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

Software 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

5/22/2025

Contents

1 PCB Design

Altium Designer - https://www.altium.com/education/students

2 Enclosure Design

SolidWorks

3 Web App Development

The web app was developed using Node.js.

Link: https://uniofmoramy.sharepoint.com/:f:/g/personal/wickramasinghesd$_2$2$_u om_l k/Emx1S-QYZOlAi52qVPWJZnoBJQriCgSUmRu-N7Q6K0C1tQ?e=sf9qnY$

4 Database Implementation

The database was implemented using Supabase(PostgreSQL).

5 Schema

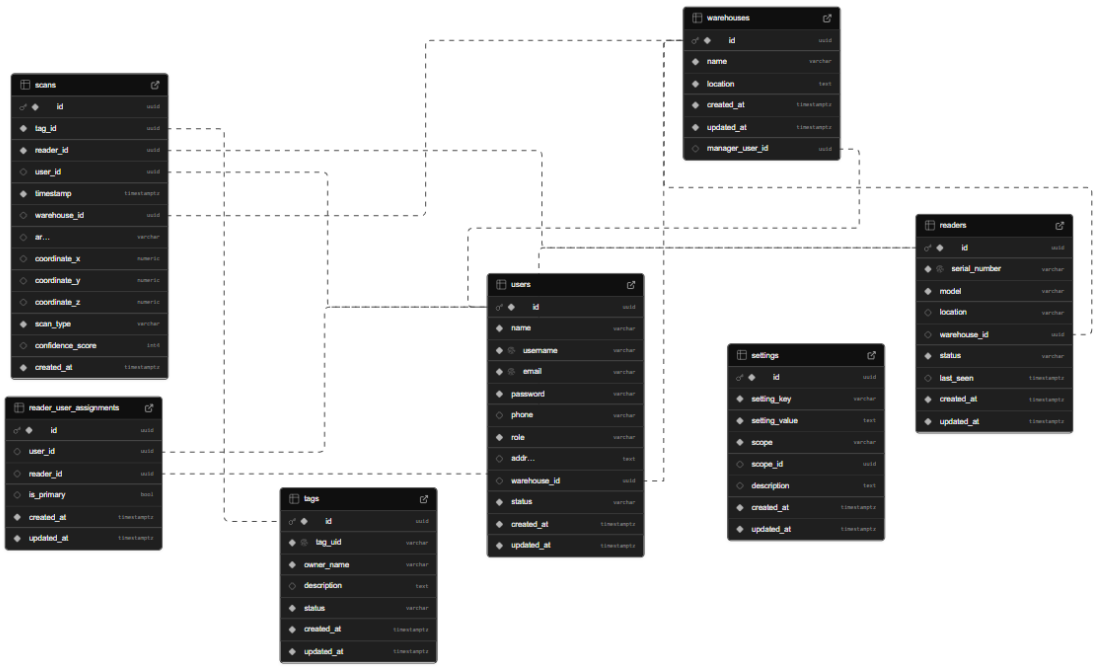

Figure 1: Schema

```
    -- Drop tables if they exist for clean migration
DROP TABLE IF EXISTS settings CASCADE;
DROP TABLE IF EXISTS scans CASCADE;
DROP TABLE IF EXISTS tags CASCADE;
DROP TABLE IF EXISTS reader_user_assignments CASCADE;
DROP TABLE IF EXISTS readers CASCADE;
DROP TABLE IF EXISTS users CASCADE;
DROP TABLE IF EXISTS warehouses CASCADE;

-- Create warehouses table (removed the manager_user_id initially to
  ↪ avoid circular reference)
CREATE TABLE warehouses (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    location TEXT NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Create users table
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    username VARCHAR(100) NOT NULL UNIQUE,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    phone VARCHAR(20),
    role VARCHAR(20) NOT NULL CHECK (role IN ('admin', 'manager', '
      ↪ employee')),
    address TEXT,
    warehouse_id UUID REFERENCES warehouses(id) ON DELETE SET NULL,
    status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('
      ↪ active', 'suspended', 'inactive')),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Now add the manager_user_id to warehouses after users table exists
ALTER TABLE warehouses ADD COLUMN manager_user_id UUID;
ALTER TABLE warehouses ADD CONSTRAINT fk_warehouses_manager
    FOREIGN KEY (manager_user_id) REFERENCES users(id) ON DELETE SET
      ↪ NULL;

-- Create readers table
CREATE TABLE readers (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    serial_number VARCHAR(100) NOT NULL UNIQUE,
    model VARCHAR(100) NOT NULL,
    location VARCHAR(255) NOT NULL,
    warehouse_id UUID REFERENCES warehouses(id) ON DELETE SET NULL,
    status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('
      ↪ active', 'maintenance', 'retired')),
    last_seen TIMESTAMPTZ,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Create reader_user_assignments table for many-to-many relationship
-- This is better than having assigned_user_id in readers table when
  ↪ multiple users can be assigned
CREATE TABLE reader_user_assignments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    reader_id UUID REFERENCES readers(id) ON DELETE CASCADE,
    is_primary BOOLEAN DEFAULT false,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    UNIQUE(reader_id, user_id)
);

-- Create tags table
CREATE TABLE tags (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tag_uid VARCHAR(100) NOT NULL UNIQUE,
    owner_name VARCHAR(255) NOT NULL,
    description TEXT,
    status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('
        ↪ active', 'damaged', 'retired')),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Create scans table
CREATE TABLE scans (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tag_id UUID NOT NULL REFERENCES tags(id) ON DELETE CASCADE,
    reader_id UUID NOT NULL REFERENCES readers(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE SET NULL,
    timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    warehouse_id UUID REFERENCES warehouses(id) ON DELETE SET NULL,
    area VARCHAR(255),
    coordinate_x DECIMAL(10, 2),
    coordinate_y DECIMAL(10, 2),
    coordinate_z DECIMAL(10, 2),
    scan_type VARCHAR(20) NOT NULL CHECK (scan_type IN ('entry', 'exit',
        ↪  'inventory')),
    confidence_score INTEGER CHECK (confidence_score BETWEEN 0 AND 100),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Create settings table
