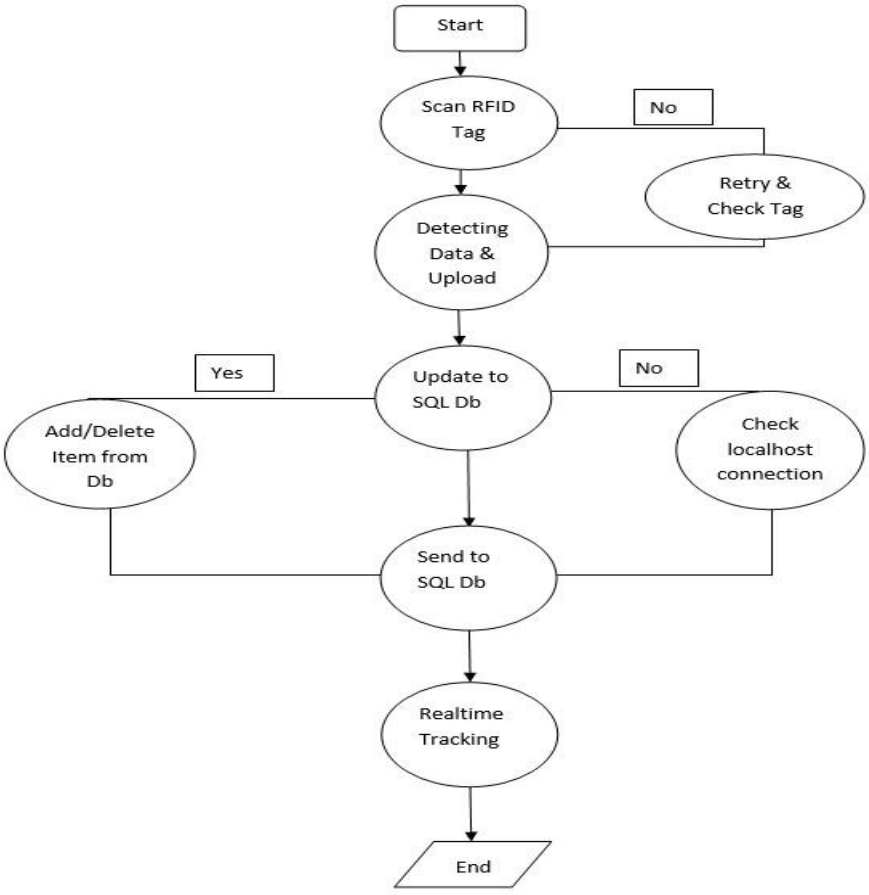
Implementation of IoT-Based Smart Inventory Management System

With the help of the ESP32 Board, we have connected the RFID Reader. Three LEDs are used to show the status of the inventory system. One LED shows the connection status and two shows the material in and out. Buzzers are used to get the alert of material movement. The Breadboard is used to connect all the components inside the box. Then we have run the code and upload the live data on the AWS cloud and show the real time status of inventory system. A dynamic page shows the live data and storage with the help of HTML and CSS.

1 Flow Chart



2 Setup code

```
void setup() {
  Serial.begin(115200);
  SPI.begin(); // Init SPI bus
  rfid.PCD_Init(); // Init MFRC522

  for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
  }
  //**********Connecting To Wifi***********
  Serial.println("Configuring access point...");
  WiFi.softAP(ssid, password);
  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);
  server.begin();
  Serial.println("Server started");

  Serial.println(F("PLEASE PLACE THE CARD INFRONT OF THE READER TO SCAN"));
  Serial.println();

  Serial.println(F("This code scan the MIFARE Classsic NUID."));
  Serial.print(F("Using the following key:"));
  printHex(key.keyByte, MFRC522::MF_KEY_SIZE);
}
```

3 RFID Reading Code

```
void loop() {

  // Reset the loop if no new card present on the sensor/reader. This saves the entire process when idle.
  if ( ! rfid.PICC_IsNewCardPresent())
    return;

  // Verify if the NUID has been readed
  if ( ! rfid.PICC_ReadCardSerial())
    return;

  Serial.print(F("PICC type: "));
  MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
  Serial.println(rfid.PICC_GetTypeName(piccType));

  // Check is the PICC of Classic MIFARE type
  if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
    piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
    piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
    Serial.println(F("Your tag is not of type MIFARE Classic."));
    return;
  }
```

4 RFID Tag Code

```
    if (rfid.uid.uidByte[0] != nuidPICC[0] ||
    rfid.uid.uidByte[1] != nuidPICC[1] ||
    rfid.uid.uidByte[2] != nuidPICC[2] ||
    rfid.uid.uidByte[3] != nuidPICC[3] ||
    rfid.uid.uidByte[4] != nuidPICC[4] )
    {
        Serial.println(F("Stock In"));

        // Store NUID into nuidPICC array
        for (byte i = 0; i < 5; i++) {
        nuidPICC[i] = rfid.uid.uidByte[i];
        }

        Serial.println(F("The NUID tag is:"));
        Serial.print(F("In hex: "));
        printHex(rfid.uid.uidByte, rfid.uid.size);
        Serial.println();
        Serial.print(F("In dec: "));
        printDec(rfid.uid.uidByte, rfid.uid.size);
        Serial.println();
  }
  else Serial.println(F("Card read previously."));
```

5 Defining Macros

```
#include <WiFiClientSecure.h>
#include "secrets.h"
#include "ThingSpeak.h" // always include thingspeak header file after other header files and custom macros

char ssid[] = SECRET_SSID;   // your network SSID (name)
char pass[] = SECRET_PASS;   // your network password
int keyIndex = 0;            // your network key Index number (needed only for WEP)

WiFiClientSecure  client;

unsigned long myChannelNumber = SECRET_CH_ID;
const char * myWriteAPIKey = SECRET_WRITE_APIKEY;

// Initialize our values
int number1 = nuidPICC[0];
int number2 = nuidPICC[1];
int number3 = nuidPICC[2];
int number4 = nuidPICC[3];
int number5 = nuidPICC[4];
String myStatus = "";
```

6 Setting ThingSpeak Field

```
// set the fields with the values
ThingSpeak.setField(1, number1);
ThingSpeak.setField(2, number2);
ThingSpeak.setField(3, number3);
ThingSpeak.setField(4, number4);
ThingSpeak.setField(5, number5);
```

7 Writing ThingSpeak Channel

```
// write to the ThingSpeak channel
int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
if(x == 200){
  Serial.println("Channel update successful.");
}
else{
  Serial.println("Problem updating channel. HTTP error code " + String(x));
}
```