Restaurant Menu Ordering System

A Mini Project Report

Submitted for

19EC552 - Microprocessors Laboratory

By

G.J.Jeyanthan (Roll No.22BEC170)

R.S.Selvaraj (Roll No.22BEC181)

III Year B.E ECE C (V Semester)

Department of Electronics and Communication Engineering

Academic Year 2024 - 2025

Mepco Schlenk Engineering College

(An Autonomous Institution Affiliated to Anna University, Chennai)

Mepco Engineering College (PO), Sivakasi – 626 005

Virudhunagar (District), Tamil Nadu

November 2024

TABLE OF CONTENTS

Abstract:

The Smart Restaurant Menu Ordering System leverages Arduino technology, an LCD display, a keypad, and Zigbee communication to create an efficient and user-friendly dining experience. This system allows customers to interact with a digital menu directly from their tables, facilitating easy selection and ordering of food items. The Arduino microcontroller serves as the central processing unit, managing inputs from the keypad and displaying menu options on the LCD screen.

Using Zigbee, a low-power wireless communication protocol, the system enables seamless data transmission between the customer interface and the restaurant's order management system. This ensures that orders are processed in real-time, reducing wait times and enhancing service efficiency. The keypad allows customers to navigate the menu intuitively, select items, and submit their orders with minimal effort.

This Smart Restaurant Menu Ordering System not only improves customer satisfaction by providing a modern and interactive ordering method but also optimizes restaurant operations by streamlining order management and reducing the likelihood of errors. Overall, the integration of Arduino, LCD, keypad, and Zigbee technology presents a comprehensive solution for modern dining establishments seeking to enhance their service delivery.

Components:

- Arduino
- LCD display
- Keypad
- Zigbee

Arduino Microcontroller :

Description: The Arduino microcontroller acts as the core of the ordering system, providing control over hardware interactions. As an open-source platform, it allows for flexible coding and hardware management in embedded applications.

Function: The Arduino processes user inputs from the keypad, manages the display output on the LCD, and communicates with the Zigbee module to transmit and receive order data. It executes the program that handles ordering, item selection, and quantity adjustments.

LCD Display :

Description: The LCD is a 20x4 character display used to communicate information to the user, such as menu options and order status.

Function: This display serves as the user interface, showing the menu, item codes, available quantities, and feedback messages. It guides the user through the ordering process, confirms orders, and displays system messages.

Keypad :

Description: The keypad is a 3x4 matrix input device enabling users to interact with the system.

Function: It allows users to select items, enter quantities, and place orders by pressing specific keys. Each keypress is interpreted by the Arduino, triggering corresponding actions in the ordering system, like item selection or order submission.

Zigbee Module :

