INVENTORY MANAGEMENT SYSTEM WITH REAL-TIME STOCK REDUCTION AND ALERTS

MINI PROJECT REPORT

Submitted by

MANIKANDAM M - 2116230701174

ELUMALAI B - 2116230701084

In partial fulfillment for the award of the degree

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2024

BONAFIDE CERTIFICATE

Certified that this project "INVENTORY MANAGEMENT SYSTEM WITH REAL-TIME STOCK REDUCTION AND ALERTS " **is the bonafide work of "MANIKANDA M (2116230701174) and ELUMALAI B (2116230701084)" who carried out the project work under my supervision.**

SIGNATURE
Dr.N.Duraimurugan, M.Tech., Ph.D.

Associate Professor,

Computer Science & Engineering

Rajalakshmi Engineering College (Autonomous)

Thandalam, Chennai -602105.

Submitted for the **ANNA UNIVERSITY** practical examination Mini-Project work viva voice held on______________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincerethanks to our Chairman **Mr. S.MEGANATHAN, B.E, F.I.E.,** our Vice Chairman**Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and our respectedChairperson **Dr. (Mrs.) THANGAM MEGANATHAN, Ph.D.,** for providing us withthe requisite infrastructure and sincere endeavoring in educating us in their premierinstitution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principalfor his kind support and facilities provided to complete our work in time. We express oursincere thanks to **Dr. P. KUMAR, M.E., Ph.D.,** Professor and Head of the Department ofComputer Science and Engineering for his guidance and encouragement throughout theproject work.

We also extend our sincere and hearty thanks to our Internal Guide **Dr.N.Duraimurugan,M.Tech., Ph.D.**Associate Professor, Department of Computer Science and Engineering for his valuable guidance and motivation during the completion of this project. Our sincere thanks to our family members, friends and other staff members of information technology.

MANIKANDAN M 230701174
ELUMALAI B 230701084

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATION

ABBREVIATION	**ACRONYM**
IOT	Internet of Things
HTTP	HyperText Transfer Protocol
API	Application Programming Interface
SMS	Short Message Service
UIID	Unique Inventory Item ID

ABSTRACT

The Inventory Management System (IMS) presented in this paper is designed to automate the management of stock in a retail environment. The system utilizes real-time stock reduction based on product sales, barcode scanning for easy product identification, and notifications to alert administrators about low stock levels. This web-based system integrates with a database to track product quantities, update stock in real-time, and generate billing information for customers. The primary goal is to improve operational efficiency, reduce errors in inventory tracking, and enhance communication regarding stock levels through automated alerts.

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Inventory management is a crucial aspect of retail businesses, ensuring that the right amount of stock is available to meet customer demands while minimizing overstock and stockouts. Traditional manual inventory management is prone to human errors, inefficient stock updates, and delayed notifications. This paper proposes a modern, automated inventory management system (IMS) designed to reduce stock in real-time using barcode scanning and provide instant alerts via email when stock levels reach critical thresholds. The system integrates seamlessly with backend databases to manage products and track sales, enhancing both inventory accuracy and business efficiency.

1.2 SCOPE OF THE WORK

This system is ideal for small-scale warehouses, retail shops, and pharmacies. It eliminates manual entries, reduces human error, and provides real-time visibility into inventory. This scalable solution can be integrated with existing inventory management software or ERP systems..

1.3 PROBLEM STATEMENT

Traditional inventory management systems often suffer from inaccuracies, delayed stock updates, and lack of timely notifications, leading to overstocking, stockouts, and missed sales opportunities. This paper proposes a real-time Inventory Management System (IMS) that integrates barcode scanning, RFID billing, sensor-based stock monitoring, and email alerts to address these issues efficiently.

1.4 AIM AND OBJECTIVES OF THE PROJECT

To develop a real-time inventory monitoring system. To implement automatic stock reduction using RFID and ultrasonic sensors. To notify administrators of low stock levels via buzzer and email alerts.To integrate a responsive web app for real-time inventory control and billing. To enhance overall efficiency and reduce human error in inventory tracking.

CHAPTER 2

SYSTEM SPECIFICATIONS

2.1IOT DEVICES

1.Arduino UNO
2.RFID Reader
3.RFID Tags
4.Ultrasonic sensors
5.Buzzer
6.Power Supply
7.Jumper wired

2.2 SOFTWARE SPECIFICATIONS

Operating System	Windows 11
Front – End	React JS
Back – End	Node js , Supabase
Browser	Google Chrome
IDE	Arduino IDE

CHAPTER 3
SYSTEM DESIGN

3.1 ARCHITECTURE DIAGRAM

An architecture diagram is a graphical representation of a set of concepts, that are part of an architecture, including their principles, elements and components.

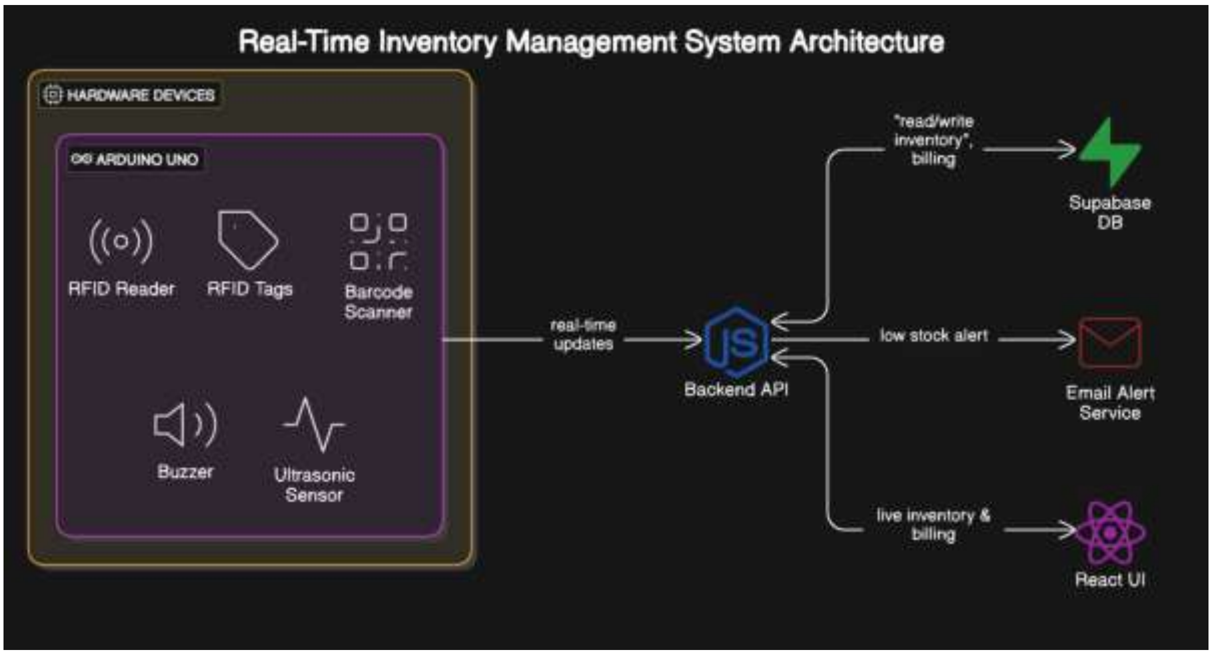


Figure 3.1 Architecture Diagram

From the above Figure 3.1, the architecture of the system is well understood.

