I'm going to build a web app and mobile app for a restaurant, i have to paste a qr code for evry table, every houses and every flats in the multistorey building. If a person order in table 1, that customer become the permanent member/customer of my app. The another customer sits in the same table later than that customer also become the permanent customer. In a houses or flats if a customer scan the qr code that should be their location, sometimes they will order from the differemt places like offices or any where. I have to send notification for the every customer for every day special offers or dishes. I want to notify the pay counter and kitchen to take orders if any orders placed from any customer. For this thing i want logical code for this

**ChatGPT said:**

Updated saved memory

You're building a location-aware food ordering system that:

1. Registers every customer as a permanent user after scanning a QR code.
2. Uses QR codes for:
    - **Restaurant tables** (static locations, like Table 1).
    - **Homes/flats** (semi-static, assigned to a location).
    - **Other locations** (dynamic like offices).
3. Sends **daily offers** to all customers.
4. Sends **real-time order notifications** to:
    - **Kitchen**.
    - **Payment counter**.

---

### 🔧 Logical Structure & Code Plan

Here's a modular approach with logic for each major part:

---

# 1. QR Code Structure and Behavior

Each QR code encodes:

json
CopyEdit

```
{
  "type": "table" | "home" | "flat",
  "location_id": "TBL001" | "HSE015" | "FLT204",
  "restaurant_id": "REST001"
}
```

When scanned:

ts
CopyEdit

```
// Pseudocode (TypeScript/Node.js style)

function handleQRCodeScan(userId: string, qrData: QRData) {
  let user = getUser(userId);
  if (!user) {
    user = createUser(userId);
  }

  // Log location context
  setUserLocation(user.id, qrData.type, qrData.location_id);

  // Create session or token for this visit
  const session = createUserSession(user.id, qrData);
  return session;
}
```

---

## 2. Placing an Order

ts
CopyEdit

```
function placeOrder(userId: string, items: Item[], currentLocation:
LocationContext) {
  const order = saveOrderToDatabase(userId, items, currentLocation);

  // Notify kitchen and counter
  notifyKitchen(order);
```

```
  notifyCounter(order);

  return order;
}
```

---

## 3. Becoming Permanent Customer

ts
CopyEdit

```
function createUser(userId: string): User {
  return db.users.insert({
    id: userId,
    joined_at: new Date(),
    is_permanent: true
  });
}
```

---

## 4. Send Notifications for Daily Offers

Use a CRON job or scheduler:

ts
CopyEdit

```
// Pseudocode for daily job
scheduleDaily('08:00', () => {
  const users = getAllPermanentUsers();
  users.forEach(user => {
    sendPushNotification(user.device_token, {
      title: "Today's Special!",
      message: "Try our Chef's Special Paneer Tikka with 15% off!",
    });
  });
});
```

---

# 5. Push Notification for Orders (Kitchen & Pay Counter)

Use WebSocket or Firebase/OneSignal:

ts
CopyEdit

```
function notifyKitchen(order: Order) {
  sendRealTimeNotification('kitchen', {
    title: "New Order",
    message: `Order #${order.id} from ${order.location_id}`,
    data: order,
  });
}

function notifyCounter(order: Order) {
  sendRealTimeNotification('counter', {
    title: "Payment Pending",
    message: `Order #${order.id} requires processing.`,
    data: order,
  });
}
```

---

## 📱 Mobile/Web App Structure

- **QR Scanner** – Scans and decodes QR, logs location.
- **Menu Display** – Show menu with prices.
- **Cart + Order** – Confirm and submit.
- **Notification Permissions** – Enable push.
- **Order History** – For repeat users.
- **Settings/Profile** – Track permanent customer details.

---

## 🔒 Auth and Identity Options

- **Phone Number Login or WhatsApp Auth** for first-time users.
- Store device tokens (for notifications) and user metadata.

---

## ✅ Tech Stack Suggestions

- **Frontend:** React (Web), React Native or Flutter (Mobile)
- **Backend:** Node.js + Express / NestJS
- **Database:** PostgreSQL or Firebase Firestore
- **QR Code Gen:** `qrcode` (Node.js package)
- **Notifications:** Firebase Cloud Messaging / OneSignal
- **Real-time updates:** WebSockets / Firebase Realtime DB

---

Would you like me to generate code for a specific part next? (e.g. QR scan flow, order placement API, notification logic)

**You said:**
Qr scan flow

**ChatGPT said:**

Great — let's build the **QR scan flow** step-by-step, for both **frontend (mobile/web app)** and **backend**, assuming the goal is:

- A customer scans a QR code.
- The app identifies the location (table, home, flat).
- The customer becomes a permanent user (if not already).
- The location is stored for the current order session.

---

# 🔧 STEP-BY-STEP QR SCAN FLOW

## 🧩 QR CODE FORMAT (What's embedded inside)

Example encoded as a URL:

bash
CopyEdit
```
https://yourrestaurant.com/order?type=table&id=TBL001
```

Or JSON (if using native mobile apps):

json
CopyEdit
```
{
  "type": "table",
  "location_id": "TBL001"
}
```

---

# 📱 FRONTEND: React Native (or Web - logic is similar)

tsx
CopyEdit
```
// react-native (expo-compatible example)
import { BarCodeScanner } from 'expo-barcode-scanner';
import { useState, useEffect } from 'react';
import { Alert } from 'react-native';
import axios from 'axios';

function QRScannerScreen({ navigation }) {
  const [hasPermission, setHasPermission] = useState(null);

  useEffect(() => {
    (async () => {
      const { status } = await
BarCodeScanner.requestPermissionsAsync();
      setHasPermission(status === 'granted');
    })();
```

```
  }, []);

  const handleQRCodeScanned = async ({ data }) => {
    try {
      const qrData = JSON.parse(data); // or parse query string if
it's a URL
      const userId = await getUserIdFromStorage(); // Get from
local/session

      const response = await
axios.post('https://api.yourrestaurant.com/qr-scan', {
        user_id: userId,
        qr_data: qrData,
      });

      navigation.navigate('Menu', { session_id:
response.data.session_id });
    } catch (error) {
      Alert.alert("Scan Failed", "Invalid QR or network error.");
    }
  };

  if (hasPermission === null) return null;
  if (hasPermission === false) return <Text>No access to
camera</Text>;

  return (
    <BarCodeScanner
      onBarCodeScanned={handleQRCodeScanned}
      style={{ flex: 1 }}
    />
  );
}
```

