();

    uint8_t data = 0x55; // Arbitrary test byte
    uint8_t received;

    // Pull NSS (PD1) low to start communication
    PORTD &= ~(1 << PORTD1);
    delay(1);

    // Send and receive 1 byte
    SPDR = data;
    while (!(SPSR & (1 << SPIF)));
    received = SPDR;
    Serial.println(received);

    // Pull NSS high to end communication
    PORTD |= (1 << PORTD1);

}
```

21.4 UART Communication

21.4.1 Header File

```
#ifndef MY_UART_H
#define MY_UART_H

#include <stdint.h>
#include <Arduino.h> // For String

#define RX_BUFFER_SIZE 64
#define MAX_BUFFER_SIZE 100

extern volatile uint8_t rx_buffer[RX_BUFFER_SIZE];
extern volatile uint16_t rx_count;
extern volatile uint8_t uart_tx_busy;
extern char rxBuffer[MAX_BUFFER_SIZE];
extern uint8_t index;

void USART_Init();
void USART_Transmit(uint8_t data);
uint8_t USART_Receive();
void USART_TxString(const char* str);
String USART_ReadString();
void USART_TransmitCommand(const char* cmd); // New
int USART_ReceiveCommand(); // New

#endif
```

21.4.2 C++ Code

```
#include "myUART.h"

volatile static uint8_t rx_buffer[RX_BUFFER_SIZE] = {0};
volatile static uint16_t rx_count = 0;
volatile static uint8_t uart_tx_busy = 1;
```

```
6
7  #define MAX_BUFFER_SIZE 100
8
9  char rxBuffer[MAX_BUFFER_SIZE];
10 uint8_t index = 0;
11
12
13 //Enable usart in 8-bit, 9600, normal speed mode
14 void USART_Init(){
15  // Enable USART receive and transmit
16  UCSR1B |= (1 << RXEN1) | (1 << TXEN1);
17  // Asynchronous mode, no parity, 1 stop bit
18  UCSR1C &= ~(1 << UMSEL11) & ~(1 << UMSEL10) & ~(1 << UPM11) & ~(1 << UPM10)
        ↪  & ~(1 << USBS1);
19  // 8-bit data
20  UCSR1C |= (1 << UCSZ11) | (1 << UCSZ10);
21  UCSR1B &= ~(1 << UCSZ12);
22  // Normal speed
23  UCSR1A &= ~(1 << U2X1);
24  // Set baud rate to 9600 (16 MHz clock -> UBRR1 = 103)
25  UBRR1 = 103;
26 }
27
28
29 void USART_Transmit(uint8_t data) {
30         while (!(UCSR1A & (1 << UDRE1)));  // Wait until buffer empty
31         UDR1 = data;
32 }
33
34 // Receive one byte
35 uint8_t USART_Receive() {
36         while (!(UCSR1A & (1 << RXC1)));  // Wait for data
37         return UDR1;
38 }
39
40 void USART_TxString(const char* str) {
41         while (*str) {
42                 USART_Transmit(*str++);
43         }
44 }
45
46
47 String USART_ReadString() {
48   uint8_t index = 0;
49   uint32_t timeout = 10000; // ~10ms at 16 MHz
50   while (timeout > 0) {
51     if (UCSR1A & (1 << RXC1)) {
52       char c = USART_Receive();
53       if (c == '\n') {
54         rxBuffer[index] = '\0';
55         return String(rxBuffer);
56       } else if (index < MAX_BUFFER_SIZE - 1) {
57         rxBuffer[index++] = c;
58       } else {
59         rxBuffer[0] = '\0';
60         return String("");
61       }
62     }
63     timeout--;
```

```
  }
  rxBuffer[0] = '\0';
  return String(""); // Timeout, no complete message
}


int USART_ReceiveCommand() {
  String response = USART_ReadString();
  Serial.println(response);
  if (response.length() == 0) {
    return 0; // No complete message received
  }