3.2 USE CASE DIAGRAM

A use case is a list of actions or event steps typically defining the interactions between a role (known in the Unified Modelling Language as an actor) and a system to achieve a goal. The actor can be a human or other external system.

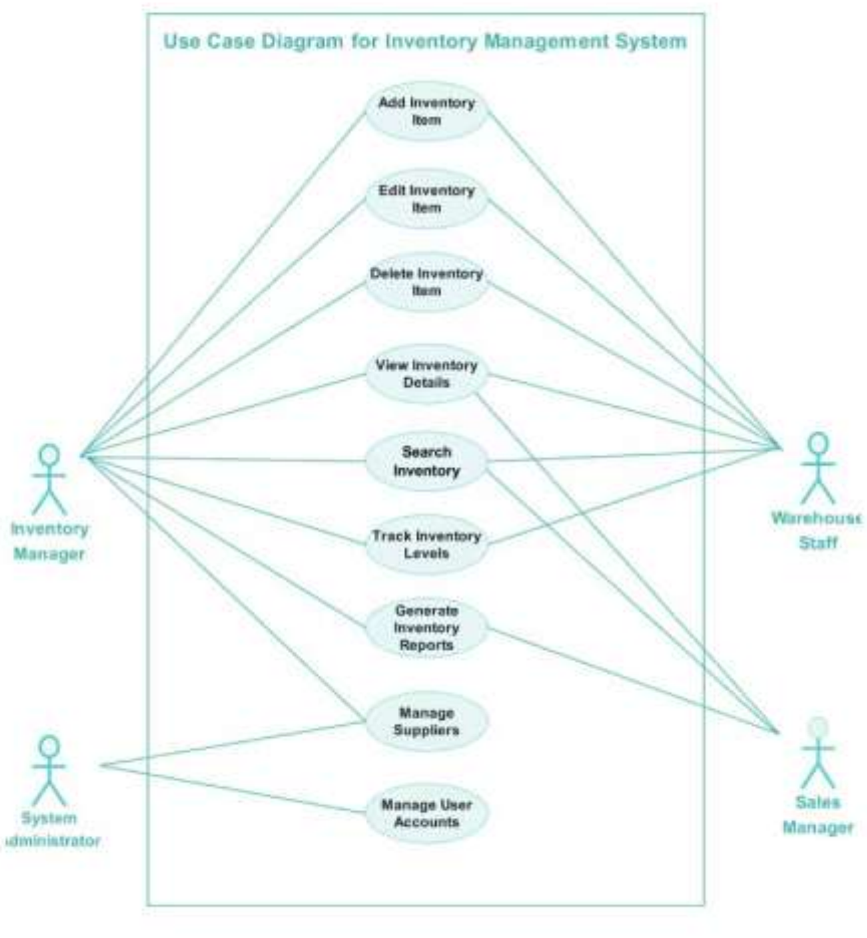
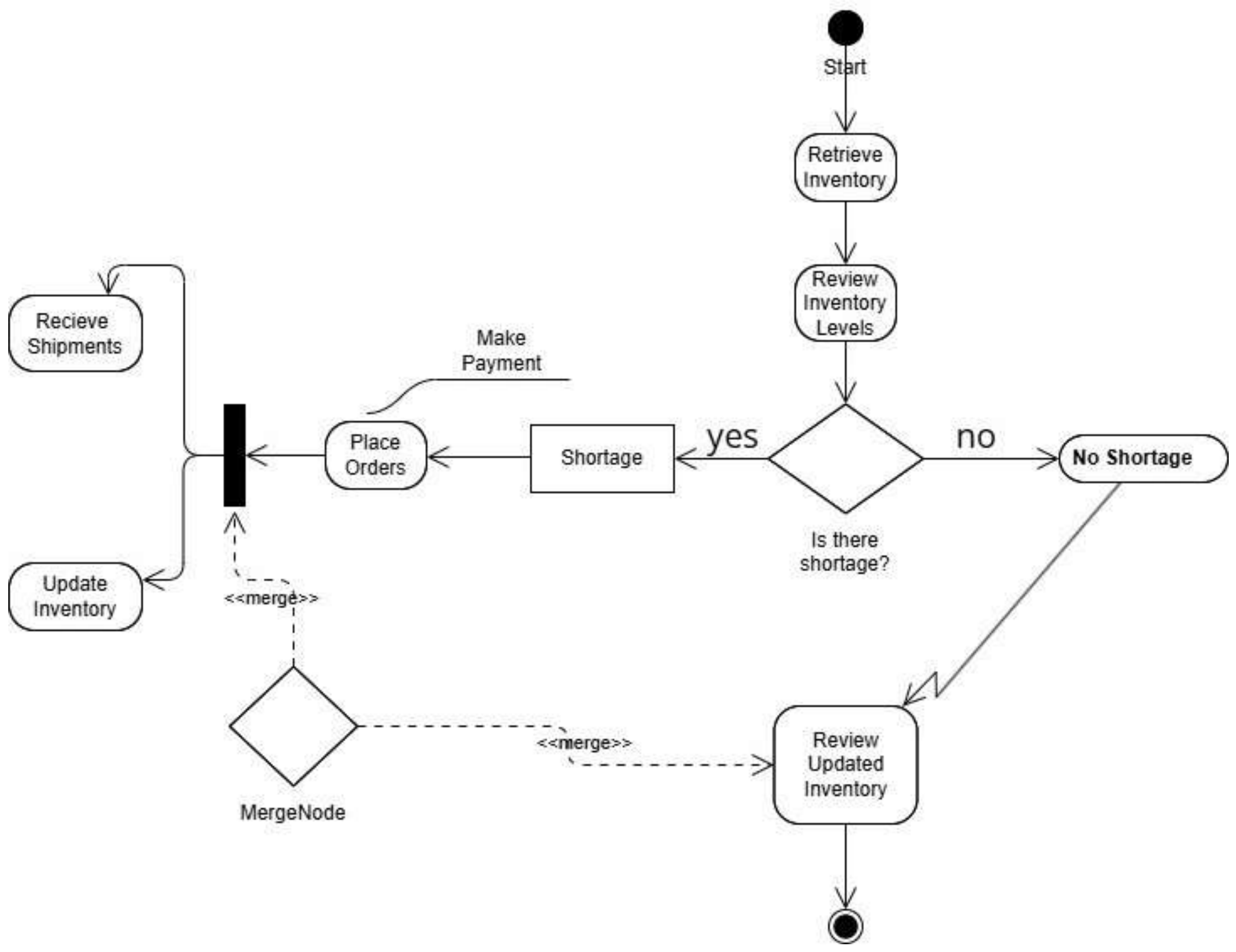


Figure 3.2 Use case diagram

From the above figure 3.2, the interactions between a role in the system is shown

3.3 ACTIVITY DIAGRAM

An activity in Unified Modelling Language (UML) is a major task that must take place in order to fulfill an operation contract. Activities can be represented inactivity diagrams. An activity can represent: The invocation of an operation. A step in a business process.