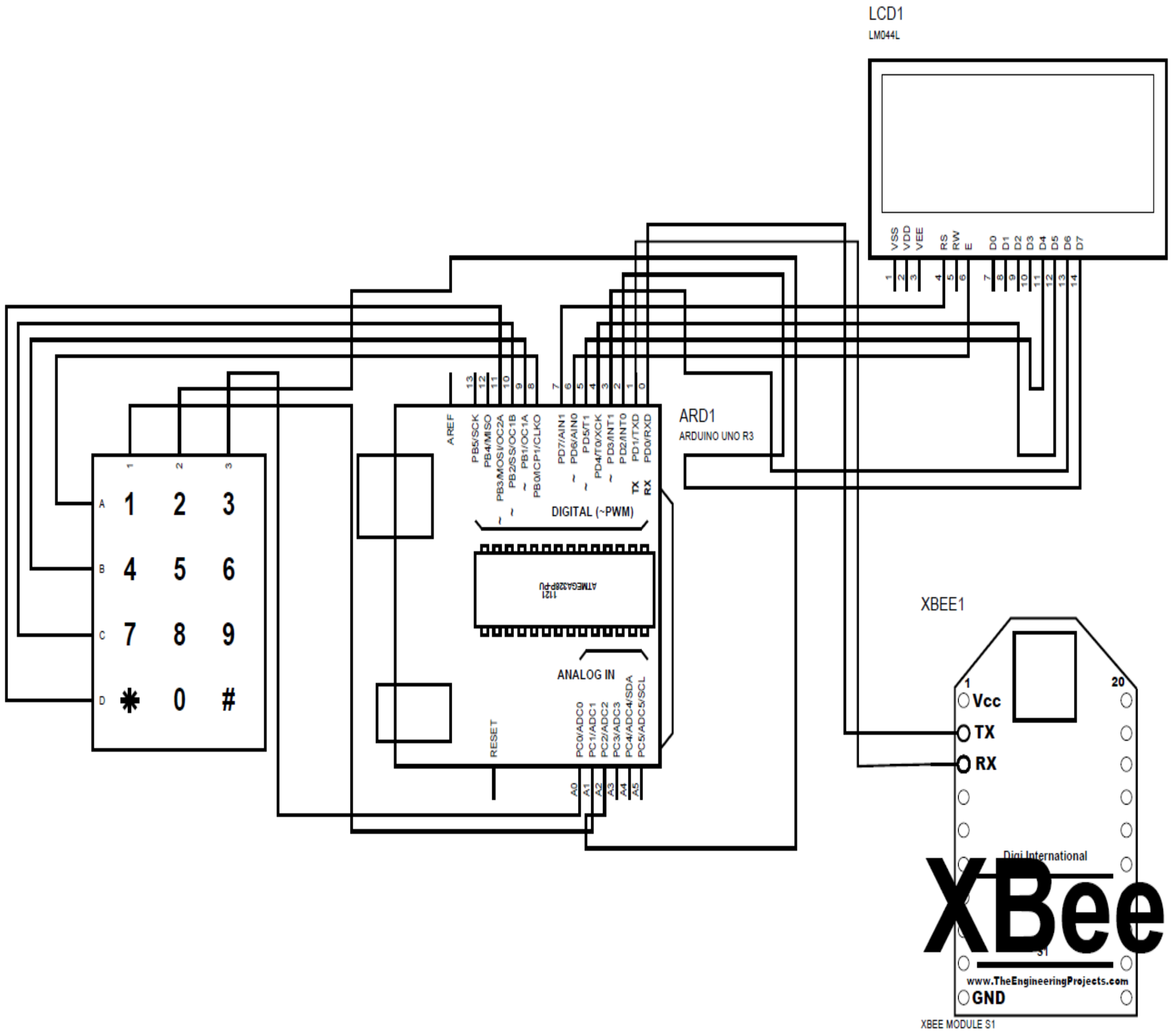
Description: The Zigbee module, typically an XBee series, facilitates wireless communication within the ordering system.

Function: It establishes a wireless connection between the Arduino and the restaurant's central order management system, allowing orders to be transmitted instantly. Zigbee’s low-power, high-efficiency communication enables real-time updates, enhancing the ordering experience.

In this project, UART (Universal Asynchronous Receiver/Transmitter) is used for serial communication between the Arduino microcontroller and other devices, such as a computer or the Zigbee module. UART is a widely used protocol for asynchronous serial data transmission that allows devices to communicate without needing a clock signal, making it both simple and effective for short-range communication.

Circuit Diagram:

Transmitter side:



The transmitter side of the Smart Restaurant Menu Ordering System plays a crucial role in taking user orders and communicating them wirelessly to the restaurant's order management system.

1. **Arduino Microcontroller**

Role: Acts as the primary control unit for the transmitter side.

Function: The Arduino reads user inputs from the keypad, processes them, and displays appropriate messages on the LCD screen. It then transmits the processed order information to the receiver side (restaurant management system) using the Zigbee module.

2. **LCD Display (20x4)**

Role: Provides a user-friendly interface for displaying the menu, input prompts, and feedback messages.

Function: The LCD shows menu items, item codes, available quantities, and confirms order placements. It can also display error messages if an invalid item code or quantity is entered, guiding users through the ordering process.

3. **Keypad (3x4 Configuration)**

Role: Allows users to interact with the system by selecting items and entering quantities.

Function: The keypad lets customers input item codes and quantities. Buttons like * and # are used to navigate through the ordering process, confirming selections and placing orders. The Arduino reads keypad inputs and translates them into commands.

4. **Zigbee Module (XBee)**

Role: Facilitates wireless communication between the transmitter (customer interface) and the receiver (order management system).

Function: Once the order details are confirmed, the Arduino uses UART (Universal Asynchronous Receiver-Transmitter) communication to send the order data to the Zigbee module. The Zigbee then transmits the data wirelessly to the receiver side in real-time, allowing the restaurant staff to receive and process the order without delay.

5. UART Communication

Role: Manages serial communication between the Arduino and Zigbee module for reliable data transmission.