CREATE TABLE settings (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    setting_key VARCHAR(100) NOT NULL,
    setting_value TEXT NOT NULL,
    scope VARCHAR(50) NOT NULL DEFAULT 'global' CHECK (scope IN ('global
        ↪ ', 'reader', 'warehouse')),
    scope_id UUID, -- This will store reader_id or warehouse_id based on
        ↪  scope
    description TEXT,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Add a partial unique constraint that treats NULL scope_id as a
    ↪ specific value
CREATE UNIQUE INDEX idx_settings_unique ON settings (setting_key, scope,
    ↪  COALESCE(scope_id, '00000000-0000-0000-0000-000000000000'::UUID))
    ↪ ;

-- Create functions and triggers for updated_at timestamps
```

```sql
    CREATE OR REPLACE FUNCTION update_timestamp()
    RETURNS TRIGGER AS $$
    BEGIN
        NEW.updated_at = NOW();
        RETURN NEW;
    END;
    $$ language 'plpgsql';

    -- Create triggers for each table
    CREATE TRIGGER update_warehouses_timestamp BEFORE UPDATE ON warehouses
        FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
    CREATE TRIGGER update_users_timestamp BEFORE UPDATE ON users
        FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
    CREATE TRIGGER update_readers_timestamp BEFORE UPDATE ON readers
        FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
    CREATE TRIGGER update_tags_timestamp BEFORE UPDATE ON tags
        FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
    CREATE TRIGGER update_settings_timestamp BEFORE UPDATE ON settings
        FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
    CREATE TRIGGER update_reader_user_assignments_timestamp BEFORE UPDATE ON
        ↪  reader_user_assignments
        FOR EACH ROW EXECUTE PROCEDURE update_timestamp();

    -- Create indexes for performance
    CREATE INDEX idx_scans_timestamp ON scans(timestamp);
    CREATE INDEX idx_scans_tag_id ON scans(tag_id);
    CREATE INDEX idx_scans_reader_id ON scans(reader_id);
    CREATE INDEX idx_readers_warehouse ON readers(warehouse_id);
    CREATE INDEX idx_users_warehouse ON users(warehouse_id);
    CREATE INDEX idx_tags_status ON tags(status);
    -- We already have the unique index that acts as our index for these
        ↪ columns
    CREATE INDEX idx_reader_assignments_user ON reader_user_assignments(
        ↪ user_id);
    CREATE INDEX idx_reader_assignments_reader ON reader_user_assignments(
        ↪ reader_id);

    -- Create RLS (Row Level Security) policies for Supabase
    -- Enable RLS on all tables
    ALTER TABLE warehouses ENABLE ROW LEVEL SECURITY;
    ALTER TABLE users ENABLE ROW LEVEL SECURITY;
    ALTER TABLE readers ENABLE ROW LEVEL SECURITY;
    ALTER TABLE tags ENABLE ROW LEVEL SECURITY;
    ALTER TABLE scans ENABLE ROW LEVEL SECURITY;
    ALTER TABLE settings ENABLE ROW LEVEL SECURITY;
    ALTER TABLE reader_user_assignments ENABLE ROW LEVEL SECURITY;

    -- Comprehensive RLS policies

    -- Warehouse policies
    CREATE POLICY "Admins can do anything with warehouses" ON warehouses
        FOR ALL TO authenticated
        USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));

    CREATE POLICY "Managers can update their warehouses" ON warehouses
        FOR UPDATE TO authenticated
        USING (auth.uid() = manager_user_id);

    CREATE POLICY "All users can view warehouses" ON warehouses
        FOR SELECT TO authenticated
        USING (true);

    -- User policies
    CREATE POLICY "Admins can do anything with users" ON users
        FOR ALL TO authenticated
        USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));

    CREATE POLICY "Managers can view and update users in their warehouse" ON users
        FOR SELECT TO authenticated
        USING (
            (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))
            AND
            (warehouse_id IN (SELECT warehouse_id FROM users WHERE id = auth.uid())