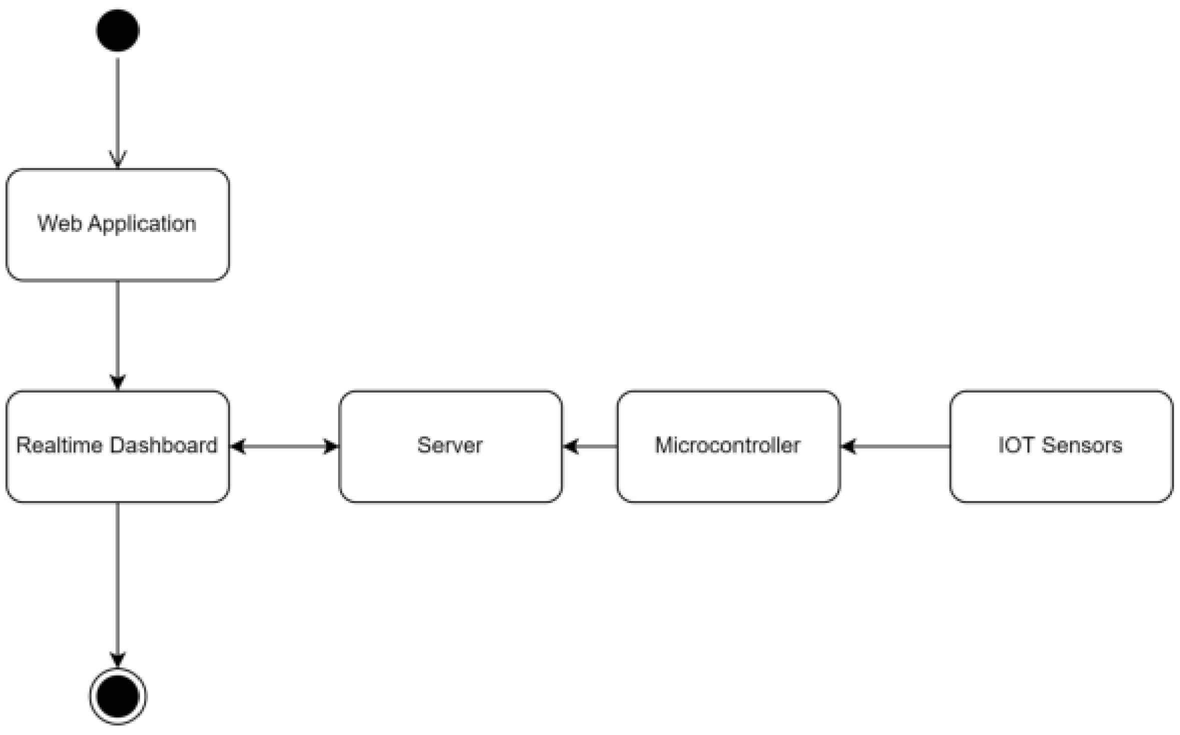


Figure 3.3 Activity Diagram

From the above figure 3.3, the activities of the system are shown

3.4 CLASS DIAGRAM

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modelling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity.

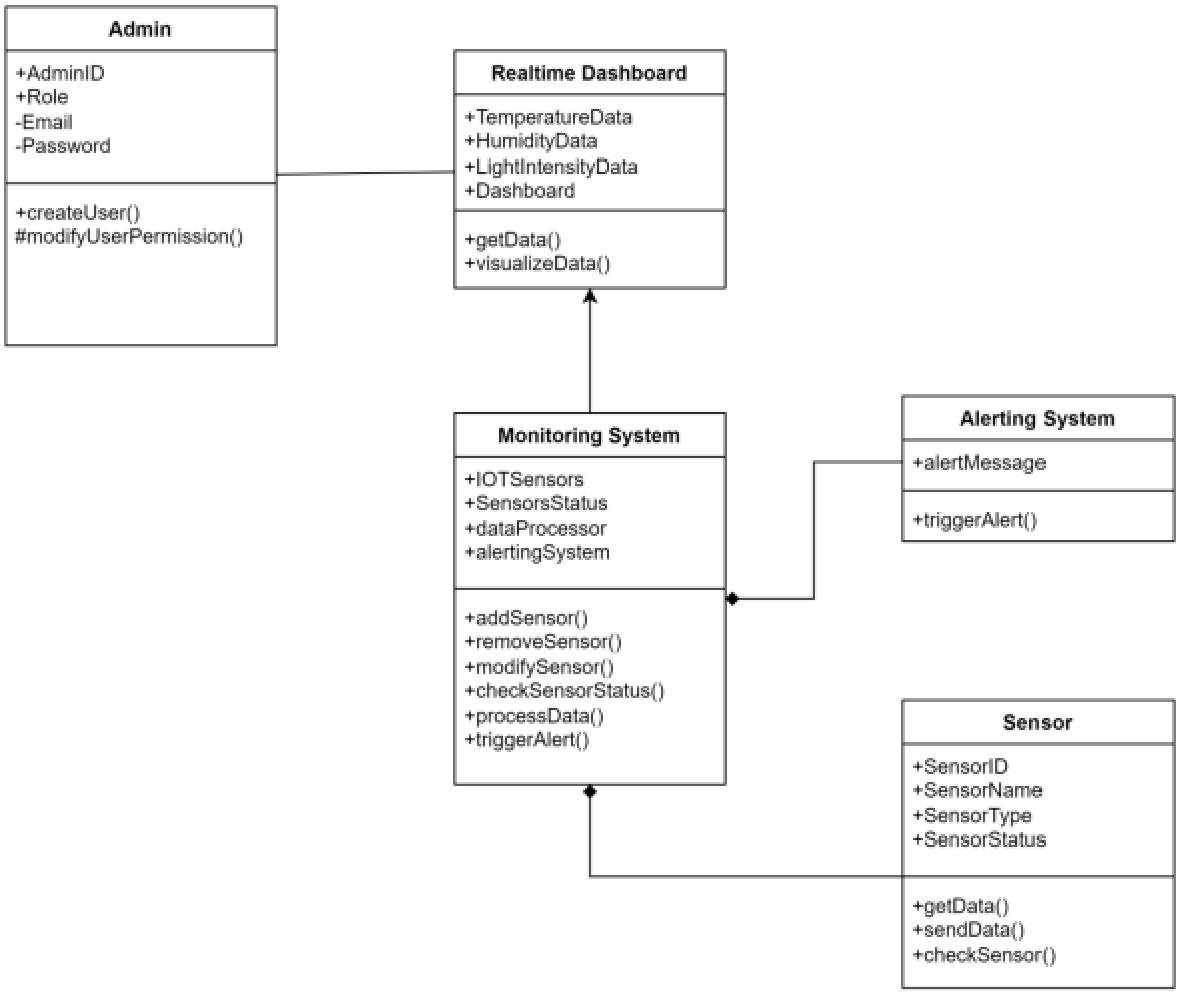


Figure 3.4 Class Diagram

The above Figure 3.4 is the class diagram for the system.