---

## 🖥️ BACKEND (Node.js + Express)

ts

CopyEdit

```
// POST /qr-scan
app.post("/qr-scan", async (req, res) => {
  const { user_id, qr_data } = req.body;

  // qr_data: { type: "table" | "home" | "flat", location_id: "TBL001"
}
  const user = await getOrCreateUser(user_id);
  const session = await createSession(user.id, qr_data);

  res.json({ success: true, session_id: session.id });
});

async function getOrCreateUser(user_id: string) {
  let user = await db.users.findUnique({ where: { id: user_id } });
  if (!user) {
    user = await db.users.create({
      data: {
        id: user_id,
        is_permanent: true,
        joined_at: new Date()
      }
    });
  }
  return user;
}

async function createSession(userId: string, qrData: any) {
  return await db.sessions.create({
    data: {
      user_id: userId,
      location_type: qrData.type,
      location_id: qrData.location_id,
      started_at: new Date()
    }
  });
}
```

---

# 🗂️ DATABASE STRUCTURE (Simplified)

### `users` table:

ts
CopyEdit

```
{
  id: string,
  joined_at: Date,
  is_permanent: boolean,
  phone_number?: string,
  whatsapp_id?: string
}
```

### `sessions` table:

ts
CopyEdit

```
{
  id: string,
  user_id: string,
  location_type: 'table' | 'home' | 'flat',
  location_id: string,
  started_at: Date
}
```

---

# ✅ OPTIONAL IMPROVEMENTS

- Attach device token during scan for push notifications.
- Add session expiration logic.
- Store `lat/lon` if user scans from a dynamic location (office/travel).
- Display location-aware menu (special table-only or flat-only dishes).

---

Would you like me to generate the **QR code generator script**, or move next to the **order placing flow**?

**You said:**

Order placement api

**ChatGPT said:**

Here's the complete **Order Placement API** logic for your restaurant app — covering backend structure, business logic, and real-time notifications to kitchen and pay counter.

---

## 🔥 Flow Summary

1. Customer places an order from a scanned location (table/flat/home).
2. Order is stored in the DB.
3. Kitchen and Pay Counter get notified in real-time.
4. Order is linked to the current session.

---

## 🧾 Order Data Structure

Sample incoming payload from app:

json
CopyEdit

```
{
  "session_id": "SESSION1234",
  "user_id": "USER5678",
  "items": [
    { "item_id": "ITEM001", "quantity": 2 },
    { "item_id": "ITEM005", "quantity": 1 }
  ],
  "notes": "Extra spicy, no onions"
}
```

---

## 📦 Backend: `POST /place-order`

### Example (Node.js + Express + Prisma ORM):

ts
CopyEdit

```
app.post("/place-order", async (req, res) => {
  const { session_id, user_id, items, notes } = req.body;

  try {
    // Validate session
    const session = await db.session.findUnique({ where: { id:
session_id } });
    if (!session) return res.status(400).json({ error: "Invalid
session" });

    // Create order
    const order = await db.order.create({
      data: {
        user_id,
        session_id,
        location_id: session.location_id,
        location_type: session.location_type,
        notes,
        status: "pending",
        placed_at: new Date()
      }
    });

    // Add order items
    for (let item of items) {
      await db.orderItem.create({
        data: {
          order_id: order.id,
          item_id: item.item_id,
          quantity: item.quantity
        }
      });
    }
```

```
    // Notify kitchen & counter (see below)
    notifyKitchen(order.id);
    notifyCounter(order.id);

    res.json({ success: true, order_id: order.id });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Server error" });
  }
});
```

---

## 🔔 Real-Time Notification Logic

Let's assume you use WebSockets or Firebase Cloud Messaging (FCM).

### A. WebSocket Approach (Example):

ts
CopyEdit

```
function notifyKitchen(orderId) {
  const message = {
    type: "new_order",
    target: "kitchen",
    order_id: orderId
  };
  sendWebSocketTo("kitchen", message);
}

function notifyCounter(orderId) {
  const message = {
    type: "new_order",
    target: "counter",
    order_id: orderId
  };
  sendWebSocketTo("counter", message);
}
```

### B. Firebase Cloud Messaging (Optional for App-Based UI)

ts
CopyEdit

```
function notifyKitchen(order) {
  const message = {
    notification: {
      title: "New Order",
      body: `Order #${order.id} from ${order.location_id}`
    },
    token: kitchenDeviceToken
  };
  firebaseAdmin.messaging().send(message);
}
```

---

## 🗂️ Database Schema (Simplified)

### `orders` table

ts
CopyEdit

```
{
  id: string,
  user_id: string,
  session_id: string,
  location_type: 'table' | 'home' | 'flat',
  location_id: string,
  notes: string,
  status: 'pending' | 'completed' | 'cancelled',
  placed_at: Date
}
```

### `order_items` table

ts
CopyEdit

```
{
  id: string,
```

```
  order_id: string,
  item_id: string,
  quantity: number
}
```

---

## ✅ Notes

- You can also add a `payment_status` field if integrated with Razorpay, Stripe, etc.
- If you want to allow **reordering**, store menu snapshots in the order (in case prices change later).
- Add `customer_name` if you support guest checkout without login.

---

Would you like:

- Code for generating QR codes for each location?
- Admin panel API to add/edit daily offers?
- Notification scheduler logic for daily offers to customers?

Let me know where to go next!

**You said:**
Notification logic

**ChatGPT said:**

Here's the **Notification Logic** you need — for **daily offers** to all customers, and **real-time order alerts** to the **kitchen** and **pay counter**.