             OR warehouse_id IN (SELECT id FROM warehouses WHERE manager_user_id = auth.uid()))
        );

    CREATE POLICY "Users can view their own data" ON users
        FOR SELECT TO authenticated
        USING (id = auth.uid());

    -- Reader policies
    CREATE POLICY "Admins can do anything with readers" ON readers
        FOR ALL TO authenticated
        USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));

    CREATE POLICY "Managers can manage readers in their warehouse" ON readers
        FOR ALL TO authenticated
        USING (
            (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))
            AND
            (warehouse_id IN (SELECT id FROM warehouses WHERE manager_user_id = auth.uid()))
        );

    CREATE POLICY "Users can view readers they are assigned to" ON readers
        FOR SELECT TO authenticated
        USING (
            id IN (
                SELECT reader_id FROM reader_user_assignments
                WHERE user_id = auth.uid()
            )
        );

    -- Reader Assignment policies
    CREATE POLICY "Admins can do anything with reader assignments" ON reader_user_assignments
        FOR ALL TO authenticated
        USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));

    CREATE POLICY "Managers can manage reader assignments in their warehouse " ON reader_user_assignments
        FOR ALL TO authenticated
        USING (
            (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))
```

```
            AND
            (reader_id IN (
                SELECT id FROM readers
                WHERE warehouse_id IN (
                    SELECT id FROM warehouses
                    WHERE manager_user_id = auth.uid()
                )
            ))
        );

    CREATE POLICY "Users can view their reader assignments" ON
        ↪ reader_user_assignments
        FOR SELECT TO authenticated
        USING (user_id = auth.uid());

    -- Tags policies
    CREATE POLICY "Admins can do anything with tags" ON tags
        FOR ALL TO authenticated
        USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));

    CREATE POLICY "All authenticated users can view tags" ON tags
        FOR SELECT TO authenticated
        USING (true);

    -- Scans policies
    CREATE POLICY "Admins can do anything with scans" ON scans
        FOR ALL TO authenticated
        USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));

    CREATE POLICY "Managers can view scans in their warehouse" ON scans
        FOR SELECT TO authenticated
        USING (
            (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))
            AND
            (warehouse_id IN (SELECT id FROM warehouses WHERE
                ↪ manager_user_id = auth.uid()))
        );

    CREATE POLICY "Users can view their scans" ON scans
        FOR SELECT TO authenticated
        USING (user_id = auth.uid());

    -- Settings policies
    CREATE POLICY "Admins can do anything with settings" ON settings
        FOR ALL TO authenticated
        USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));

    CREATE POLICY "All authenticated users can view global settings" ON
        ↪ settings
        FOR SELECT TO authenticated
        USING (scope = 'global');

    CREATE POLICY "Managers can view and edit settings for their warehouse"
        ↪ ON settings
        FOR ALL TO authenticated
        USING (
            (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))
            AND
            (
                (scope = 'warehouse' AND scope_id IN (
                    SELECT id FROM warehouses WHERE manager_user_id = auth.
                        ↪ uid()
                ))
                OR
                (scope = 'reader' AND scope_id IN (
                    SELECT id FROM readers
                    WHERE warehouse_id IN (
                        SELECT id FROM warehouses WHERE manager_user_id =
                            ↪ auth.uid()
                    )
                ))
            )
        );
```