CHAPTER 4

MODULE DESCRIPTION

4.1 HARDWARE MODULE:

This module consists of smart hardware components such as barcode scanners, RFID readers, or weight sensors connected to microcontrollers like NodeMCU (ESP8266) or Raspberry Pi. These IoT devices are responsible for capturing product data such as unique IDs or quantity changes. The hardware transmits this information wirelessly to the cloud using Wi-Fi, allowing for seamless, real-time updates without manual input. The use of IoT in the hardware module ensures automatic data collection and reduces dependency on human intervention.

4.2 DATA COLLECTION AND PROCESSING MODULE:

Once data is received from the IoT hardware, it is processed instantly in the cloud. This module verifies product entries against the database and updates stock quantities accordingly. Using cloud platforms such as Firebase or MQTT brokers, the module ensures fast and reliable communication between the hardware and backend. It maintains logs of inventory changes and supports real-time decision-making for accurate inventory tracking.

4.3 ALERTING MODULE:

The alerting module monitors stock levels continuously. When the quantity of a product falls below a specified threshold, an automatic alert is generated and sent via email, SMS, or push notifications. These alerts are triggered using IoT logic, eliminating the need for manual checks. This proactive system ensures that administrators are instantly notified about low stock conditions, enabling timely restocking and avoiding product shortages.

4.4 WEB APPLICATION MODULE:

The web application module acts as the control center for managing the entire IoT-enabled system. It displays real-time stock levels, device status, and alert messages on a user-friendly dashboard. The interface allows admins to add or update product details, monitor sensor data, and remotely manage devices. Through cloud synchronization, the web app ensures that all inventory and device data is always up to date and accessible from anywhere.

4.5 INTEGRATION MODULE:

This module facilitates the communication between IoT devices and the backend database. It uses lightweight protocols like HTTP or MQTT to handle data transmission securely and efficiently. It ensures that all modules—hardware, data processing, alerting, and web interface—work cohesively as a unified system. The integration module is responsible for managing authentication, data routing, and system scalability, providing a solid backbone for the entire IoT-enabled Inventory Management System.

CHAPTER 5

TABLE

5.1 MEDICINE TABLE

S.NO	ATTRIBUTE	TYPE
1	InventoryID	NUMBER(5)
2	ProductID	VARCHAR(45)
3	ProductName	VARCHAR(45)
4	Category	NUMBER(5,2)

5	Currentstock	NUMBER(5,2)
6	Reorderlevel	NUMBER(7,2)
7	lastUpdate	NUMBER(5,2)
8	Warehouse	DATE
9	Locations	VARCHAR(50)

5.2 STORAGE TABLE

S.NO	**ATTRIBUTE**	**TYPE**
1.	STORAGE_ID	NUMBER(5)
2.	InventoryID	NUMBER(5)
3.	ProductID	DATE
4.	EXPIRY_DATE	DATE

5.3 HISTORY TABLE

S.NO	ATTRIBUTE	TYPE
1.	HISTORY_ID	NUMBER(5)
2.	STORAGE_ID	INTEGER
3.	InventoryID	NUMBER(5,2)
4.	ProductID	NUMBER(5,2)
5.	lastUpdate	NUMBER(7,2)
6.	Category	NUMBER(5,2)

5.4 CURRENT DATA TABLE

S.NO	ATTRIBUTE	TYPE
1.	ProductID	NUMBER(5,2)
2.	ProductName	NUMBER(5,2)
3.	UpdatedBY	NUMBER(7,2)
4.	StockStatus	NUMBER(5,2)

CHAPTER 6

SAMPLE CODING

ARDUINO Program

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>
#include <SPI.h>
#include <MFRC522.h>

#define RST_PIN D3
#define SS_PIN D4
#define TRIG_PIN D5
#define ECHO_PIN D6
#define BUZZER_PIN D7

const char* ssid = "YOUR_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

const char* supabaseUrl = "https://your-project.supabase.co/rest/v1";
const char* supabaseKey =
"YOUR_SUPABASE_ANON_OR_SERVICE_ROLE_KEY";

// Your table names
const char* inventoryTable = "inventory";
const char* rfidLogsTable = "rfid_logs";

MFRC522 mfrc522(SS_PIN, RST_PIN);

int inventory = 5;
const int maxStock = 5;
bool alerted = false;
bool restocking = false;

long distance;
unsigned long lastPenTime = 0;
bool firstPenTaken = false;

WiFiClientSecure client; // for HTTPS requests
```

```
void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");

  // Setup pins
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);

  SPI.begin();
  mfrc522.PCD_Init();

  client.setInsecure(); // Disable certificate verification (not secure for prod!)

  uploadInventory(inventory); // Initial upload
}

void loop() {
  // Ultrasonic sensor reading
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  distance = pulseIn(ECHO_PIN, HIGH) * 0.034 / 2;

  if (distance < 20 && !restocking) {
    unsigned long currentMillis = millis();

    if (!firstPenTaken) {
      delay(4000);
      inventory--;
      Serial.println("Pen taken");
      firstPenTaken = true;
      lastPenTime = currentMillis;
      uploadInventory(inventory);
    } else if (currentMillis - lastPenTime >= 5000) {
      if (inventory > 0) {
        inventory--;
        Serial.println("Pen taken");
        lastPenTime = currentMillis;
        uploadInventory(inventory);
      }
    }

    if (inventory == 1 && !alerted) {
      beepBuzzer(5000);
      alerted = true;
    }

    if (inventory == 0) {
      beepBuzzer(5000);
      restocking = true;
      alerted = false;
    }
    delay(1000);
  }

  if (restocking) {
    inventory = maxStock;
    Serial.println("Restocked");
    uploadInventory(inventory);
    restocking = false;
    firstPenTaken = false;
  }

  checkRFID();
}

void checkRFID() {
  if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial())
{
    String uid = "";
    for (byte i = 0; i < mfrc522.uid.size; i++) {
      if (mfrc522.uid.uidByte[i] < 0x10) uid += "0"; // pad leading 0
      uid += String(mfrc522.uid.uidByte[i], HEX);
    }
    uid.toUpperCase();
    Serial.print("RFID UID: ");
    Serial.println(uid);

    logRFID(uid);
    mfrc522.PICC_HaltA();
  }
}

void uploadInventory(int count) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient https;
    String url = String(supabaseUrl) + "/" + inventoryTable + "?id=eq.1";