Function: UART enables the Arduino to communicate with the Zigbee module by converting parallel data from the microcontroller into a serial data stream. The order data is sent byte-by-byte through UART to the Zigbee module, ensuring that each item code and quantity is correctly transmitted to the receiver. Overall Process When a customer begins an order, they are prompted to enter an item code and quantity. Once entered, the order is displayed on the LCD, and after confirmation, the Arduino updates the item stock. The order details are then sent to the receiver side through Zigbee, ensuring that the restaurant staff promptly receives the order, streamlining the service process. This setup enhances the ordering experience by providing a wireless, interactive, and efficient system, reducing wait times and minimizing order errors.

Transmitter side code:

Arduino code:

```
     // include the library code:
#include <LiquidCrystal.h>
#include <Keypad.h>
#include <Arduino.h>

// Initialize the LiquidCrystal library with the pins connected to the LCD
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// Keypad setup
const byte ROWS = 4; // Four rows
const byte COLS = 3; // Three columns
char keys[ROWS][COLS] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};
byte rowPins[ROWS] = {9,8,7,6}; // Connect to the row pinouts of the keypad
byte colPins[COLS] = {A2,A1,A0};  // Connect to the column pinouts of the
keypad
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
unsigned int quant[10] = {10, 16, 10, 10, 10, 10, 10, 10, 10, 10};
void setup() {
  Serial.begin(9600);
  unsigned char temp_str[2];
char ab=0;
char item_code[2];
    char flag =0;
    unsigned char quantity[3];   // To store quantity (up to 2 digits)
    unsigned char idx = 0;
    unsigned int order[20];
    unsigned char idx1 = 0;
    unsigned char idx2 = 0;
    char ba=0;
    char key;
  menu1:
  lcd.begin(20, 4);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Welcome to the");
  lcd.setCursor(0, 1);
  lcd.print("Hotel");
  lcd.setCursor(0, 2);
  lcd.print("Press * to");
  lcd.setCursor(0, 3);
  lcd.print("Continue");
  idx2=0;
lq1:        if(Serial.available() > 0)
        {
      uart_receive();
        goto menu1;
        }
lq2: while(keypad.getKey()!='*')
        {
            if (Serial.available() > 0)
```

```
                goto lq1;
            goto lq2;
        };

  menu:
        item_code[0]='\0';
      item_code[1]='\0';
      temp_str[0]='\0';
      temp_str[1]='\0';
      quantity[0]='\0';
      quantity[1]='\0';
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Enter the item");
  lcd.setCursor(0, 1);
  lcd.print("code and press *");
  lcd.setCursor(0, 2);
  lcd.print("to continue");
  idx = 0;
  idx1=0;
  lcd.setCursor(0, 3);
  while(1)
  {
    key=keypad.getKey();
    if(key!='\0'){
            if(key!='#'&&key!='*'){
            lcd.print(key);
            item_code[idx++]=key;
    }
    else if(key=='*'&&idx==0){
              lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("INVALID ITEM");
            lcd.setCursor(0, 1);
            lcd.print("CODE");
            delay(1000);
            goto menu;
                }
      else if(key=='*'&&idx!=0){
            if(item_code[0]=='0'&&idx<3){
                goto l2;
             }
            else{
                   lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("INVALID ITEM");
            lcd.setCursor(0, 1);
            lcd.print("CODE");
            delay(1000);
            goto menu;
             }
             l2:
                     lcd.clear();
                         lcd.setCursor(0, 0);
                        lcd.print("Enter quantity");
                lcd.setCursor(0, 2);
                lcd.print("Available: ");
                      ab=0;
                      ab = atoi(item_code);
                sprintf(temp_str, "%d", quant[ab]);
                lcd.print((char*)temp_str);
```

```
                    lcd.setCursor(0, 3);
                    lcd.print("#-place order");
                    lcd.setCursor(0, 1);
                    idx1 = 0;  // Reset quantity index

                    while(1) {
                        key = keypad.getKey();
                        if(key != '\0') {
                            if (key != '*' && key != '#') {
                                lcd.print(key);
                                quantity[idx1++] = key;  // Store the quantity
input
                            }
                            else if (key == '#'&& idx1!=0) {
                                quantity[idx1] = '\0';
                                                          ba=atoi(quantity);
                                                          if(ba<=quant[ab]){

quant[ab]=quant[ab]-ba;
                                                            order[idx2++]=ab;
                                                            order[idx2++]=ba;
                                                            lcd.clear();
                                        lcd.setCursor(0, 0);
                                                            lcd.print("Order
placed");
                                                            temp_str[0]='\0';
                                  temp_str[1]='\0';
                                order1(order, idx2);
                                delay(1000);
                                goto menu1;  // Return to the menu
                            }
                                                            else{
                                                            lcd.clear();
                                 lcd.setCursor(0, 0);
                                 lcd.print("Invalid");
                                                             lcd.setCursor(0,
1);
                                 lcd.print("quantity");
                                                                delay(1000);

quantity[0]='\0';
                                    quantity[1]='\0';
                                                                goto l2;
                                                            }
                                            }
                                                  else if (key == '*'  &&
idx1!=0){
                                                          ba=atoi(quantity);

if(ba<=quant[ab]&&ba!=0){

quant[ab]=quant[ab]-ba;
                                                            order[idx2++]=ab;
                                                            order[idx2++]=ba;
                                                            goto menu;
                                                }
                                                            else{
                                                            lcd.clear();
                                 lcd.setCursor(0, 0);
                                 lcd.print("Invalid");
lcd.setCursor(0, 1);
                             lcd.print("quantity");
                                                               delay(1000);

quantity[0]='\0';
                                   quantity[1]='\0';
                                                               goto l2;
                                                           }
                                               }
                        else if ((key == '#' || key == '*') &&idx1==0){
                              lcd.clear();
                            lcd.setCursor(0, 0);
                            lcd.print("Quantity not");
                             lcd.setCursor(0, 1);
                            lcd.print("Entered");
                            delay(1000);
                            lcd.clear();
                                                          goto l2;
                                          }


                        }
                    }
                }
}
  }
}
void order1(unsigned int *a, unsigned int idx2) {
    char temp_str[10];  // Buffer to hold each value as a string
    unsigned int *ptr = a;  // Pointer to iterate through the array

    while (ptr < a + idx2) {
        // Convert and send the item code
        snprintf(temp_str, sizeof(temp_str), "%u", *ptr);  // Convert item
code to string
        Serial.println(temp_str);  // Send item code followed by newline

        ptr++;  // Move to the next value (quantity)

        // Convert and send the quantity
        snprintf(temp_str, sizeof(temp_str), "%u", *ptr);  // Convert
quantity to string
        Serial.println(temp_str);  // Send quantity followed by newline

        ptr++;  // Move to the next item code
    }

    Serial.println("End of Order");  // Signal end of order
}
void uart_receive() {
    unsigned char received_item = 0;
    unsigned char received_quantity = 0;

    // Wait for the first character (item code, 0-9)
    while (Serial.available() == 0);  // Wait for the item code to be
received
    received_item = Serial.read() - '0';  // Convert ASCII to digit (0-9)

    // Wait for the second character (first digit of quantity)
    while (Serial.available() == 0);  // Wait for the first digit of
quantity
    received_quantity = (Serial.read() - '0') * 10;  // Multiply by 10 to
get tens place

    // Wait for the third character (second digit of quantity)
    while (Serial.available() == 0);  // Wait for the second digit of
quantity
    received_quantity += (Serial.read() - '0');  // Add the units place

    // Validate item code and update the quantity safely
    if (received_item < 10 && received_quantity < 100) {  // Ensure item
and quantity are valid
        quant[received_item] = received_quantity;  // Update the item
quantity

        lcd.clear();  // Clear the LCD
        lcd.print("Item ");  // Display item label
        lcd.print(received_item);  // Display item code
        lcd.print(" updated.");
        delay(1000); // Display updated message
        return;
    } else {
        Serial.println("Update failed");
        return;
    }
}
void loop() {

 }
```