6 Firmware Development

6.1 Atmgea32u4 Programming Code (.cpp)

```
    #include <avr/io.h>
    #include <util/delay.h>
    #include <stdio.h>
    #include <string.h>
    #include <stdint.h>
    #include "mySPI.h"
    #include "PN5180.h"
    #include "myUART.h"

    // Define F_CPU if not already defined
    #ifndef F_CPU
    #define F_CPU 16000000UL
    #endif

    // Buffer size for UID string
    #define UID_STRING_SIZE 32

    // PN5180 instance (NSS = PD1/Pin 2, BUSY = PB6/Pin 10, RST = PB4/Pin 8)
    PN5180 pn5180(/* NSS / 2, / BUSY / 10, / RST */ 8);

    // Function prototypes
    int handshake(void);
    int connectWiFi(void);
    void uart_transmit(const char *data); // For debug output
    void uart_transmit_num(int16_t num);  // For number output

    int main(void) {
        // Initialize UART (USART1)
        USART_Init();

        // Initialize SPI
        SPI_init();
        SPI_setConfig();

        // Initialize PN5180
        pn5180.begin();

        // Configure Pin 12 (PD6) as input with pull-up, Pin 6 (PD7) as
            ↪ output
        DDRD &= ~(1 << DDD6);  // PD6 as input (button)
        DDRD |= (1 << DDD7);   // PD7 as output (LED)
        PORTD |= (1 << PORTD6); // Enable pull-up on PD6
        PORTD &= ~(1 << PORTD7); // LED off initially

        // Handshake with ESP
        for (uint8_t retries = 3; retries > 0; retries--) {
                USART_TransmitCommand("PING");
                int response = handshake();
                if (response == 3) break; // OK received
                _delay_ms(500);
        }

        // Connect to WiFi
        int wifiStatus = connectWiFi();
        char wifiMsg[32];
        snprintf(wifiMsg, sizeof(wifiMsg), "WiFi Status: %d\r\n", wifiStatus
            ↪ );
        uart_transmit(wifiMsg);

        // Main loop
        while (1) {
                if (!(PIND & (1 << PIND6))) { // Button pressed (PD6 low)
                        _delay_ms(50); // Debounce
                        if (!(PIND & (1 << PIND6))) { // Confirm button
                            ↪ press
                                uint8_t uid[8];
                                if (pn5180.getInventoryFake(uid)) {
                                        char uidStr[17];
                                        for (int i = 7, j = 0; i >= 0; i--,
                                            ↪ j += 2) {
                                                sprintf(uidStr + j, "%02X",
                                                    ↪ uid[i]);
                                        }
                                        char cmd[32];
                                        snprintf(cmd, sizeof(cmd), "SEND:UID
                                            ↪ :%s", uidStr);
                                        USART_TransmitCommand(cmd); // Send
                                            ↪ UID to NodeMCU
                                        int response = USART_ReceiveCommand
                                            ↪ ();
                                        char responseMsg[16];
                                        snprintf(responseMsg, sizeof(
                                            ↪ responseMsg), "Response: %d\r\
                                            ↪ n", response);
                                        uart_transmit(responseMsg);
                                        if (response == 3) {