---

## ✅ Notification Types

### 1. Real-Time Order Notifications

- When an order is placed → notify:
    - Kitchen
    - Pay counter

### 2. Daily Offer Notifications

- Sent to all **permanent customers** (users who have scanned and ordered before)
- Scheduled once a day (e.g., every morning at 8 AM)

---

# 🔔 Push Notification Setup Options

Use one of the following services:

- **Firebase Cloud Messaging (FCM)** — Free, mobile/web support.
- **OneSignal** — Easy setup, rich analytics, UI-friendly.
- **Socket.IO/WebSockets** — For live internal updates (kitchen/counter).

---

# 💥 PART 1: Real-Time Order Notifications

### 📦 Backend Code (Node.js + Firebase)

ts
CopyEdit

```
// Firebase Admin setup
const admin = require("firebase-admin");
admin.initializeApp({
  credential: admin.credential.cert(YOUR_FIREBASE_SERVICE_ACCOUNT)
});

async function notifyKitchen(order) {
```

```
  const kitchenDeviceTokens = await getDeviceTokens('kitchen'); //
from DB

  const message = {
    notification: {
      title: "New Order Received",
      body: `Order #${order.id} from
${order.location_type.toUpperCase()} ${order.location_id}`
    },
    tokens: kitchenDeviceTokens
  };

  await admin.messaging().sendMulticast(message);
}

async function notifyCounter(order) {
  const counterDeviceTokens = await getDeviceTokens('counter');

  const message = {
    notification: {
      title: "New Order to Process",
      body: `Order #${order.id} needs payment processing.`
    },
    tokens: counterDeviceTokens
  };

  await admin.messaging().sendMulticast(message);
}
```

### `getDeviceTokens` Example:

ts
CopyEdit

```
async function getDeviceTokens(role) {
  const users = await db.users.findMany({ where: { role } }); // role:
'kitchen', 'counter'
  return users.map(u => u.device_token).filter(Boolean);
}
```

---

# 🌅 PART 2: Daily Offer Notifications to All Customers

## 🎯 Goal:

Send daily dish offers (e.g., “Buy 1 Get 1 Free”) to all users who’ve scanned and ordered before.

---

## 📆 Scheduler (CRON + Node.js)

Use `node-cron` or set a daily job (e.g., at 8:00 AM):

bash
CopyEdit

```
npm install node-cron
```

ts
CopyEdit

```
const cron = require('node-cron');

cron.schedule('0 8 * * *', async () => {
  const users = await db.users.findMany({ where: { is_permanent: true
} });

  const tokens = users.map(u => u.device_token).filter(Boolean);

  const message = {
    notification: {
      title: "🍽️ Today’s Special Offer!",
      body: "Paneer Butter Masala at 20% off only today!"
    },
    tokens
  };

  const response = await admin.messaging().sendMulticast(message);
  console.log(`Notifications sent: ${response.successCount}`);
});
```

---

### 🔐 Device Token Storage

When a user logs in the first time:

ts
CopyEdit
```
POST /save-device-token
{
  "user_id": "USER123",
  "device_token": "fcmtoken123456"
}
```

ts
CopyEdit
```
// Save token to DB
await db.users.update({
  where: { id: user_id },
  data: { device_token }
});
```

---

## 🧠 Tips

- You can categorize users (based on location) to send targeted offers (e.g., flat-only discounts).
- If you support WhatsApp login, you can integrate with Twilio or Gupshup for WhatsApp message offers.
- Store last sent time in DB to avoid duplicate notifications.

---

## 📚 Optional Table for Tracking Notifications

### `notifications_sent`

ts

CopyEdit

```
{
  id: string,
  user_id: string,
  type: "daily_offer" | "order_alert",
  sent_at: Date,
  message: string
}
```

---

## Want This Visualized?

I can generate a **flow diagram** or a **WebSocket version** for internal live updates (without Firebase), if you're building this for kitchen screens.

Would you like that?

**You said:**
Qr code generation script

**ChatGPT said:**

Here's a complete **QR Code Generation Script** — so you can generate **unlimited QR codes** for:

- Restaurant tables (e.g., `Table-1`, `Table-2`, ...)
- Houses (e.g., `House-101`)
- Flats (e.g., `Flat-A201`)

Each QR code will point to your web app (or mobile app deep link) and carry location metadata.

---

## ✅ What It Will Do

- Accept a list of location types (`table`, `house`, `flat`) and IDs.
- Generate a QR code with encoded URL or JSON data.

- Save it as PNG or SVG.

---

## 🧰 Requirements (Node.js)

Install these packages:

bash
CopyEdit

```
npm install qrcode fs path
```

---

## 🧾 Example Script (`generateQRCodes.js`)

js
CopyEdit

```
const QRCode = require('qrcode');
const fs = require('fs');
const path = require('path');

// 🔧 Configuration
const BASE_URL = 'https://yourrestaurant.com/order'; // your web app
URL
const OUTPUT_DIR = './qrcodes';

// Example locations
const locations = [
  { type: 'table', id: 'TBL001' },
  { type: 'table', id: 'TBL002' },
  { type: 'house', id: 'HOUSE101' },
  { type: 'flat', id: 'FLAT-A301' },
  { type: 'flat', id: 'FLAT-B204' },
  // Add more as needed
];

// Ensure output directory exists
if (!fs.existsSync(OUTPUT_DIR)) {
```

```
  fs.mkdirSync(OUTPUT_DIR);
}

// Generate QR codes
locations.forEach(async (loc) => {
  const dataUrl = `${BASE_URL}?type=${loc.type}&id=${loc.id}`;

  const fileName =
`${loc.type}-${loc.id}.png`.replace(/[^a-zA-Z0-9-_\.]/g, '_');
  const filePath = path.join(OUTPUT_DIR, fileName);

  try {
    await QRCode.toFile(filePath, dataUrl, {
      width: 300,
      margin: 2,
      color: {
        dark: '#000000',
        light: '#ffffff'
      }
    });
    console.log(`✅ QR code saved: ${filePath}`);
  } catch (err) {
    console.error(`❌ Failed to generate for ${loc.id}:`, err);
  }
});
```