Receiver side:

The receiver side of the smart restaurant menu ordering system is designed to handle incoming orders from customers and manage the associated data efficiently. **Key**

Features

1. **Connection to COM Port**:
   - The system establishes a connection to a COM port, allowing it to communicate with the transmitter side, where customer orders are generated. This connection is crucial for real-time order processing

2. **Data Reception**:
    - The receiver continuously listens for incoming data from the transmitter. It captures raw order data, which includes item codes and quantities, and processes this information to create a structured order.
3. **Order Processing**:
    - Upon receiving an "End of Order" signal, the system parses the incoming data to extract item codes and corresponding quantities. It validates this data to ensure that it conforms to expected formats (e.g., even number of entries representing item code and quantity pairs).
4. **Menu Integration**:
    - The receiver uses a predefined menu, where each item is associated with a unique code, name, and price. This integration allows the system to look up item details based on the received codes and calculate the total price for each order.
5. **Order Storage**:
    - All processed orders are stored in an array, enabling the system to keep track of multiple orders. Each order is assigned a unique order number for easy reference.
6. **Order Summary Notification**:
    - A notification section displays the summary of received orders. This allows restaurant staff to quickly view and access the details of incoming orders, enhancing workflow efficiency.
7. **Bill Generation**:
    - For each processed order, a bill is generated, detailing the items ordered, their quantities, individual prices, and the overall total. This bill can be displayed in a user-friendly table format.