                                                PORTD |= (1 << PORTD7); //
                                                    ↪ LED on
                                                _delay_ms(500);
                                                PORTD &= ~(1 << PORTD7); //
                                                    ↪ LED off
                                        } else if (response == 2) {
                                                // Handle ERROR (add if
                                                    ↪ needed)
                                        }
                                }
```

```c
                    while (!(PIND & (1 << PIND6))) {
                        _delay_ms(10); // Wait for button release
                    }
                }
            }
        }

        return 0; // Never reached
    }

    // UART transmit function for strings
    void uart_transmit(const char *data) {
        while (*data) {
            while (!(UCSR1A & (1 << UDRE1))); // Wait for empty transmit buffer
            UDR1 = *data++; // Send character
        }
    }

    // UART transmit function for numbers
    void uart_transmit_num(int16_t num) {
        char buffer[16];
        snprintf(buffer, sizeof(buffer), "%d\r\n", num);
        uart_transmit(buffer);
    }

    // Handshake with ESP
    int handshake(void) {
        _delay_ms(500);
        USART_TransmitCommand("PING");
        uart_transmit("Waiting for ESP response...\r\n");

        // Wait for response with timeout (~3 seconds)
        char response[MAX_BUFFER_SIZE] = {0};
        for (uint32_t i = 0; i < 3000000; i++) { // Approx 3 seconds at 16 MHz
            USART_ReadString(response, MAX_BUFFER_SIZE, 1000); // Short timeout per read
            if (strlen(response) > 0) {
                char debugMsg[128];
                snprintf(debugMsg, sizeof(debugMsg), "Received from ESP: %s\r\n", response);
                uart_transmit(debugMsg);
                if (strcmp(response, "OK") == 0) {
                    uart_transmit("ESP communication successful.\r\n");
                    return 3;
                    } else {
                    uart_transmit("ESP responded incorrectly.\r\n");
                    return -1;
                }
            }
            _delay_us(1); // Small delay
        }

        uart_transmit("Timeout waiting for ESP.\r\n");
        return -1;
```

```c
}

// Connect to WiFi
int connectWiFi(void) {
    const uint32_t maxTimeout = 30000000; // ~30 seconds at 16 MHz
    const uint32_t retryInterval = 12000000; // ~12 seconds per attempt
    uint32_t attemptCount = maxTimeout / retryInterval;

    for (uint32_t attempt = 0; attempt < attemptCount; attempt++) {
            USART_TransmitCommand("WIFI:CONNECT");
            char response[MAX_BUFFER_SIZE] = {0};

            // Wait for response
            for (uint32_t i = 0; i < retryInterval; i++) {
                    USART_ReadString(response, MAX_BUFFER_SIZE, 1000);
                        ↪ // Short timeout
                    if (strlen(response) > 0) {
                            if (strcmp(response, "OK") == 0) {
                                    return 3;
                            } else if (strcmp(response, "ERROR")
                                ↪  == 0) {
                                    break;
                            }
                    }
                    _delay_us(1); // Small delay
            }

    }
    return -1;
}
```

