S3 BUCKET STATIC HOSTING

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::soobucket321/*"
    }
  ]
}
```

EC2 BACKEND

```
const express = require("express");
const app = express();
const PORT = 5000;

// Middleware to parse JSON body
app.use(express.json());

// Example menu items
const menu = [
 { id: 1, name: "Burger", price: 100 },
 { id: 2, name: "Fries", price: 50 },
 { id: 3, name: "Coke", price: 30 },
 { id: 4, name: "Pizza", price: 200 }
```

```
];

// GET endpoint to return available menu
app.get("/menu", (req, res) => {
  res.json({
    message: "Available menu items",
    items: menu
  });
});

// POST endpoint to place an order
app.post("/order", (req, res) => {
  const orderItems = req.body.items; // Expecting [{ id, quantity }]

  if (!orderItems || !Array.isArray(orderItems)) {
    return res.status(400).json({ error: "Invalid order format" });
  }

  let total = 0;
  let orderedDetails = [];

  orderItems.forEach(order => {
    const menuItem = menu.find(m => m.id === order.id);
    if (menuItem) {
      const cost = menuItem.price * order.quantity;
      total += cost;
      orderedDetails.push({
        name: menuItem.name,
```

```
      quantity: order.quantity,
      cost: cost
    });
  }
 });

 res.json({
  message: "Order received successfully",
  order: orderedDetails,
  totalPrice: total
 });
});

// Start server
app.listen(PORT, () => {
 console.log(`Server running on http://0.0.0.0:${PORT}`);
});
```

EC2 FRONTEND

```
import { useEffect, useState } from "react";

function App() {
 const [menu, setMenu] = useState([]);
 const [order, setOrder] = useState({});
 const [summary, setSummary] = useState(null);

 // Fetch menu from backend
 useEffect(() => {
```

```
  fetch("http://localhost:5000/menu")
    .then(res => res.json())
    .then(data => setMenu(data.items))
    .catch(err => console.error("Error fetching menu:", err));
}, []);

// Handle quantity change
const handleQuantityChange = (id, quantity) => {
  setOrder(prev => ({ ...prev, [id]: Number(quantity) }));
};

// Submit order
const placeOrder = () => {
  const items = Object.entries(order)
    .filter(([_, qty]) => qty > 0)
    .map(([id, qty]) => ({
      id: Number(id),
      quantity: qty
    }));

  fetch("http://localhost:5000/order", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ items })
  })
    .then(res => res.json())
    .then(data => setSummary(data))
    .catch(err => console.error("Error placing order:", err));
```

```
};

return (
  <div style={{ padding: "20px", fontFamily: "Arial, sans-serif" }}>
    <h1>🍔 Simple Order System</h1>

    <h2>Menu</h2>
    {menu.length === 0 && <p>Loading menu...</p>}
    <ul>
      {menu.map(item => (
        <li key={item.id} style={{ margin: "10px 0" }}>
          <strong>{item.name}</strong> - ₹{item.price}
          <input
            type="number"
            min="0"
            placeholder="Qty"
            style={{ marginLeft: "10px", width: "60px" }}
            onChange={e => handleQuantityChange(item.id, e.target.value)}
          />
        </li>
      ))}
    </ul>

    <button
      onClick={placeOrder}
      style={{
        marginTop: "20px",
        padding: "10px 20px",
```

```
        backgroundColor: "#007bff",
        color: "white",
        border: "none",
        borderRadius: "5px",
        cursor: "pointer"
      }}
    >
      Place Order
    </button>

    {summary && (
      <div style={{ marginTop: "30px" }}>
        <h2>Order Summary</h2>
        <ul>
          {summary.order.map((item, idx) => (
            <li key={idx}>
              {item.quantity} × {item.name} = ₹{item.cost}
            </li>
          ))}
        </ul>
        <h3>Total: ₹{summary.totalPrice}</h3>
      </div>
    )}
  </div>
 );
}

export default App;
```

LAMBDA

```
import boto3
import csv
import io

def lambda_handler(event, context):
    # Get bucket and file name from the S3 event
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    print(f"Processing file: s3://{bucket}/{key}")

    s3 = boto3.client('s3')

    # Download CSV from S3
    obj = s3.get_object(Bucket=bucket, Key=key)
    data = obj['Body'].read().decode('utf-8')

    # Parse CSV
    csv_reader = csv.DictReader(io.StringIO(data))
    for row in csv_reader:
        try:
            mark = float(row['mark'])
            if mark > 70:
                print(f"Student: {row.get('name','Unknown')}, Mark: {mark}")
        except ValueError:
            print(f"Skipping invalid mark: {row.get('mark')}")
```

```
    return {
        'statusCode': 200,
        'body': f"Processed {key}"
    }
```

```
name,mark
Alice,85
Bob,65
Charlie,92
```

VM – SOCKET – SERVER

SERVER

```
import java.io.*;
import java.net.*;
import java.util.*;

public class OrderServer {

    private static List<Map<String, Object>> menu = List.of(
        Map.of("id", 1, "name", "Burger", "price", 100),
        Map.of("id", 2, "name", "Fries", "price", 50),
        Map.of("id", 3, "name", "Coke", "price", 30),
        Map.of("id", 4, "name", "Pizza", "price", 200)
    );

    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(5000);
        System.out.println("Order server listening on port 5000...");
```

```
        while (true) {
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " + clientSocket.getInetAddress());

            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            // Send menu to client
            out.println(menu);

            // Receive order from client
            String orderLine = in.readLine(); // expecting a simple format:
id:quantity,id:quantity,...
            String[] items = orderLine.split(",");
            int total = 0;
            List<String> orderDetails = new ArrayList<>();

            for (String item : items) {
                String[] parts = item.split(":");
                int id = Integer.parseInt(parts[0]);
                int quantity = Integer.parseInt(parts[1]);

                Map<String, Object> menuItem = menu.stream()
                        .filter(m -> (int)m.get("id") == id)
                        .findFirst().orElse(null);

                if (menuItem != null) {
```

```
            int cost = (int) menuItem.get("price") * quantity;
            total += cost;
            orderDetails.add(quantity + " x " + menuItem.get("name") + " = " + cost);
          }
        }

        out.println("Order Summary:");
        orderDetails.forEach(out::println);
        out.println("Total: " + total);

        clientSocket.close();
      }
    }
}
```

CLIENT

```
import java.io.*;
import java.net.*;

public class OrderClient {
   public static void main(String[] args) throws IOException {
      // Replace with VM IP
      String serverIP = "192.168.x.x";
      int port = 5000;

      Socket socket = new Socket(serverIP, port);
      BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
```

```
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

        // Read menu from server
        System.out.println("Menu from server:");
        String line;
        for (int i = 0; i < 4; i++) {  // assuming 4 menu items
            System.out.println(in.readLine());
        }

        // Input order: format id:quantity,id:quantity,...
        System.out.print("Enter your order (e.g., 1:2,2:1): ");
        String orderInput = stdin.readLine();

        // Send order to server
        out.println(orderInput);

        // Read order summary
        System.out.println("\nOrder Summary from server:");
        while ((line = in.readLine()) != null) {
            System.out.println(line);
        }

        socket.close();
    }
}
```