8. **Editable Bill**:
   - The system provides functionality for staff to edit quantities directly in the bill table. As quantities are updated, the total amount is recalculated in real-time, ensuring accurate billing.
9. **Print Functionality**:
   - Once the bill is finalized, the system allows for printing the bill, making it easy for staff to provide physical copies to customers.
10. **User Interface**:
    - The receiver features a clean and intuitive user interface, allowing easy navigation between updating quantities, processing bills, and connecting to the COM port. It provides visual feedback through notifications and status updates.

HTML CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Restaurant Menu Ordering System</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Restaurant Menu Ordering System</h1>
        <button onclick="showSection('updateQuantity')">Update
Quantity</button>
        <button onclick="showSection('processBill')">Process Bill</button>
        <button class="connect-btn" onclick="connectCOMPort()">Connect to
COM Port</button>
        <p id="comStatus"></p>

        <!-- Quantity Section -->
        <div class="update-quantity">
            <h2>Update Quantity</h2>
            <div class="input-box">
                <label for="item-code">Item Code (0-9)</label>
                <input type="number" id="item-code" min="0" max="9"
placeholder="Enter Item Code" required>
```

```
            </div>
            <div class="input-box">
                <label for="quantity">Quantity (0-99)</label>
                <input type="number" id="quantity" min="0" max="99"
placeholder="Enter Quantity" required>
            </div>
            <button onclick="sendOrder()">Update Quantity</button>
            <p id="status"></p>
            <button class="back-btn" onclick="showMain()">Back</button>
        </div>

        <!-- Bill Section -->
        <div id="billContainer">
            <h2>Generated Bill</h2>
            <table id="billTable">
                <thead>
                    <tr>
                        <th>S.No</th>
                        <th>Item Code</th>
                        <th>Item Name</th>
                        <th>Quantity</th>
                        <th>Price (₹)</th>
                        <th>Total (₹)</th>
                    </tr>
                </thead>
                <tbody id="billBody"></tbody>
            </table>
            <button onclick="printBill()">Print Bill</button>
            <button class="back-btn" onclick="showMain()">Back</button>
        </div>

        <!-- Notification Section for Orders -->
        <div class="notification" id="notification"></div>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

CSS CODE:

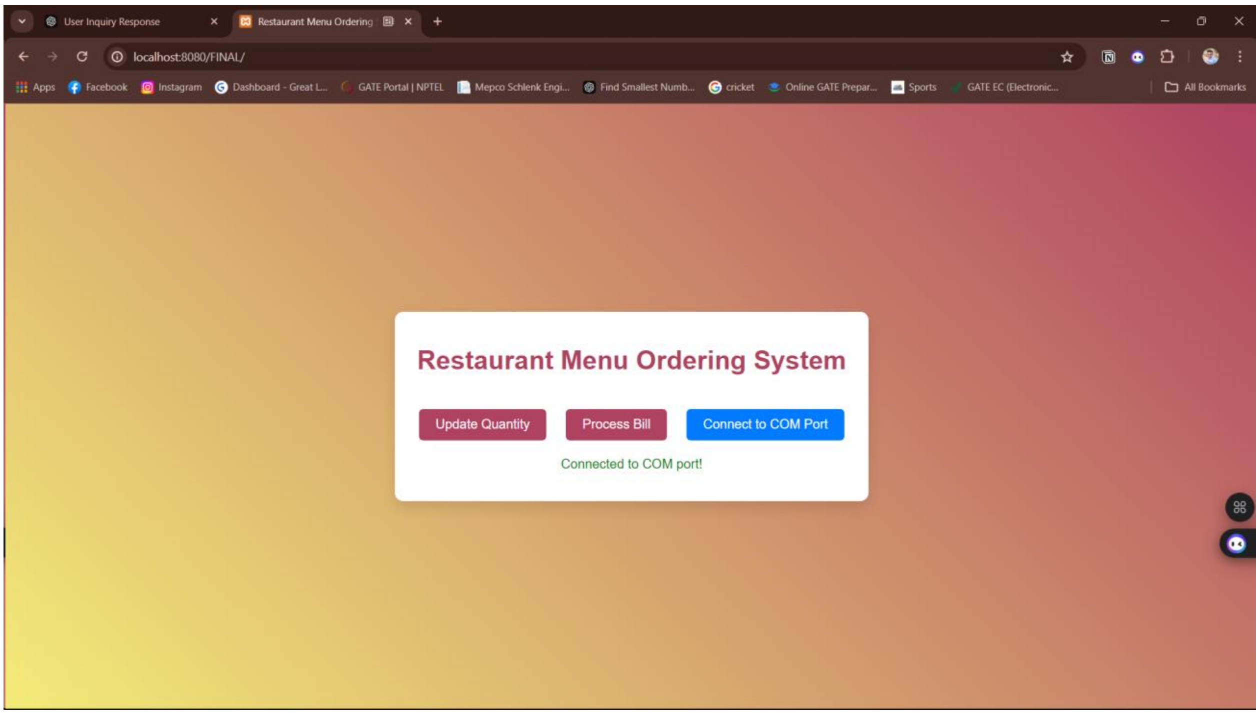
```
body {
    font-family: Arial, sans-serif;
    background: linear-gradient(45deg, #f3ec78, #af4261);
    color: #333;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
}
.container {
    background: #ffffff;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.1);
    text-align: center;
}
h1 { color: #af4261; }
button {
    background-color: #af4261;
    color: #ffffff;
    border: none;
    padding: 10px 20px;
    cursor: pointer;
    border-radius: 5px;
    margin: 10px;
    font-size: 16px;
}
button:hover { background-color: #f3ec78; color: #af4261; }
.connect-btn { background-color: #007bff; margin-top: 20px; }
.connect-btn:hover { background-color: #0056b3; }
.update-quantity, #billContainer { display: none; }
#comStatus, #status { margin-top: 10px; color: green; }

table { width: 100%; border-collapse: collapse; margin-top: 20px; }
th, td { padding: 10px; border: 1px solid #ddd; text-align: left; }
th { background-color: #f3ec78; color: #333; }
.notification { color: green; margin: 10px 0; cursor: pointer; }
```