    // Build JSON body for PATCH (update where id=1)
    String jsonBody = "{\"count\":";
    jsonBody += String(count);
    jsonBody += "}";

    https.begin(client, url);
    https.addHeader("apikey", supabaseKey);
    https.addHeader("Authorization", "Bearer " + String(supabaseKey));
    https.addHeader("Content-Type", "application/json");

    int httpCode = https.PATCH(jsonBody);
    if (httpCode > 0) {
      String payload = https.getString();
      Serial.print("Inventory updated: ");
      Serial.println(payload);
    } else {
      Serial.print("Error updating inventory: ");
      Serial.println(httpCode);
    }
    https.end();
  }
}
```

```
void logRFID(String uid) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient https;
    String url = String(supabaseUrl) + "/" + rfidLogsTable;

    // Build JSON body for POST (insert)
    String jsonBody = "{\"uid\":\"" + uid + "\", \"timestamp\": \"now()\"}";

    https.begin(client, url);
    https.addHeader("apikey", supabaseKey);
    https.addHeader("Authorization", "Bearer " + String(supabaseKey));
    https.addHeader("Content-Type", "application/json");

    int httpCode = https.POST(jsonBody);
    if (httpCode > 0) {
      String payload = https.getString();
      Serial.print("RFID logged: ");
      Serial.println(payload);
    } else {
      Serial.print("Error logging RFID: ");
      Serial.println(httpCode);
    }
    https.end();
  }
}

void beepBuzzer(int durationMs) {
  digitalWrite(BUZZER_PIN, HIGH);
  delay(durationMs);
  digitalWrite(BUZZER_PIN, LOW);
}
```

Web Application

1. App.tsx

```
import { Toaster } from "@/components/ui/toaster";

import { Toaster as Sonner } from "@/components/ui/sonner";

import { TooltipProvider } from "@/components/ui/tooltip";

import { QueryClient, QueryClientProvider } from "@tanstack/react-query";

import { BrowserRouter, Routes, Route } from "react-router-dom";

import { InventoryProvider } from "./context/InventoryContext";

import Layout from "./components/layout/Layout";

import Dashboard from "./pages/Dashboard";

import Products from "./pages/Products";

import AddProduct from "./pages/AddProduct";

import EditProduct from "./pages/EditProduct";

import Scanner from "./pages/Scanner";

import Bills from "./pages/Bills";

import BillDetail from "./pages/BillDetail";

import NotFound from "./pages/NotFound";


const queryClient = new QueryClient();


const App = () => (
<QueryClientProvider client={queryClient}>
  <TooltipProvider>
    <InventoryProvider>
      <Toaster />
      <Sonner />
      <BrowserRouter>
        <Layout>
          <Routes>
            <Route path="/" element={<Dashboard />} />
            <Route path="/products" element={<Products />} />
            <Route path="/products/add" element={<AddProduct />} />
            <Route path="/products/edit/:id" element={<EditProduct />} />
            <Route path="/scanner" element={<Scanner />} />
            <Route path="/bills" element={<Bills />} />
            <Route path="/bills/:id" element={<BillDetail />} />
            <Route path="*" element={<NotFound />} />
          </Routes>
        </Layout>
      </BrowserRouter>
    </InventoryProvider>
  </TooltipProvider>
  </QueryClientProvider>

);

export default App;
```

2. Dashboard.jsx

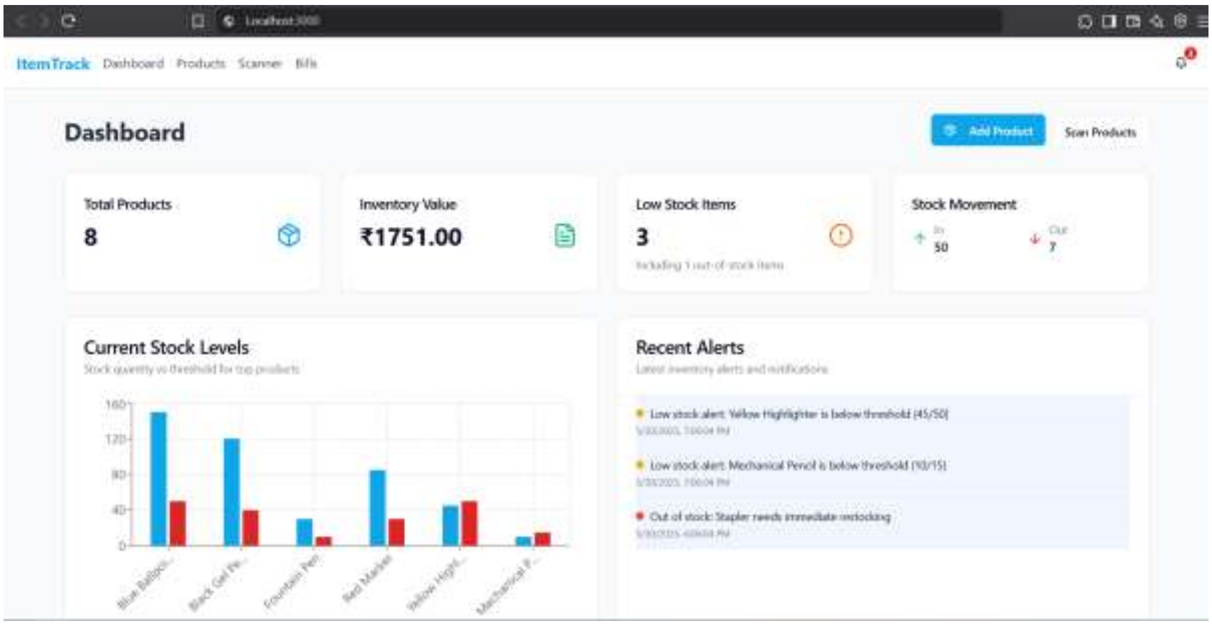
```
import React from 'react';
import { useNavigate } from 'react-router-dom';
import { useInventory } from '@/context/InventoryContext';
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from
'@/components/ui/card';
import { Button } from '@/components/ui/button';
import { BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip,
ResponsiveContainer } from 'recharts';
import { AlertCircle, PackagePlus, FileText, Package, ArrowUp, ArrowDown }
from 'lucide-react';

const Dashboard = () => {
  const { products, alerts, stockMovements, bills } = useInventory();
  const navigate = useNavigate();

  // Calculate stats
  const totalProducts = products.length;
  const totalValue = products.reduce((sum, product) => sum + (product.price *
product.quantity), 0);
  const lowStockCount = products.filter(p => p.quantity <= p.threshold).length;
  const outOfStockCount = products.filter(p => p.quantity === 0).length;
  const recentSales = bills.slice(0, 5);