6.2 PN5180 Programming Code (.h)

```c
/*
 * PN5180.h
 *
 * Created: 5/22/2025 1:47:50 PM
 *  Author: AGRA
 */

#ifndef PN5180_H
#define PN5180_H

#include "mySPI.h"

// PN5180 Commands
#define PN5180_WRITE_REGISTER           (0x00)
#define PN5180_WRITE_REGISTER_OR_MASK   (0x01)
#define PN5180_WRITE_REGISTER_AND_MASK  (0x02)
#define PN5180_READ_REGISTER            (0x04)
#define PN5180_WRITE_EEPROM             (0x06)
#define PN5180_READ_EEPROM              (0x07)
#define PN5180_SEND_DATA                (0x09)
#define PN5180_READ_DATA                (0x0A)
#define PN5180_LOAD_RF_CONFIG           (0x11)
#define PN5180_RF_ON                    (0x16)
#define PN5180_RF_OFF                   (0x17)
```

```
// 11.9.1, Table 73 PN5180 Register Address Overview
#define SYSTEM_CONFIG       (0x00)
#define IRQ_ENABLE          (0x01)
#define IRQ_STATUS          (0x02)
#define IRQ_CLEAR           (0x03)
#define RX_STATUS           (0x13)
#define TX_WAIT_CONFIG      (0x17)
#define TX_CONFIG           (0x18)
// 11.9.1, Table 76 IRQ_STATUS Register
#define RX_IRQ_STAT                 (1<<0)  // End of RF reception IRQ
#define TX_IRQ_STAT                 (1<<1)  // End of RF transmission
    ↪ IRQ
#define IDLE_IRQ_STAT               (1<<2)  // Idle IRQ
#define RFOFF_DET_IRQ_STAT          (1<<6)  // RF Field OFF detection
    ↪ IRQ
#define RFON_DET_IRQ_STAT           (1<<7)  // RF Field ON detection IRQ
#define TX_RFOFF_IRQ_STAT           (1<<8)  // RF Field OFF in PCD IRQ
#define TX_RFON_IRQ_STAT            (1<<9)  // RF Field ON in PCD IRQ
// 11.9.1 Table 92 RX_STATUS Register
#define RX_COLL_POS (1<<19) // These bits show the bit position of the
    ↪ first detected collision in a received frame (7 bits)
#define RX_COLLISION_DETECTED (1<<18) // This flag is set to 1, when a
    ↪ collision has occurred
// 11.9.1 Table 97 TX_CONFIG Register
#define TX_DATA_ENABLE (1<<10) // If set to 1, transmission of data is
    ↪ enabled otherwise only symbols are transmitted.

// The PN5180 receive buffer can hold a max of 508 bytes
// But we only need to transfer 2 bytes + 8 bytes per tag = 10 bytes for
    ↪  a single card
#ifndef READ_BUFFER_SIZE
#define READ_BUFFER_SIZE 10
#endif
// Other constants and enums
enum ISO15693ErrorCode {
    EC_NO_CARD = -1,
    ISO15693_EC_OK = 0,
    ISO15693_EC_NOT_SUPPORTED = 0x01,
    ISO15693_EC_NOT_RECOGNIZED = 0x02,
    ISO15693_EC_OPTION_NOT_SUPPORTED = 0x03,
    ISO15693_EC_UNKNOWN_ERROR = 0x0F,
    ISO15693_EC_BLOCK_NOT_AVAILABLE = 0x10,
    ISO15693_EC_BLOCK_ALREADY_LOCKED = 0x11,
    ISO15693_EC_BLOCK_IS_LOCKED = 0x12,
    ISO15693_EC_BLOCK_NOT_PROGRAMMED = 0x13,
    ISO15693_EC_BLOCK_NOT_LOCKED = 0x14,
    ISO15693_EC_CUSTOM_CMD_ERROR = 0xA0
};

class PN5180 {
    public:
    PN5180(uint8_t nssPin, uint8_t busyPin, uint8_t rstPin);
    void begin();
    void hardReset();
    bool getInventory(uint8_t *uid);
    bool getInventoryFake(uint8_t *uid);
    void loadISO15693config();
    bool activateRF();
    bool disableRF();
```

```
        bool checkIdle();
        void clearIRQ();
        void setIdle();
        void activateTransceive();
        void sendInventoryCmd();
        void sendEndOfFrame();
        bool readReceptionBuffer(uint8_t *buffer, int16_t len);
        uint32_t readRegister(uint8_t regAddress);
        bool sendBytes(uint8_t *sendBuffer, size_t sendBufferLen);
        bool readBytes(uint8_t *recvBuffer, size_t recvBufferLen);

        private:
        uint8_t _nss, _busy, _rst;
        // SPISettings _spiSettings;
        uint16_t _commandTimeout = 800;
        bool waitUntilAvailable();
        bool waitUntilBusy();
        void errorHandler(ISO15693ErrorCode errorCode);

        uint8_t* buffer = (uint8_t*)malloc(READ_BUFFER_SIZE);
    };

    #endif
```