---

## 📦 What the QR Contains

Example for Table-1:

bash
CopyEdit
`https://yourrestaurant.com/order?type=table&id=TBL001`

When scanned, your app will:

- Decode the `type` and `id`
- Start a session for the user
- Track location

---

## 📲 Optional: Encode JSON Instead of URL

If your app reads raw JSON:

js
CopyEdit

```
const data = {
  type: loc.type,
  location_id: loc.id,
  restaurant_id: "REST001"
};
await QRCode.toFile(filePath, JSON.stringify(data), options);
```

---

## 🧩 Usage Ideas

| Location Type | ID Format | Placement |
|---|---|---|
| `table` | `TBL001` | On each restaurant table |
| `house` | `HOUSE123` | Door or wall outside home |
| `flat` | `FLAT-A203` | Door or lift lobby |

---

## Want This as a Web Tool?

I can convert this script into a simple **React/Next.js tool** with a form to generate QR codes on the fly and download them in ZIP format.

Would you like that next?

**You said:**

From the above answers/your suggestion what are yet not provided

**ChatGPT said:**

Great question — here's a quick **checklist** of everything you asked for so far, and what's **already provided vs. pending** based on your restaurant web/mobile app requirements:

---

## ✅ Already Provided

| Feature / Module | Status | Notes |
|---|---|---|
| **QR Code Scan Flow** | ✅ Done | React Native + Backend session creation |
| **Order Placement API** | ✅ Done | Saves order, links session, notifies kitchen & counter |
| **Notification Logic** | ✅ Done | Real-time (order), scheduled (daily offers), using Firebase |
| **QR Code Generation Script** | ✅ Done | Node.js script with PNG output and dynamic location support |

---

## 🔜 Not Yet Provided (Can Be Built Next)

| Feature / Module | Description |
|---|---|
| **Customer Login (Phone/WhatsApp)** | Logic to log in users via mobile number or WhatsApp, and store user info |
| **Admin Panel API (Daily Offers)** | Admin interface or API to manage daily offers (title, message, time, image) |
| **Order Status Flow** | APIs/UI for kitchen/counter to mark order as "preparing", "ready", "paid", etc. |
| **Menu Management (Admin)** | CRUD API or tool for adding/editing menu items, pricing, categories |
| **Reorder / Order History** | Show previous orders to user, allow 1-click reordering |

| | |
|---|---|
| **Customer Location Tracking (for dynamic locations like offices)** | Optional GPS/location data capture for non-fixed QR scans |
| **Push Notification Opt-in + Token Storage** | Mobile/web logic to store FCM tokens securely during login or scan |
| **Web Dashboard (Kitchen/Counter View)** | Real-time order display panel for kitchen and pay counter |
| **Multi-Restaurant Support (Optional)** | If you later support multiple branches, add restaurant ID to all QR/session logic |

---

Would you like to continue with any of these next?

For example:

- 🔐 **User Login Flow with Phone/WhatsApp**
- 📋 **Admin API for Managing Daily Offers**
- 📦 **Kitchen/Counter Dashboard UI**

Let me know your top priority and I'll build it step-by-step.

**You said:**
Admin panel api

**ChatGPT said:**

Perfect — let's now define the **Admin Panel API** for your restaurant app.

This API will allow the admin (you or staff) to:

- ✅ Create & manage daily offers
- ✅ Manage menu items (optional)
- ✅ View orders (optional)
- ✅ Secure access via admin login

---

# 📦 Admin Panel API Scope (MVP)

We'll start with:

1. **Admin Login (with token-based auth)**
2. **Create/Update/Delete Daily Offers**
3. **Get List of Offers (for mobile app or scheduler)**

Later you can extend it to:

- Menu management
- Order overview
- Sales reports

---

# 🛠️ Tech Stack Assumption

- Backend: Node.js + Express
- Auth: JWT (JSON Web Token)
- DB: PostgreSQL or MongoDB (you can adapt based on your stack)

---

# 1️⃣ Admin Login API

## 🔐 POST `/admin/login`

json
CopyEdit

```
// Request
{
  "email": "admin@yourrestaurant.com",
  "password": "admin123"
```

```
}
```

json
CopyEdit

```
// Response
{
  "token": "JWT_TOKEN_HERE"
}
```

### 🔐 Sample Code

ts
CopyEdit

```
const jwt = require('jsonwebtoken');
const SECRET = 'your_jwt_secret';

app.post('/admin/login', async (req, res) => {
  const { email, password } = req.body;

  const admin = await db.admin.findUnique({ where: { email } });
  if (!admin || admin.password !== password) return
res.status(401).json({ error: "Invalid credentials" });

  const token = jwt.sign({ adminId: admin.id }, SECRET, { expiresIn:
'1d' });
  res.json({ token });
});
```

---

## 2️⃣ Create Daily Offer

### 🔧 POST `/admin/offers`

json
CopyEdit

```
// Headers
Authorization: Bearer JWT_TOKEN

// Body
```

```
{
  "title": "Paneer Butter Masala 20% Off",
  "message": "Only for today!",
  "image_url": "https://example.com/paneer.jpg",
  "active_date": "2025-08-07"
}
```

### 🧠 Example DB Table: `offers`

| id | title | message | image_url | active_date |
|---|---|---|---|---|

### 🔧 Code Example

ts
CopyEdit

```
app.post('/admin/offers', verifyAdmin, async (req, res) => {
  const { title, message, image_url, active_date } = req.body;
  const offer = await db.offers.create({
    data: { title, message, image_url, active_date: new
Date(active_date) }
  });
  res.json({ success: true, offer });
});
```

---

## 3️⃣ Get Offers (for user app or notifications)

### 📥 GET `/offers?date=2025-08-07`

Returns today's active offers.