  // Prepare chart data
  const stockData = products
    .filter(p => p.quantity > 0)
    .slice(0, 6)
    .map(p => ({
      name: p.name.length > 12 ? p.name.substring(0, 12) + '...' : p.name,
      quantity: p.quantity,
      threshold: p.threshold,
    }));

  // Recent stock movement stats
  const incomingStock = stockMovements
    .filter(m => m.type === 'in')
    .reduce((sum, m) => sum + m.quantity, 0);

  const outgoingStock = stockMovements
    .filter(m => m.type === 'out')
    .reduce((sum, m) => sum + m.quantity, 0);

  return (
    <div className="space-y-8 animate-fade-in">
      <div className="flex justify-between items-center">
        <h1 className="text-3xl font-bold text-gray-800">Dashboard</h1>
        <div className="space-x-2">
          <Button onClick={() => navigate('/products/add')} className="bg-
inventory-blue hover:bg-inventory-darkBlue">
            <PackagePlus className="h-4 w-4 mr-2" /> Add Product
          </Button>
          <Button onClick={() => navigate('/scanner')} variant="outline">
            Scan Products
          </Button>
        </div>
      </div>

      {/* Stats Overview */}
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
        <Card className="bg-white">
          <CardHeader className="pb-2">
            <CardTitle className="text-lg">Total Products</CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex justify-between items-center">
              <span className="text-3xl font-bold">{totalProducts}</span>
              <Package className="h-8 w-8 text-inventory-blue" />
            </div>
          </CardContent>
        </Card>

        <Card className="bg-white">
          <CardHeader className="pb-2">
            <CardTitle className="text-lg">Inventory Value</CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex justify-between items-center">
              <span className="text-3xl font-
  bold">₹{totalValue.toFixed(2)}</span>
              <FileText className="h-8 w-8 text-inventory-green" />
            </div>
          </CardContent>
        </Card>

        <Card className="bg-white">
          <CardHeader className="pb-2">
            <CardTitle className="text-lg">Low Stock Items</CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex justify-between items-center">
              <span className="text-3xl font-bold">{lowStockCount}</span>
              <AlertCircle className="h-8 w-8 text-inventory-orange" />
            </div>
            <p className="text-sm mt-2 text-gray-500">
              Including {outOfStockCount} out-of-stock items
            </p>
          </CardContent>
        </Card>

        <Card className="bg-white">
          <CardHeader className="pb-2">
            <CardTitle className="text-lg">Stock Movement</CardTitle>
          </CardHeader>
          <CardContent>
            <div className="grid grid-cols-2 gap-4">
              <div className="flex items-center">
                <ArrowUp className="h-5 w-5 mr-2 text-inventory-green" />
                <div>
                  <p className="text-sm text-gray-500">In</p>
                  <p className="font-semibold">{incomingStock}</p>
                </div>
              </div>
              <div className="flex items-center">
                <ArrowDown className="h-5 w-5 mr-2 text-inventory-red" />
                <div>
                  <p className="text-sm text-gray-500">Out</p>
                  <p className="font-semibold">{outgoingStock}</p>
                </div>
              </div>
            </div>
          </CardContent>
        </Card>
      </div>

      {/* Stock Level Chart */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <Card className="bg-white">
          <CardHeader>
            <CardTitle>Current Stock Levels</CardTitle>
            <CardDescription>Stock quantity vs threshold for top
      products</CardDescription>
          </CardHeader>
          <CardContent>
            <div className="h-[300px]">
              <ResponsiveContainer width="100%" height="100%">
                <BarChart
                  data={stockData}
                  margin={{ top: 10, right: 10, left: 0, bottom: 40 }}
                  barSize={20}
                >
                  <CartesianGrid strokeDasharray="3 3" />
                  <XAxis
                    dataKey="name"
                    angle={-45}
                    textAnchor="end"
                    height={70}
                  />
                  <YAxis />
                  <Tooltip />
                  <Bar dataKey="quantity" fill="#0EA5E9" name="Current Stock" />
                  <Bar dataKey="threshold" fill="#DC2626" name="Threshold" />
                </BarChart>
              </ResponsiveContainer>
            </div>
          </CardContent>
        </Card>

        {/* Recent Alerts */}
        <Card className="bg-white">
          <CardHeader className="border-b">
```

```
          <CardTitle>Recent Alerts</CardTitle>
          <CardDescription>Latest inventory alerts and
notifications</CardDescription>
        </CardHeader>
        <CardContent className="divide-y">
          {alerts.length > 0 ? (
            alerts.slice(0, 5).map((alert) => (
              <div key={alert.id} className={`py-3 ${!alert.read ? 'bg-blue-50' :
''}`}>
                <div className="flex justify-between">
                  <div className="flex items-center">
                    {alert.type === 'low-stock' && (
                      <div className="h-2.5 w-2.5 rounded-full bg-yellow-500 mr-2"
/>
                    )}
                    {alert.type === 'out-of-stock' && (
                      <div className="h-2.5 w-2.5 rounded-full bg-red-500 mr-2" />
                    )}
                    {alert.type === 'restock' && (
                      <div className="h-2.5 w-2.5 rounded-full bg-green-500 mr-2"
/>
                    )}
                    <span className="text-sm">{alert.message}</span>
                  </div>
                </div>
                <p className="text-xs text-gray-500 mt-1">
                  {new Date(alert.timestamp).toLocaleString()}
                </p>
              </div>
            ))
          ) : (
            <div className="py-4 text-center text-gray-500">No recent
alerts</div>
          )}
          {alerts.length > 5 && (
            <div className="pt-3">
              <Button
                variant="ghost"
                className="w-full text-inventory-blue hover:text-inventory-
darkBlue"
                onClick={() => navigate('/alerts')}
              >
                View all alerts
              </Button>
            </div>
          )}
        </CardContent>
      </Card>
    </div>

    {/* Recent Activity */}
    <Card className="bg-white">
      <CardHeader className="border-b">
        <CardTitle>Recent Sales</CardTitle>
        <CardDescription>Latest transactions and bill
generation</CardDescription>
      </CardHeader>
      <CardContent>
        <div className="divide-y">
          {recentSales.length > 0 ? (
            recentSales.map((bill) => (
              <div key={bill.id} className="py-4">
                <div className="flex justify-between items-start">
                  <div>
                    <p className="font-medium">
                      Bill #{bill.id.substring(0, 8)}
                      {bill.customerName && ` - ₹{bill.customerName}`}
                    </p>
                    <p className="text-sm text-gray-500">
                      {new Date(bill.timestamp).toLocaleString()}
                    </p>
                    <p className="text-sm mt-1">
                      {bill.items.length} {bill.items.length === 1 ? 'item' : 'items'}
                    </p>
                  </div>
                  <div className="text-right">
                    <p className="font-bold">₹{bill.total.toFixed(2)}</p>
                    <Button
                      variant="outline"
                      size="sm"
                      className="mt-1"
                      onClick={() => navigate(`/bills/₹{bill.id}`)}
                    >
                      View Bill
                    </Button>
                  </div>
                </div>
              </div>
            ))
          ) : (
            <div className="py-4 text-center text-gray-500">No recent
sales</div>
          )}
        </div>
        {bills.length > 5 && (
          <div className="mt-4">
            <Button
              variant="outline"
              className="w-full"
              onClick={() => navigate('/bills')}
            >
              View All Bills
            </Button>
          </div>
        )}
      </CardContent>
    </Card>
  </div>
);
};