  // Compare response against expected commands
  if (response == "OK") {
    return 1; // Success (e.g., WiFi connected)
  } else if (response == "ERROR") {
    return 2; // Failure (e.g., WiFi failed)
  } else if (response.startsWith("DA")) {
    return 3; // Data received (e.g., server response)
  } else {
    return -1; // Unknown response
  }
}

void USART_TransmitCommand(const char* cmd) {
  char buffer[MAX_BUFFER_SIZE] = {0};
  uint8_t len = strlen(cmd);

  // Check if cmd ends with '\n'
  if (len > 0 && cmd[len - 1] != '\n') {
    // Copy cmd and add '\n'
    snprintf(buffer, MAX_BUFFER_SIZE - 1, "CMD:%s\n", cmd);
  } else {
    // Copy cmd as-is with prefix
    snprintf(buffer, MAX_BUFFER_SIZE - 1, "CMD:%s", cmd);
  }

  // Ensure buffer is null-terminated
  buffer[MAX_BUFFER_SIZE - 1] = '\0';

  // Send via UART
  USART_TxString(buffer);
}
```

22 Software Development

Now the mobile app is used only as the interface to access the database base, just like the web app. There is no tag scanning capability.

22.1 Mobile App

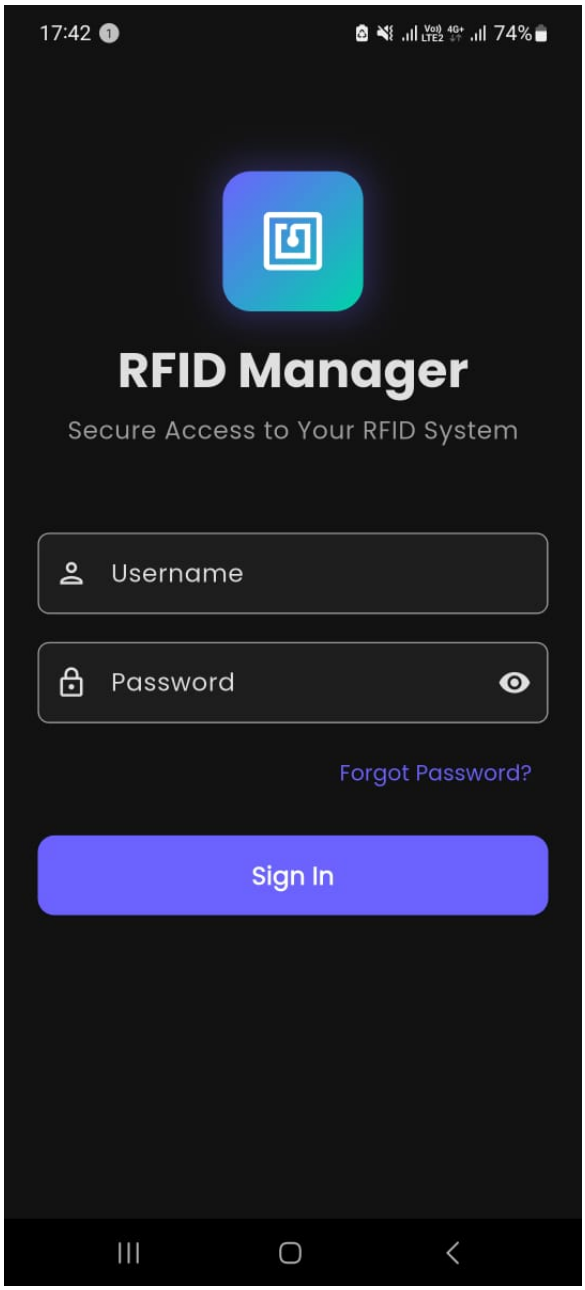


Figure 92: Mobile App - Login Page

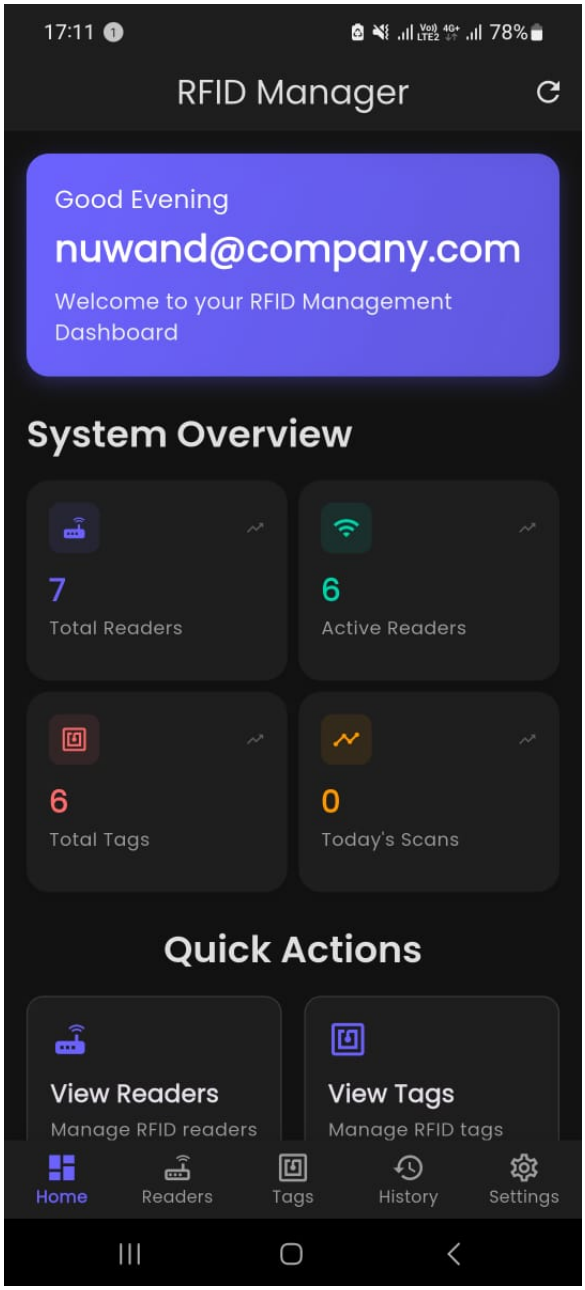


Figure 93: Mobile App - Home Page

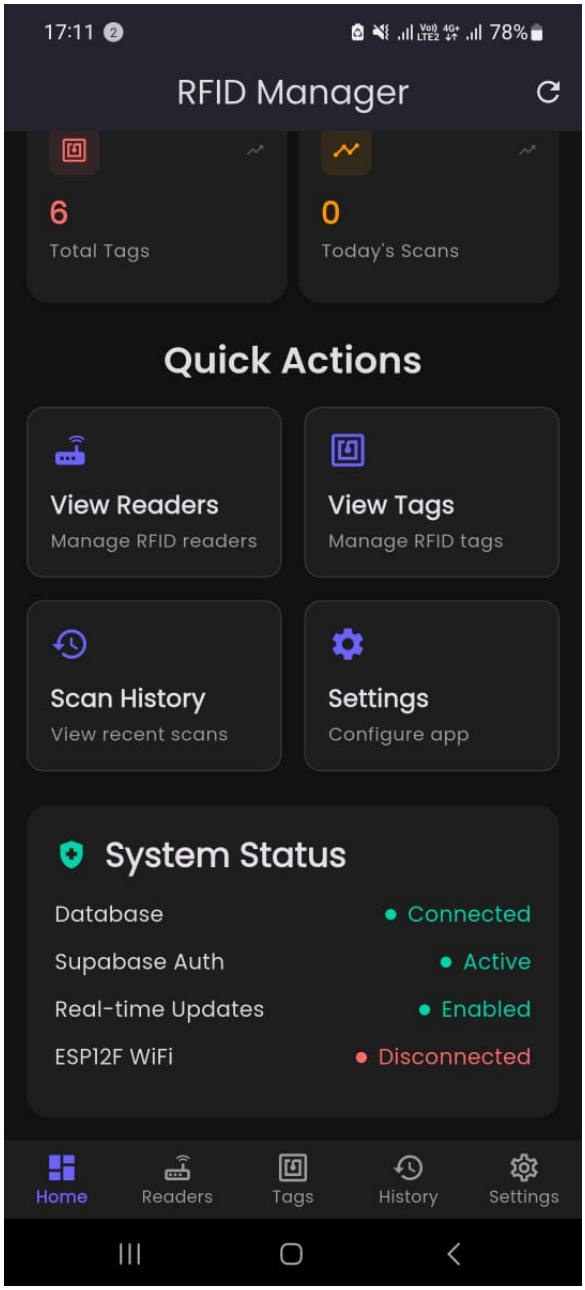


Figure 94: Mobile App - Home Page

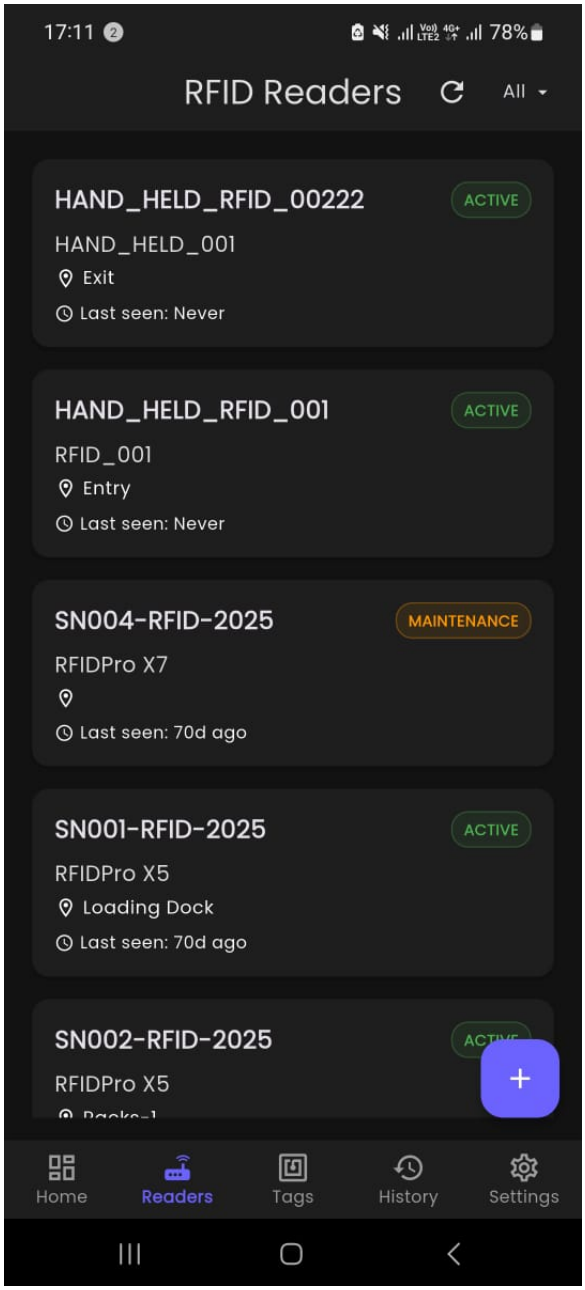


Figure 95: Mobile App - Readers Page

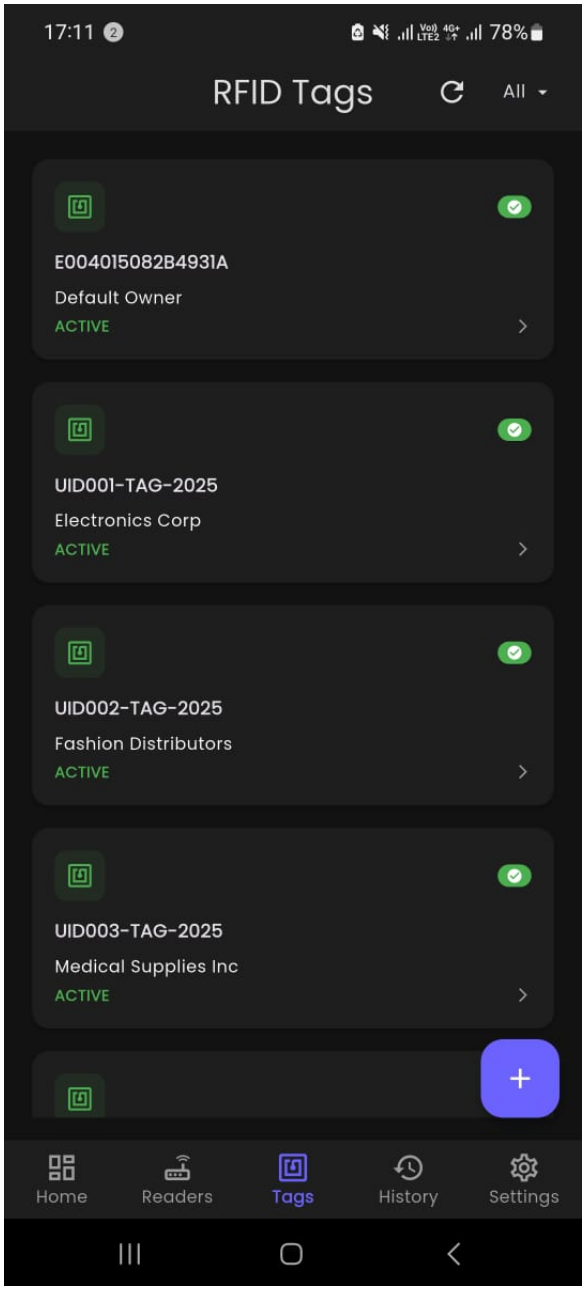