json
CopyEdit

```
[
  {
    "title": "Paneer Butter Masala 20% Off",
    "message": "Only for today!",
    "image_url": "https://example.com/paneer.jpg"
```

```
  }
]
```

---

## 4️⃣ Update or Delete Offer (Optional)

ts
CopyEdit

```
// PUT /admin/offers/:id
// DELETE /admin/offers/:id
```

---

## 🔒 Middleware: `verifyAdmin`

ts
CopyEdit

```
function verifyAdmin(req, res, next) {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.status(403).json({ error: "Token required"
});

  try {
    const payload = jwt.verify(token, SECRET);
    req.adminId = payload.adminId;
    next();
  } catch (e) {
    return res.status(401).json({ error: "Invalid token" });
  }
}
```

---

## ✅ Summary

| API | Method | Auth | Description |
|---|---|---|---|
| `/admin/login` | POST | ❌ | Admin login |
| `/admin/offers` | POST | ✅ | Create offer |

| | | | |
|---|---|---|---|
| `/offers?date=YYYY-MM-DD` | GET | ❌ | Publicly accessible offers |
| `/admin/offers/:id` | PUT/DELETE | ✅ | Update/Delete offer (optional) |

---

Would you like me to generate:

- A **React-based Admin Dashboard UI** for these APIs?
- Or next: **menu management APIs**?

Let me know how you'd like to proceed.

**You said:**
Menu management api

**ChatGPT said:**

Perfect — here's the **Menu Management API** for your restaurant app's admin panel.

It allows your admin to:

- ✅ Add new dishes
- ✅ Edit/update dishes
- ✅ Delete dishes
- ✅ List dishes (for both admin & customer menu view)

---

## 🧠 Assumptions

- Backend: Node.js + Express
- Database: PostgreSQL / MySQL / MongoDB (any supported ORM)
- Auth: JWT-based `verifyAdmin` middleware (already provided)

---

## 📦 Menu Item Data Structure (DB table: `menu_items`)

| Field | Type | Example |
|---|---|---|
| id | string (UUID) | `item_1234` |
| name | string | `"Paneer Butter Masala"` |
| description | string | `"Creamy paneer curry with spices"` |
| price | float | `19.99` |
| image_url | string | `"https://img.com/paneer.jpg"` |
| category | string | `"Main Course"` |
| is_available | boolean | `true` |
| created_at | datetime | — |

---

# ✅ API ENDPOINTS

### 1. 🔧 Create Menu Item

**POST** `/admin/menu`

json
CopyEdit

```
// Request body
{
  "name": "Veg Biryani",
  "description": "Spicy and flavorful rice with vegetables",
  "price": 15.99,
  "image_url": "https://cdn.example.com/veg-biryani.jpg",
  "category": "Main Course",
  "is_available": true
}
```

**Sample Code:**

ts
CopyEdit
```ts
app.post('/admin/menu', verifyAdmin, async (req, res) => {
  const item = await db.menu_items.create({ data: req.body });
  res.json({ success: true, item });
});
```

---

## 2. 🖊️ Update Menu Item

**PUT** `/admin/menu/:id`

json
CopyEdit
```json
{
  "name": "Veg Biryani Deluxe",
  "price": 17.99
}
```

**Sample Code:**

ts
CopyEdit
```ts
app.put('/admin/menu/:id', verifyAdmin, async (req, res) => {
  const item = await db.menu_items.update({
    where: { id: req.params.id },
    data: req.body
  });
  res.json({ success: true, item });
});
```

---

## 3. ❌ Delete Menu Item

**DELETE** `/admin/menu/:id`

**Sample Code:**

ts
CopyEdit

```ts
app.delete('/admin/menu/:id', verifyAdmin, async (req, res) => {
  await db.menu_items.delete({ where: { id: req.params.id } });
  res.json({ success: true });
});
```

---

### 4. 📋 List All Menu Items (Admin Panel)

**GET** `/admin/menu`

json
CopyEdit

```json
[
  {
    "id": "item_001",
    "name": "Veg Biryani",
    "category": "Main Course",
    "price": 15.99,
    "is_available": true
  },
  ...
]
```

---

### 5. 🍽️ List Available Menu for Customers

**GET** `/menu`

Filters `is_available = true`

---

## 🔒 Reminder: Middleware `verifyAdmin`

Already provided in previous answers. Add to all `/admin/*` routes to secure them.

---

## 💡 Bonus Ideas

- Add `restaurant_id` if you'll support multiple restaurants
- Add `is_special_offer: boolean` to mark today's special
- Group categories dynamically on frontend: `"Starters"`, `"Main Course"`, `"Desserts"`

---

## ✅ Summary of Endpoints

| Endpoint | Method | Auth | Description |
|---|---|---|---|
| `/admin/menu` | POST | ✅ | Add a new menu item |
| `/admin/menu/:id` | PUT | ✅ | Update menu item |
| `/admin/menu/:id` | DELETE | ✅ | Delete menu item |
| `/admin/menu` | GET | ✅ | List all menu items |
| `/menu` | GET | ❌ | List available items for app |

---

Would you like a **React Admin Panel UI** for this next? Or should we build:

- 🔢 **Order status management** for kitchen/counter
- 📊 **Sales reports API**
- 📱 **User order history / reorder**

Let me know!

**You said:**
Order status management

**ChatGPT said:**

Excellent — let's build the **Order Status Management API** to allow:

- 👨‍🍳 **Kitchen staff** to update order status: *preparing*, *ready*, etc.
- 💵 **Pay counter** to mark payment as *completed*.
- 📱 **Customers** to view order progress in real-time.