JavaScript code:

```
// Define the Menu
const menu = [
    { code: 0, name: 'Idli', price: 10},
    { code: 1, name: 'Dosa', price: 25 },
    { code: 2, name: 'Sambar Rice', price: 50 },
    { code: 3, name: 'Vegetable Biryani', price: 80 },
    { code: 4, name: 'Pongal', price: 40 },
    { code: 5, name: 'Vada', price: 12 },
    { code: 6, name: 'Chettinad Chicken Curry', price: 150 },
    { code: 7, name: 'Fish Curry', price: 180 },
    { code: 8, name: 'Mutton Briyani', price: 220 },
    { code: 9, name: 'Parotta', price: 15 }
];


let port = null;
let orders = []; // To store received orders
let orderCounter = 1; // Counter to keep track of order numbers
let receivedData = '';  // Buffer to store incoming data

// Connect to the COM port
async function connectCOMPort() {
    try {
        if (!port) {
            port = await navigator.serial.requestPort();
            await port.open({ baudRate: 9600 });
            document.getElementById('comStatus').textContent = "Connected
to COM port!";
            listenForData(); // Start listening for incoming data
        }
    } catch (err) {
        document.getElementById('comStatus').textContent = "Failed to 
connect: " + err;
    }
}

// Listen for incoming data from the COM port
async function listenForData() {
    const reader = port.readable.getReader();
    try {
        while (true) {
            const { value, done } = await reader.read();
            if (done) {
                console.log("Stream closed");
                break;
            }
            if (value) {
                receivedData += new TextDecoder().decode(value);
                console.log("Received raw data:", receivedData); // Log 
received raw data for debugging

                // Process the incoming data when 'end of order' is 
detected
                if (receivedData.includes('End of Order')) {
                    processOrderData(receivedData);
                    receivedData = ''; // Clear buffer after processing
                }
            }
        }
    } catch (err) {
        console.error('Error reading data:', err);
    } finally {
        reader.releaseLock();
    }
}

// Process the received order data
function processOrderData(data) {
    const cleanedData = data.split('\n').map(line => 
line.trim()).filter(line => line !== ''); // Split data by lines and trim
    const items = cleanedData.slice(0, -1);  // Remove 'End of Order' line

    console.log("Parsed items from received data:", items); // Log parsed 
items for debugging

    // Check if the received data has valid pairs of item code and quantity
    if (items.length % 2 === 0) {
        let orderItems = [];
        for (let i = 0; i < items.length; i += 2) {
            const code = parseInt(items[i]);
            const quantity = parseInt(items[i + 1]);
            const menuItem = menu.find(item => item.code === code);
            if (menuItem) {
                orderItems.push({
                    code,
                    name: menuItem.name,
                    quantity,
                    price: menuItem.price,
                    total: menuItem.price * quantity
                });
            }
        }
        // Add the processed order to the orders array
        orders.push({
            orderNumber: orderCounter++,
            items: orderItems
        });

        displayOrderSummary(); // Update the order notification
    } else {
        document.getElementById('comStatus').textContent = "Error: Invalid
order data received.";
    }
}

// Display order summary in the notification
function displayOrderSummary() {
    const notification = document.getElementById('notification');
    notification.innerHTML = '';

    // Display each order in the notification
    orders.forEach(order => {
        const orderElement = document.createElement('div');
        orderElement.textContent = `Order ${order.orderNumber}`;
        orderElement.onclick = () => displayBill(order);
        notification.appendChild(orderElement);
    });
}

// Display the bill for an order
function displayBill(order) {
    const billBody = document.getElementById('billBody');
    billBody.innerHTML = ''; // Clear previous bill
    let grandTotal = 0;

    // Create rows for each item in the order
    order.items.forEach((item, index) => {
        const row = document.createElement('tr');
        const total = item.quantity * item.price;
        grandTotal += total;

        row.innerHTML = `
            <td>${index + 1}</td>
            <td>${item.code}</td>
            <td>${item.name}</td>
            <td contenteditable="true" oninput="updateTotal(this,
${item.price})">${item.quantity}</td>
            <td>${item.price}</td>
            <td>${total}</td>
        `;
        billBody.appendChild(row);
    });

    // Display grand total
    const totalRow = document.createElement('tr');
    totalRow.innerHTML = `<td colspan="5">Overall
Total:</td><td>${grandTotal}</td>`;
    billBody.appendChild(totalRow);

    document.getElementById('billContainer').style.display = 'block'; //
Show the bill container
    showSection('processBill'); // Switch to the bill section
```

```
}

// Update the total amount when quantity is edited
function updateTotal(cell, price) {
    const quantity = parseInt(cell.textContent);
    const totalCell = cell.parentElement.cells[5];
    totalCell.textContent = quantity * price;

    // Update overall total
    const billBody = document.getElementById('billBody');
    let grandTotal = 0;
    for (let i = 0; i < billBody.rows.length - 1; i++) {
        grandTotal += parseInt(billBody.rows[i].cells[5].textContent);
    }
    billBody.rows[billBody.rows.length - 1].cells[1].textContent =
grandTotal; // Update overall total
}

// Send order data to the COM port
// Send updated quantity for a specific item code
function sendOrder() {
    const itemCode = document.getElementById('item-code').value;
    const quantity = document.getElementById('quantity').value;
    const data = `${itemCode}${quantity}`; // Ensure correct format

    if (port) {
        const writer = port.writable.getWriter();
        const encoded = new TextEncoder().encode(data); // Send data as new
line separated
        writer.write(encoded);
        writer.releaseLock();
        document.getElementById('status').textContent = "Order sent
successfully!";
    } else {
        document.getElementById('status').textContent = "COM port not
connected.";
    }
}



// Switch between sections
function showSection(section) {
    document.querySelector('.update-quantity').style.display = section ===
'updateQuantity' ? 'block' : 'none';
    document.getElementById('billContainer').style.display = section ===
'processBill' ? 'block' : 'none';
}

// Hide all sections except main
function showMain() {
    document.querySelector('.update-quantity').style.display = 'none';
    document.getElementById('billContainer').style.display = 'none';
}

// Print the bill
function printBill() {
    window.print();
```