6.3 SPI Programming Code

6.3.1 .h

```
#ifndef MYUART_H
#define MYUART_H

#include <stdint.h>

#define RX_BUFFER_SIZE 64
#define MAX_BUFFER_SIZE 100

void USART_Init(void);
void USART_Transmit(uint8_t data);
uint8_t USART_Receive(void);
void USART_TxString(const char *str);
void USART_ReadString(char *buffer, uint8_t maxLen, uint32_t timeout);
int USART_ReceiveCommand(void);
void USART_TransmitCommand(const char *cmd);

#endif
```

6.3.2 .cpp

```
/*
 * mySPI.cpp
 *
 * Created: 5/22/2025 1:46:21 PM
 *  Author: AGRA
 */

#include <avr/io.h>
```

```c
#include <util/delay.h>
#include <stdio.h>
#include <stdint.h>
#include "mySPI.h"

// Define F_CPU if not already defined
#ifndef F_CPU
#define F_CPU 16000000UL
#endif


// Initialize SPI as Master, Mode 0 (CPOL=0, CPHA=0), 125 kHz (F_CPU
    ↪ /128)
void SPI_init(void) {
    // Set MOSI (PB2), SCK (PB1), and NSS (PD1) as outputs; MISO (PB3)
        ↪ as input
    DDRB |= (1 << DDB2) | (1 << DDB1); // MOSI, SCK
    DDRD |= (1 << DDD1);               // NSS
    DDRB &= ~(1 << DDB3);              // MISO as input

    // Configure SPI: Master, Mode 0, 125 kHz
    SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
    SPSR &= ~(1 << SPI2X);
}

// Reapply SPI configuration (for PN5180 compatibility)
void SPI_setConfig(void) {
    // Reapply SPI settings: Master, Mode 0, 125 kHz
    SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
    SPSR &= ~(1 << SPI2X);

}

// Perform bidirectional SPI transfer
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
    }
}

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

// Test SPI communication
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_ms(1);

    // Send and receive 1 byte
```

```
        SPDR = data;
        while (!(SPSR & (1 << SPIF)));
        received = SPDR;

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

6.4 UART Programming Code

6.4.1 .h

```
    #ifndef MYUART_H
    #define MYUART_H

    #include <stdint.h>

    #define RX_BUFFER_SIZE 64
    #define MAX_BUFFER_SIZE 100

    void USART_Init(void);
    void USART_Transmit(uint8_t data);
    uint8_t USART_Receive(void);
    void USART_TxString(const char *str);
    void USART_ReadString(char *buffer, uint8_t maxLen, uint32_t timeout);
    int USART_ReceiveCommand(void);
    void USART_TransmitCommand(const char *cmd);

    #endif
```

6.4.2 .cpp

```
    /*
     * myUART.cpp
     *
     * Created: 5/22/2025 1:44:47 PM
     *  Author: AGRA
     */

    #include <avr/io.h>
    #include <util/delay.h>
    #include <stdio.h>
    #include <string.h>
    #include <stdint.h>
    #include "myUART.h"

    // Define F_CPU if not already defined
    #ifndef F_CPU
    #define F_CPU 16000000UL
    #endif

    // UART buffers
    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;
    static char rxBuffer[MAX_BUFFER_SIZE];
```