---

## ✅ Key Features

1. Update order status:
    - `pending` → `preparing` → `ready` → `completed`
    - or `cancelled`
2. Mark payment status:
    - `unpaid` → `paid`
3. Get status of any order (for customer tracking)

---

## 🗃️ Assumed Order Table

ts
CopyEdit

```
orders {
  id: string,
  user_id: string,
  session_id: string,
  location_type: "table" | "flat" | "house",
  location_id: string,
  status: "pending" | "preparing" | "ready" | "completed" |
"cancelled",
  payment_status: "unpaid" | "paid",
  placed_at: Date,
```

```
  updated_at: Date
}
```

---

## 🔐 Assumptions

- Auth middleware for kitchen/counter roles (e.g., `verifyKitchen`, `verifyCounter`)
- Real-time update is optional but supported via Firebase/WebSocket

---

## 🛠️ API ENDPOINTS

### 1️⃣ Update Order Status (Kitchen Use)

**PUT** `/kitchen/orders/:id/status`

json
CopyEdit

```
// Request body
{
  "status": "preparing"
}
```

ts
CopyEdit

```
app.put('/kitchen/orders/:id/status', verifyKitchen, async (req, res)
=> {
  const { status } = req.body;
  const validStatuses = ['preparing', 'ready', 'completed',
'cancelled'];

  if (!validStatuses.includes(status)) return res.status(400).json({
error: "Invalid status" });

  const order = await db.orders.update({
    where: { id: req.params.id },
```

```
    data: { status, updated_at: new Date() }
  });

  notifyCustomer(order.user_id, `Order #${order.id} is now
${status}`);
  res.json({ success: true, order });
});
```

---

## 2️⃣ Update Payment Status (Pay Counter Use)

**PUT** `/counter/orders/:id/payment`

json
CopyEdit

```
// Request body
{
  "payment_status": "paid"
}
```

ts
CopyEdit

```
app.put('/counter/orders/:id/payment', verifyCounter, async (req, res)
=> {
  const { payment_status } = req.body;
  if (!['paid', 'unpaid'].includes(payment_status)) return
res.status(400).json({ error: "Invalid status" });

  const order = await db.orders.update({
    where: { id: req.params.id },
    data: { payment_status, updated_at: new Date() }
  });

  res.json({ success: true, order });
});
```

---

## 3️⃣ Get Order Status (Customer Use)

**GET** `/orders/:id/status`

json
CopyEdit

```
{
  "status": "preparing",
  "payment_status": "unpaid"
}
```

ts
CopyEdit

```
app.get('/orders/:id/status', async (req, res) => {
  const order = await db.orders.findUnique({
    where: { id: req.params.id },
    select: { status: true, payment_status: true }
  });
  if (!order) return res.status(404).json({ error: "Order not found"
});
  res.json(order);
});
```

---

## 🔔 Optional: Notify Customer in Real-Time

ts
CopyEdit

```
function notifyCustomer(userId, message) {
  const token = getDeviceToken(userId);
  sendFCM(token, {
    title: "Order Update",
    body: message
  });
}
```

---

## ✅ Summary of Endpoints

| Endpoint | Method | Auth | Role | Description |
|---|---|---|---|---|
| `/kitchen/orders/:id/status` | PUT | ✅ `kitchen` | Kitchen | Update food prep status |
| `/counter/orders/:id/payment` | PUT | ✅ `counter` | Pay Counter | Mark order as paid/unpaid |
| `/orders/:id/status` | GET | ❌ | Public/User | View order status |

---

## 🔄 Example Status Flow

plaintext
CopyEdit

```
User places order
→ Kitchen marks as "preparing"
→ Kitchen marks as "ready"
→ Counter marks as "paid"
→ Kitchen/Counter marks as "completed"
```

---

Would you like a **kitchen dashboard UI** or **counter tablet interface** to interact with this API?

Or shall we move to:

- 📱 **Customer order history + reorder**
- 📊 **Sales reports**
- 🧾 **Invoice generation / receipt sharing**?

Let me know your next priority!

**You said:**
Customer order history and reorder

**ChatGPT said:**

Perfect! Let's implement **Customer Order History + Reorder API** to allow users to:

- 📋 **View past orders**
- 🔁 **Reorder** any previous order with 1 click

---

## 🗃️ Assumed Tables

### `orders`

Stores one order record per order

### `order_items`

Stores items inside each order

ts
CopyEdit

```
order_items {
  id: string,
  order_id: string,
  item_id: string,
  item_name: string,
  quantity: number,
  price: number
}
```

---

## 🔐 Requirements

- User must be logged in (via mobile or WhatsApp)
- We'll use `user_id` to fetch and copy orders

---

## ✅ 1. Get Order History

**GET** `/orders/history?user_id=USER123`

**Response:**

json
CopyEdit

```
[
  {
    "order_id": "ORD001",
    "placed_at": "2025-08-06T10:12:00Z",
    "status": "completed",
    "items": [
      { "item_name": "Chicken Biryani", "quantity": 1, "price": 12.5
},
      { "item_name": "Lassi", "quantity": 1, "price": 3.0 }
    ],
    "total": 15.5
  },
  ...
]
```

**Example Code:**

ts
CopyEdit

```
app.get('/orders/history', async (req, res) => {
  const { user_id } = req.query;
  const orders = await db.orders.findMany({
    where: { user_id },
    orderBy: { placed_at: 'desc' },
    include: {
      items: {
        select: { item_name: true, quantity: true, price: true }
      }
    }
  });

  const response = orders.map(order => ({
    order_id: order.id,
```

```
    placed_at: order.placed_at,
    status: order.status,
    items: order.items,
    total: order.items.reduce((sum, item) => sum + item.price *
item.quantity, 0)
  }));

  res.json(response);
});
```