Figure 96: Mobile App - Tags Page

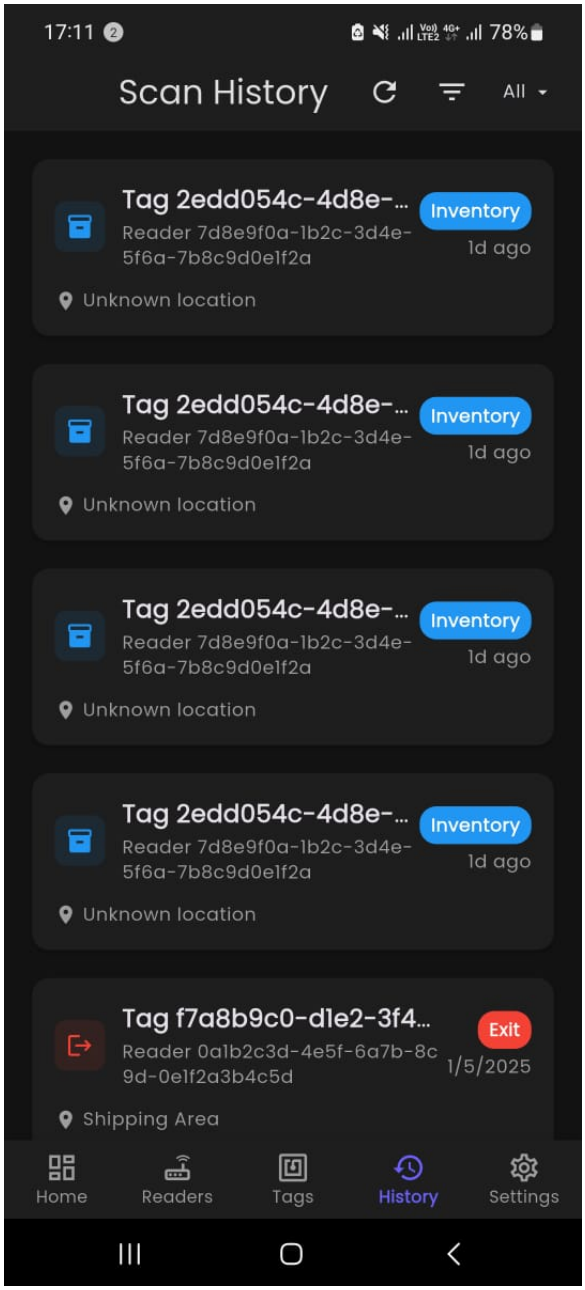


Figure 97: Mobile App - History Page

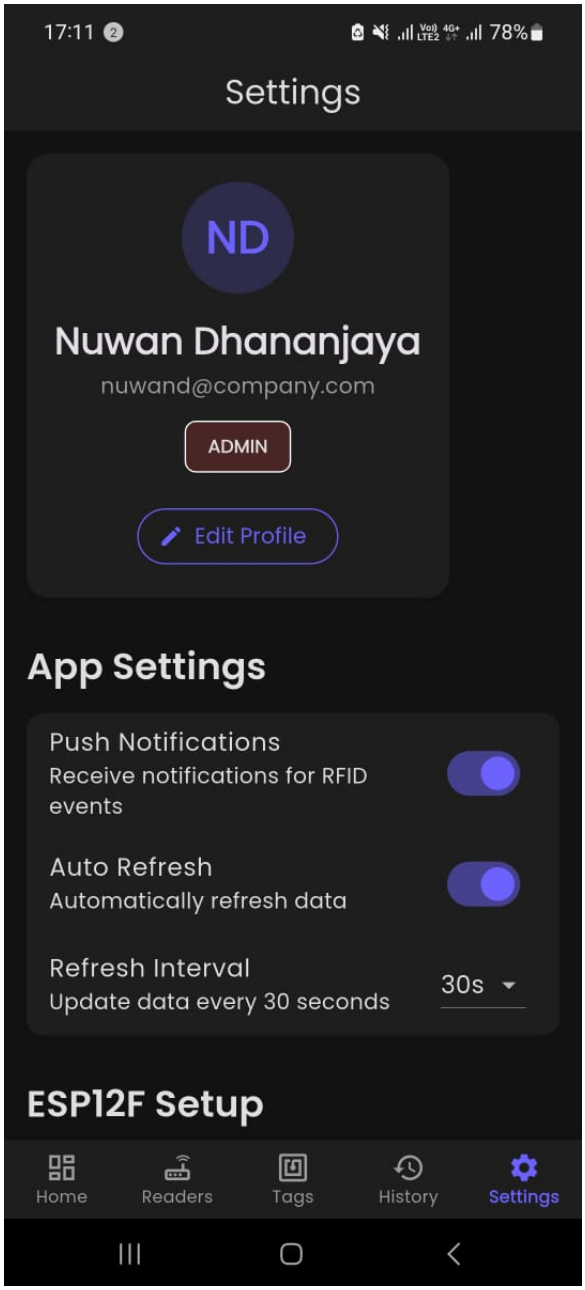


Figure 98: Mobile App - Settings Page

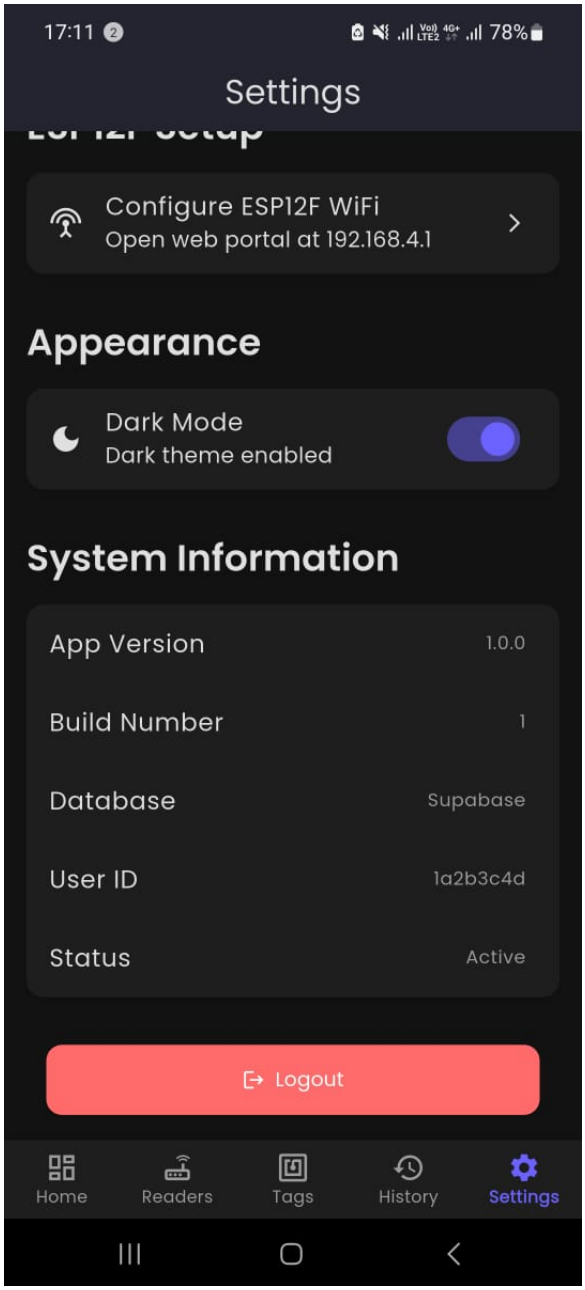


Figure 99: Mobile App - Settings Page

22.2 Web App

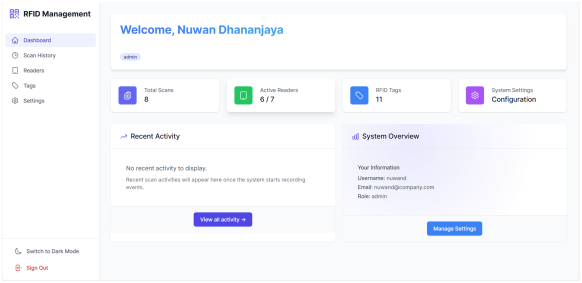


Figure 100: Web App - Dashboard

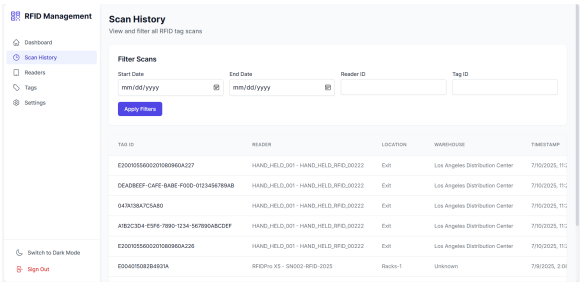


Figure 101: Web App - Scan History