```c
static uint8_t index = 0;

// UART transmit function (for debug output, consistent with previous files)
static void uart_transmit(const char *data) {
    while (*data) {
            while (!(UCSR1A & (1 << UDRE1))); // Wait for empty transmit buffer
            UDR1 = *data++; // Send character
    }
}

// Initialize USART1: 9600 baud, 8-bit, 1 stop bit, no parity, asynchronous
void USART_Init(void) {
    // Set baud rate to 9600 (16 MHz -> UBRR1 = 103)
    UBRR1 = 103;

    // Enable transmitter and receiver
    UCSR1B = (1 << RXEN1) | (1 << TXEN1);

    // Set frame format: 8-bit data, 1 stop bit, no parity, asynchronous
    UCSR1C = (1 << UCSZ11) | (1 << UCSZ10);
    UCSR1B &= ~(1 << UCSZ12); // Ensure 8-bit data
    UCSR1C &= ~((1 << UMSEL11) | (1 << UMSEL10) | (1 << UPM11) | (1 << UPM10) | (1 << USBS1));

    // Normal speed (disable double speed)
    UCSR1A &= ~(1 << U2X1);

    // Debug output
    char msg[] = "USART1 initialized\r\n";
    uart_transmit(msg);
}

// Transmit one byte
void USART_Transmit(uint8_t data) {
    while (!(UCSR1A & (1 << UDRE1))); // Wait for empty transmit buffer
    UDR1 = data;
}

// Receive one byte
uint8_t USART_Receive(void) {
    while (!(UCSR1A & (1 << RXC1))); // Wait for data
    return UDR1;
}

// Transmit a null-terminated string
void USART_TxString(const char *str) {
    while (*str) {
            USART_Transmit(*str++);
    }
}

// Receive a string into buffer (non-blocking, with timeout)
void USART_ReadString(char *buffer, uint8_t maxLen, uint32_t timeout) {
    uint8_t index = 0;
    buffer[0] = '\0'; // Initialize buffer

    // Timeout loop (~10 ms at 16 MHz, adjusted for cycles)
    while (timeout > 0) {
            if (UCSR1A & (1 << RXC1)) {
                    char c = USART_Receive();
                    if (c == '\n') {
                            buffer[index] = '\0'; // Null-terminate
                            return;
                            } else if (index < maxLen - 1) {
                            buffer[index++] = c;
                            } else {
                            buffer[0] = '\0'; // Overflow, return empty string
                            return;
                    }
            }
            _delay_us(1); // ~1  s  delay per iteration
            timeout--;
    }
    buffer[0] = '\0'; // Timeout, return empty string
}

// Receive and parse command
int USART_ReceiveCommand(void) {
    char response[MAX_BUFFER_SIZE];
    USART_ReadString(response, MAX_BUFFER_SIZE, 10000); // ~10 ms timeout

    // Debug output
    char debugMsg[128];
    snprintf(debugMsg, sizeof(debugMsg), "Received: %s\r\n", response);
    uart_transmit(debugMsg);

    if (strlen(response) == 0) {
            return 0; // No complete message
    }

    // Compare response
    if (strcmp(response, "OK") == 0) {
            return 1; // Success
            } else if (strcmp(response, "ERROR") == 0) {
            return 2; // Failure
            } else if (strncmp(response, "DA", 2) == 0) {
            return 3; // Data received
            } else {
            return -1; // Unknown response
    }
}

// Transmit command with "CMD:" prefix and '\n'
void USART_TransmitCommand(const char *cmd) {
    char buffer[MAX_BUFFER_SIZE] = {0};
    uint8_t len = strlen(cmd);

    // Add "CMD:" prefix and '\n' if needed
    if (len > 0 && cmd[len - 1] != '\n') {
            snprintf(buffer, MAX_BUFFER_SIZE - 1, "CMD:%s\n", cmd);
            } else {
            snprintf(buffer, MAX_BUFFER_SIZE - 1, "CMD:%s", cmd);
    }
```

```

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

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