export default Dashboard;
```

CHAPTER 7

SCREEN SHOTS

1. Dashboard Page



Dashboard

Total Products
8

Inventory Value
₹1751.00

Low Stock Items
3
Including 1 out-of-stock items

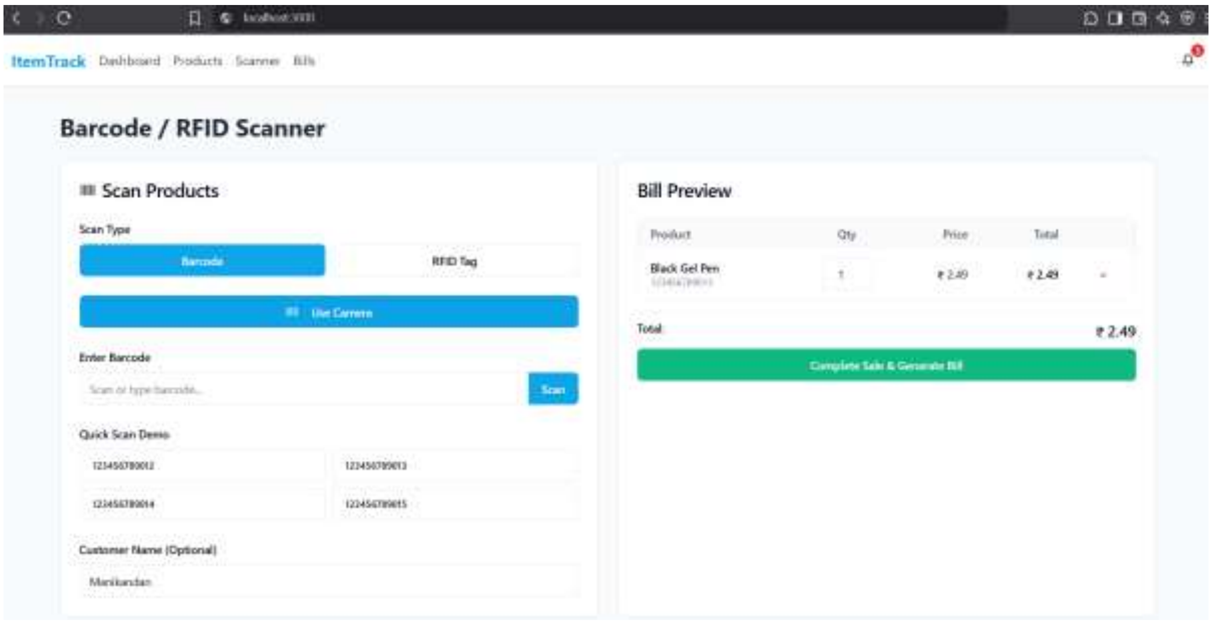
Stock Movement
In 50
Out 7

Current Stock Levels
Stock quantity vs threshold for top products

Recent Alerts
Latest inventory alerts and notifications

- Low stock alert: Yellow Highlighter is below threshold (45/50)
- Low stock alert: Mechanical Pencil is below threshold (10/15)
- Out of stock: Stapler needs immediate restocking

Figure 7.1 Responsive Dashboard



Barcode / RFID Scanner

Scan Products

Scan Type

Enter Barcode
Scan or type barcode...

Quick Scan Demo

123456789012	123456789013
123456789014	123456789015

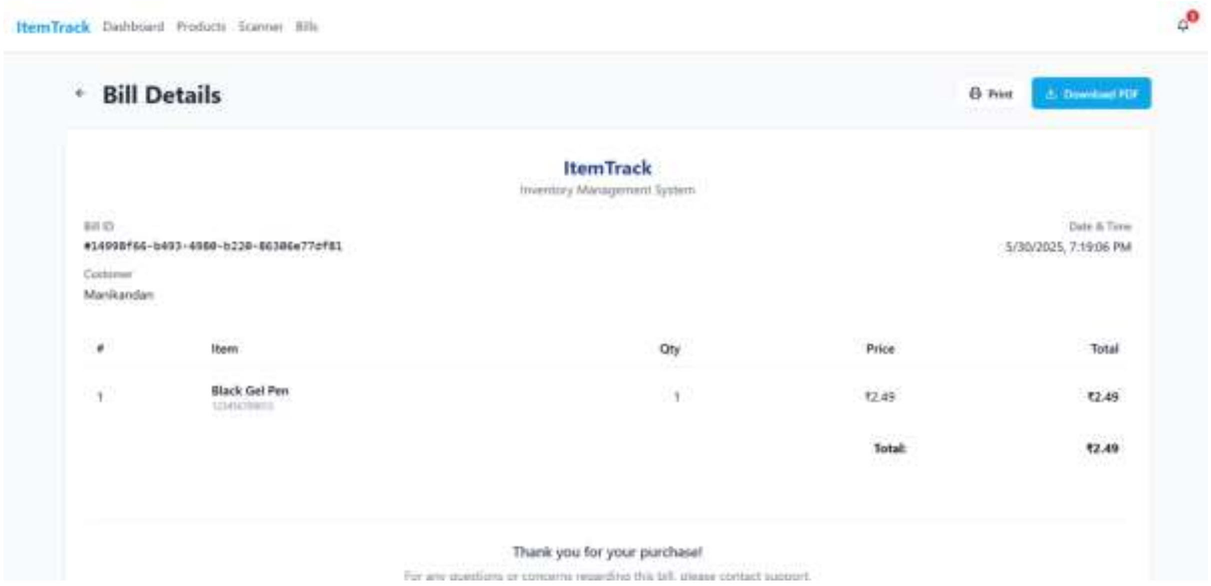
Customer Name (Optional)
Manikandan

Bill Preview

Product	Qty	Price	Total
Black Gel Pen	1	₹ 2.49	₹ 2.49

Total ₹ 2.49

Figure 7.2 RFID scanner



Bill Details

ItemTrack

Inventory Management System

Bill ID

#14998f66-b493-4980-b220-86306e77df81

Date & Time

5/30/2025, 7:19:06 PM

Customer

Manikandan

#	Item	Qty	Price	Total
1	Black Gel Pen [illegible]	1	₹2.49	₹2.49
			Total:	₹2.49

Thank you for your purchase!

For any questions or concerns regarding this bill, please contact support.

Figure 7.3 Bill Details

2.Prototype:

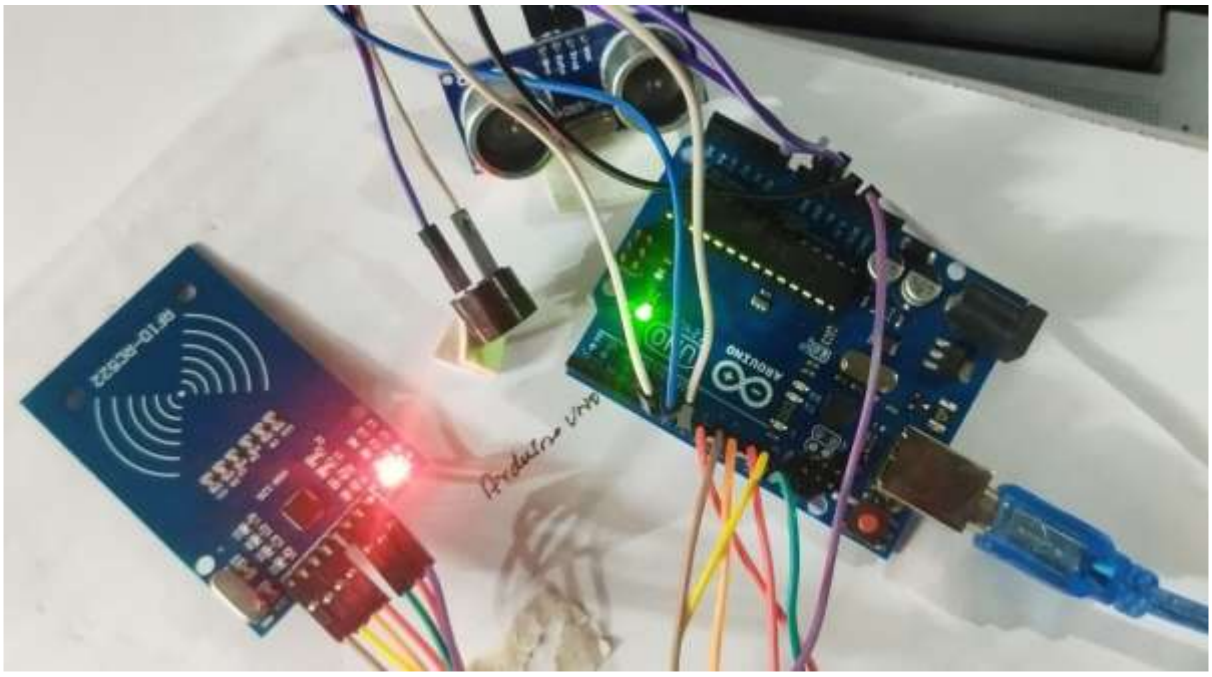
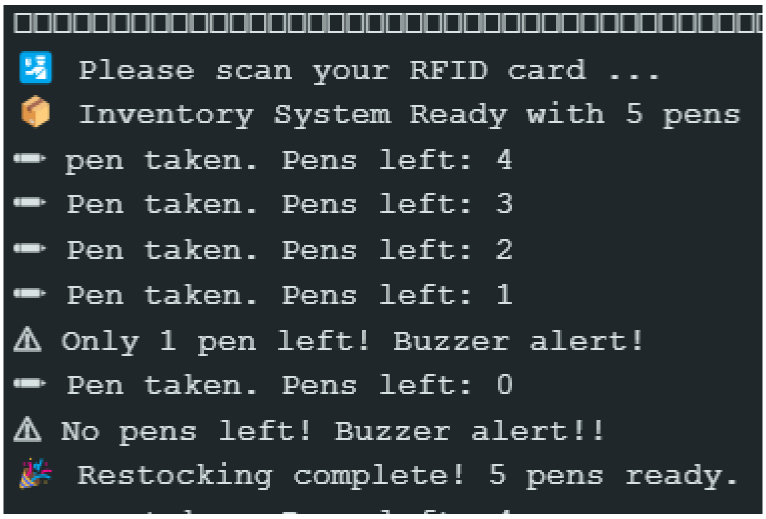


Figure 7.5 Inventory Prototype

3. Data Sent from ARDUINO,ESP8266 - Nodemcu to Server