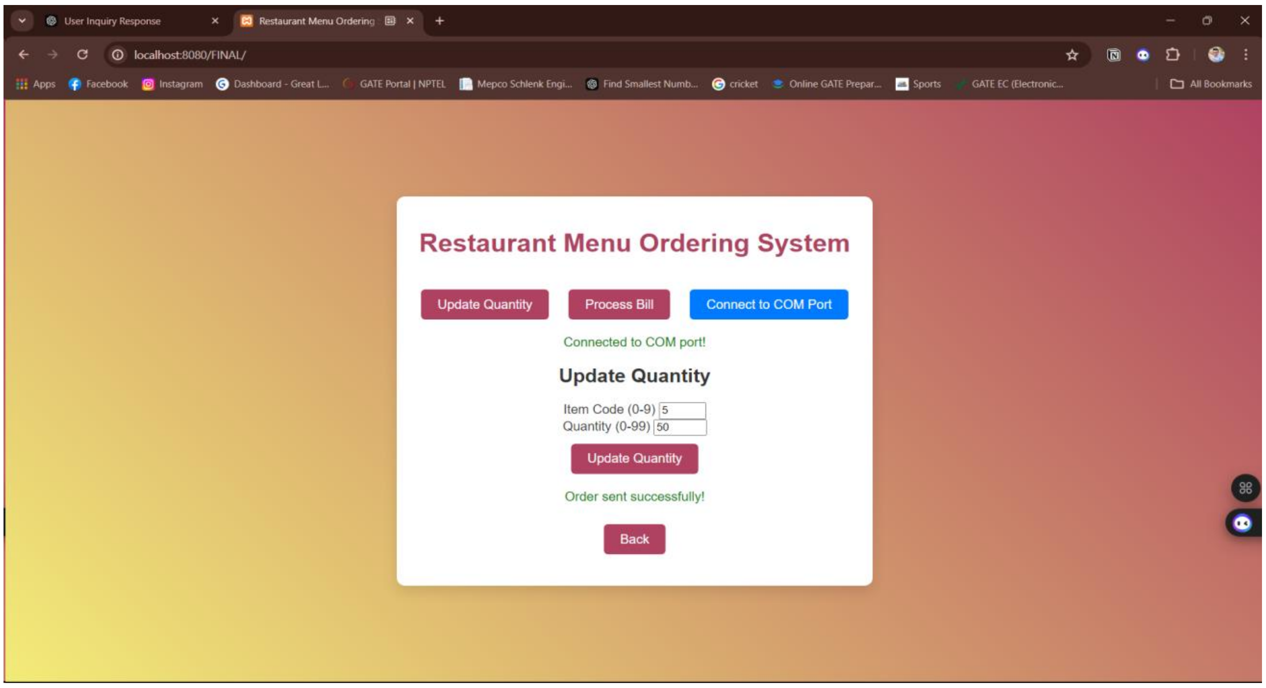
Receiver side portal:



Restaurant Menu Ordering System

Update Quantity | Process Bill | Connect to COM Port

Connected to COM port!



Restaurant Menu Ordering System

Update Quantity | Process Bill | Connect to COM Port

Connected to COM port!

Update Quantity

Item Code (0-9) 5
Quantity (0-99) 50

Update Quantity

Order sent successfully!

Back

Future scope:

 Mobile Application Development: Create a mobile app for customers to place orders and manage their profiles.

 Real-Time Order Tracking: Implement a feature for customers to track their orders from preparation to delivery.

 Online Payment Integration: Add secure online payment options for a seamless checkout experience.

 Dynamic Menu Management: Enable restaurant staff to update menu items and prices in real time through an admin interface.

 Loyalty Rewards Program: Introduce a system to reward frequent customers with points or discounts.

 Data Analytics and Reporting: Develop reporting features to provide insights into sales, customer preferences, and inventory management.

 IoT and Smart Kitchen Integration: Integrate smart kitchen devices to streamline order processing and enhance efficiency.

Conclusion:

the smart restaurant menu ordering system significantly enhances the dining experience by streamlining the ordering process, reducing errors, and expediting service. This efficiency allows restaurant staff to focus on delivering exceptional customer service. By incorporating data analytics, the system provides valuable insights into customer preferences and sales trends, enabling informed decision-making for restaurant owners. Its scalable design allows for future enhancements, such as online payments and mobile applications, ensuring adaptability as the restaurant evolves

MENU CARD:

Code	Item Name	Price (₹)
00	Idli	10
01	Dosa	25
02	Sambar Rice	50
03	Vegetable Biryani	80
04	Pongal	40
05	Vada	12
06	Chettinad Chicken Curry	150
07	Fish Curry	180
08	Mutton Biryani	220
09	Parotta	15