---

## ✅ 2. Reorder API (One-Click Repeat)

**POST** `/orders/reorder`

**Request:**

json
CopyEdit

```
{
  "order_id": "ORD001",
  "user_id": "USER123"
}
```

**What It Does:**

- Fetch the original order + items
- Create a new order with same items for this user

ts
CopyEdit

```
app.post('/orders/reorder', async (req, res) => {
  const { order_id, user_id } = req.body;

  const oldOrder = await db.orders.findUnique({
    where: { id: order_id },
    include: { items: true }
```

```
  });

  if (!oldOrder || oldOrder.user_id !== user_id) {
    return res.status(403).json({ error: "Access denied" });
  }

  const newOrder = await db.orders.create({
    data: {
      user_id,
      session_id: oldOrder.session_id, // optional
      location_id: oldOrder.location_id,
      location_type: oldOrder.location_type,
      status: 'pending',
      payment_status: 'unpaid',
      placed_at: new Date(),
      items: {
        create: oldOrder.items.map(item => ({
          item_id: item.item_id,
          item_name: item.item_name,
          quantity: item.quantity,
          price: item.price
        }))
      }
    }
  });

  notifyKitchen(newOrder.id);
  res.json({ success: true, new_order_id: newOrder.id });
});
```

---

## 🔔 Optional: Notify Kitchen

ts
CopyEdit

```
function notifyKitchen(orderId) {
  sendFCM(kitchenDeviceToken, {
    title: "New Reorder",
```

```
    body: `Reorder placed: #${orderId}`
  });
}
```

---

## ✅ Summary of Endpoints

| Endpoint | Method | Purpose |
|---|---|---|
| `/orders/history` | GET | View past orders |
| `/orders/reorder` | POST | Reorder previous items |

---

Would you like to add:

- 🔍 **Filter by date or item**
- 🧾 **Invoice view/download for each order**
- 🧠 **AI-suggested reorders or favorite items**?

Or move to:

- 📊 Admin analytics (top customers, top items)
- 👨‍🍳 Multi-kitchen support

Let me know your next priority!

**You said:**
From the above your suggestions what are provided and not yet

**ChatGPT said:**

Here's a clear **summary checklist** of what I've provided so far and what's still pending from my earlier suggestions for your restaurant app project:

---

## ✅ Already Provided

| Feature / Module | Status | Notes |
|---|---|---|
| QR Code Scan Flow | ✅ Done | Frontend + backend session handling |
| Order Placement API | ✅ Done | Order creation + notify kitchen/counter |
| Notification Logic | ✅ Done | Real-time & daily offers (Firebase based) |
| QR Code Generation Script | ✅ Done | Node.js script for unlimited QR codes |
| Admin Panel API (Daily Offers) | ✅ Done | Admin login + create/list offers |
| Menu Management API | ✅ Done | CRUD for menu items with admin auth |
| Order Status Management | ✅ Done | Update order/payment status + notifications |
| Customer Order History + Reorder | ✅ Done | View past orders + 1-click reorder |

---

## ➡️ Not Yet Provided (Pending / Optional Next Steps)

| Feature / Module | Description |
|---|---|
| Customer Login Flow (Phone/WhatsApp) | Auth flow with OTP or WhatsApp login |
| Admin Panel UI (React or Web) | Admin dashboard UI for offers, menu, orders |
| Kitchen / Pay Counter Dashboard UI | Live order management interfaces |
| Order Status Real-Time Updates UI | Real-time notifications with WebSocket UI |
| Customer Location Tracking (dynamic) | Capture GPS/location if ordering outside fixed QR |
| Push Notification Opt-in + Token Storage | Mobile/web app logic for saving FCM tokens |
| Multi-Restaurant / Branch Support | Extend APIs & DB for multiple outlets |
| Sales Reports & Analytics API | Data on top-selling items, revenue, etc. |
| Invoice/Receipt Generation | PDF/email invoices for orders |
| AI-based Recommendations | Suggest orders based on past behavior |