```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
🛂 Please scan your RFID card ...
📦 Inventory System Ready with 5 pens
🖊 pen taken. Pens left: 4
🖊 Pen taken. Pens left: 3
🖊 Pen taken. Pens left: 2
🖊 Pen taken. Pens left: 1
⚠ Only 1 pen left! Buzzer alert!
🖊 Pen taken. Pens left: 0
⚠ No pens left! Buzzer alert!!
🎉 Restocking complete! 5 pens ready.
```

Figure 7.5 Data Received by Server from ESP8266-supabase

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

. In conclusion, the IoT-based Inventory Management System significantly improves inventory tracking, product identification, and stock management in retail environments. By integrating barcode scanning and Firebase for real-time updates, the system minimizes manual errors, reduces stock discrepancies, and ensures timely low-stock alerts to administrators. This automation leads to enhanced operational efficiency, improved customer satisfaction, and more informed decision-making.

Future Enhancements may include:

- **AI-Powered Forecasting:** Predict future stock needs using machine learning.
- **Voice Assistant Integration:** Allow voice commands for inventory queries and updates.
- **Mobile App Support:** Extend accessibility through Android/iOS apps.
- **Multi-store Synchronization:** Enable centralized monitoring across branches.
- **Blockchain for Security:** Ensure tamper-proof inventory logs using blockchain.

REFERENCES

1. **Wu, Z., Takeda, K., Gupta, P., Zheng, R., Yang, L., Zhang, C., Fan, Z., Xu, H., Mukkavilli, K., & Ji, T. (2025).** Fast Inventory for 3GPP Ambient IoT Considering Device Unavailability due to Energy Harvesting. *arXiv preprint arXiv:2501.15020*.

2. **Tong, C. (2023).** An Efficient Intelligent Semi-Automated Warehouse Inventory Stocktaking System. *arXiv preprint arXiv:2309.12365*.

3. **Karri, A., Rao, S. S., Ashoka, D. V., Nethravathi, B., & Viswanath, S. (2022).** Hardware Inventory Management System Using IoT. In *Proceedings of the 2022 Fourth International Conference on Cognitive Computing and Information Processing (CCIP)* (pp. 1–6).

4. **Gultom, L. R., & Yosephine, V. S. (2024).** IoT-Based Inventory Monitoring System for SMEs. *TEKNOSAINS: Jurnal Sains, Teknologi dan Informatika*, *11*(2), 331–341.

5. **Aher, V. N., Pol, R. S., Gaikwad, S. V., Bhalke, D. G., Borkar, A. Y., & Kolte, M. T. (2023).** Smart Inventory System Using IoT and Cloud Technology. *International Journal of Intelligent Systems and Applications in Engineering*, *12*(4s), 187–192.

****