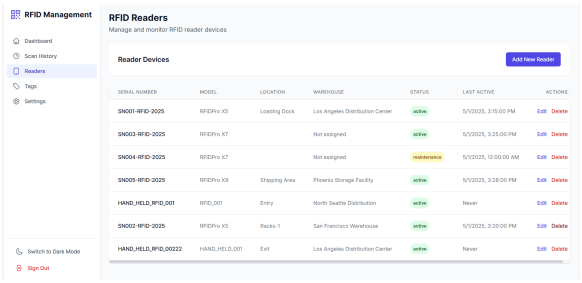


Figure 102: Web App - Readers Page

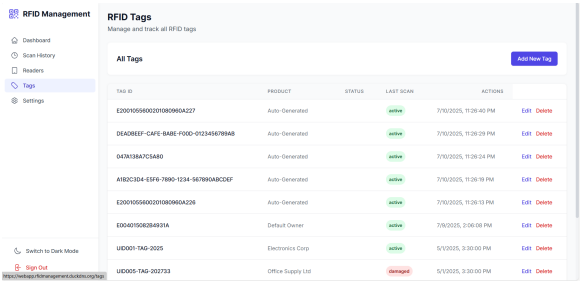


Figure 103: Web App - Tags Page

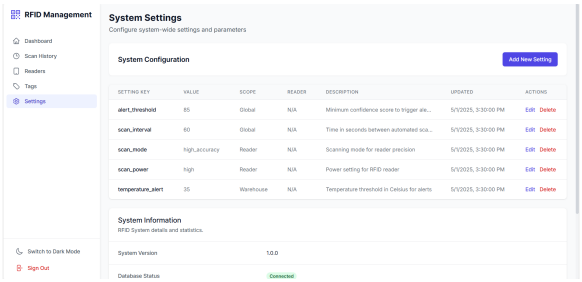


Figure 104: Web App - Settings

23 Conclusion

The design and successful completion of the handheld RFID reader project provided our team with invaluable hands-on experience in overcoming real-world engineering challenges and making informed, practical decisions. From the initial stages of market research and industrial visits to finalizing component selection and system integration, each phase was guided by a strong focus on cost-efficiency, user-friendliness, and technical viability. Completing this project deepened our understanding of the full product development life cycle. It highlighted the importance of aligning engineering principles with real-world constraints to deliver a functional, reliable, and user-centered solution.