---

If you want, I can help you build any of these next. Just let me know your priority!

```
// Customer Login Flow with Phone/WhatsApp Authentication
const express = require('express');
const jwt = require('jsonwebtoken');
const twilio = require('twilio');
const app = express();

// Twilio configuration
const accountSid = process.env.TWILIO_ACCOUNT_SID;
const authToken = process.env.TWILIO_AUTH_TOKEN;
const twilioClient = twilio(accountSid, authToken);

const JWT_SECRET = process.env.JWT_SECRET || 'your_jwt_secret';

// In-memory OTP storage (use Redis in production)
const otpStore = new Map();

// 1. Send OTP to Phone Number
app.post('/auth/send-otp', async (req, res) => {
  const { phone_number, country_code = '+91' } = req.body;

  if (!phone_number) {
    return res.status(400).json({ error: 'Phone number required' });
  }

  const fullPhoneNumber = `${country_code}${phone_number}`;
  const otp = Math.floor(100000 + Math.random() * 900000); // 6-digit OTP

  try {
    // Send OTP via SMS
    await twilioClient.messages.create({
      body: `Your restaurant app OTP is: ${otp}. Valid for 5 minutes.`,
      from: process.env.TWILIO_PHONE_NUMBER,
      to: fullPhoneNumber
    });

    // Store OTP with expiration (5 minutes)
    otpStore.set(fullPhoneNumber, {
      otp,
      expires: Date.now() + 5 * 60 * 1000
    });

    res.json({
```

```
      success: true,
      message: 'OTP sent successfully',
      phone_number: fullPhoneNumber
    });
  } catch (error) {
    console.error('SMS send error:', error);
    res.status(500).json({ error: 'Failed to send OTP' });
  }
});

// 2. Verify OTP and Login
app.post('/auth/verify-otp', async (req, res) => {
  const { phone_number, otp, device_token } = req.body;

  if (!phone_number || !otp) {
    return res.status(400).json({ error: 'Phone number and OTP required' });
  }

  const storedOtpData = otpStore.get(phone_number);

  if (!storedOtpData) {
    return res.status(400).json({ error: 'OTP not found or expired' });
  }

  if (Date.now() > storedOtpData.expires) {
    otpStore.delete(phone_number);
    return res.status(400).json({ error: 'OTP expired' });
  }

  if (parseInt(otp) !== storedOtpData.otp) {
    return res.status(400).json({ error: 'Invalid OTP' });
  }

  // OTP is valid, create or find user
  try {
    let user = await db.users.findUnique({
      where: { phone_number }
    });

    if (!user) {
      user = await db.users.create({
        data: {
          phone_number,
          is_permanent: true,
          device_token,
          joined_at: new Date()
```

```
      }
    });
  } else {
    // Update device token if provided
    if (device_token) {
      user = await db.users.update({
        where: { id: user.id },
        data: { device_token }
      });
    }
  }

  // Generate JWT token
  const token = jwt.sign(
    { userId: user.id, phone_number: user.phone_number },
    JWT_SECRET,
    { expiresIn: '30d' }
  );

  // Clean up OTP
  otpStore.delete(phone_number);

  res.json({
    success: true,
    token,
    user: {
      id: user.id,
      phone_number: user.phone_number,
      joined_at: user.joined_at
    }
  });
 } catch (error) {
  console.error('Login error:', error);
  res.status(500).json({ error: 'Login failed' });
 }
});

// 3. WhatsApp Login (Alternative)
app.post('/auth/whatsapp-login', async (req, res) => {
 const { whatsapp_number, device_token } = req.body;

 if (!whatsapp_number) {
  return res.status(400).json({ error: 'WhatsApp number required' });
 }

 try {
```

```
    let user = await db.users.findUnique({
      where: { whatsapp_number }
    });

    if (!user) {
      user = await db.users.create({
        data: {
          whatsapp_number,
          is_permanent: true,
          device_token,
          joined_at: new Date()
        }
      });
    } else {
      if (device_token) {
        user = await db.users.update({
          where: { id: user.id },
          data: { device_token }
        });
      }
    }

    const token = jwt.sign(
      { userId: user.id, whatsapp_number: user.whatsapp_number },
      JWT_SECRET,
      { expiresIn: '30d' }
    );

    res.json({
      success: true,
      token,
      user: {
        id: user.id,
        whatsapp_number: user.whatsapp_number,
        joined_at: user.joined_at
      }
    });
  } catch (error) {
    console.error('WhatsApp login error:', error);
    res.status(500).json({ error: 'WhatsApp login failed' });
  }
});

// 4. Verify JWT Token Middleware
function verifyUser(req, res, next) {
  const token = req.headers.authorization?.split(" ")[1];
```

```
  if (!token) {
    return res.status(403).json({ error: "Token required" });
  }

  try {
    const payload = jwt.verify(token, JWT_SECRET);
    req.userId = payload.userId;
    req.userPhone = payload.phone_number || payload.whatsapp_number;
    next();
  } catch (error) {
    return res.status(401).json({ error: "Invalid token" });
  }
}

// 5. Get User Profile
app.get('/auth/profile', verifyUser, async (req, res) => {
  try {
    const user = await db.users.findUnique({
      where: { id: req.userId },
      select: {
        id: true,
        phone_number: true,
        whatsapp_number: true,
        joined_at: true,
        is_permanent: true
      }
    });

    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }

    res.json(user);
  } catch (error) {
    console.error('Profile fetch error:', error);
    res.status(500).json({ error: 'Failed to fetch profile' });
  }
});

// 6. Save/Update Device Token
app.post('/auth/device-token', verifyUser, async (req, res) => {
  const { device_token } = req.body;

  if (!device_token) {
    return res.status(400).json({ error: 'Device token required' });
```

```
  }

  try {
    await db.users.update({
      where: { id: req.userId },
      data: { device_token }
    });

    res.json({ success: true, message: 'Device token updated' });
  } catch (error) {
    console.error('Token update error:', error);
    res.status(500).json({ error: 'Failed to update device token' });
  }
});

module.exports = { verifyUser };

import React, { useState, useEffect } from 'react';
import { Bell, Menu as MenuIcon, Plus, Edit, Trash2, Users, TrendingUp, ShoppingBag } from
'lucide-react';

const AdminDashboard = () => {
  const [activeTab, setActiveTab] = useState('offers');
  const [offers, setOffers] = useState([]);
  const [menuItems, setMenuItems] = useState([]);
  const [orders, setOrders] = useState([]);
  const [showAddForm, setShowAddForm] = useState(false);
  const [editingItem, setEditingItem] = useState(null);

  // Mock data - replace with actual API calls
  useEffect(() => {
    setOffers([
      { id: '1', title: 'Paneer Butter Masala 20% Off', message: 'Only for today!', active_date:
'2025-08-06' },
      { id: '2', title: 'Buy 1 Get 1 Free Biryani', message: 'Weekend special', active_date: '2025-08-07' }
    ]);

    setMenuItems([
      { id: '1', name: 'Chicken Biryani', category: 'Main Course', price: 12.5, is_available: true },
      { id: '2', name: 'Paneer Butter Masala', category: 'Main Course', price: 10.0, is_available: true },
      { id: '3', name: 'Mango Lassi', category: 'Beverages', price: 3.0, is_available: false }
    ]);

    setOrders([
      { id: 'ORD001', location: 'Table-1', status: 'preparing', total: 15.5, time: '10:30 AM' },
      { id: 'ORD002', location: 'Flat-A201', status: 'ready', total: 22.0, time: '10:45 AM' },
```

```
    { id: 'ORD003', location: 'House-101', status: 'completed', total: 18.5, time: '10:15 AM' }
  ]);
 }, []);

 const OfferForm = ({ offer = null, onSave, onCancel }) => {
  const [formData, setFormData] = useState({
    title: offer?.title || '',
    message: offer?.message || '',
    active_date: offer?.active_date || new Date().toISOString().split('T')[0]
  });

  const handleSubmit = (e) => {
    e.preventDefault();
    onSave(formData);
  };

  return (
    <div className="bg-white p-6 rounded-lg shadow-md">
      <h3 className="text-lg font-semibold mb-4">
        {offer ? 'Edit Offer' : 'Create New Offer'}
      </h3>
      <form onSubmit={handleSubmit} className="space-y-4">
        <div>
          <label className="block text-sm font-medium text-gray-700 mb-1">Title</label>
          <input
            type="text"
            className="w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500"
            value={formData.title}
            onChange={(e) => setFormData({...formData, title: e.target.value})}
            required
          />
        </div>
        <div>
          <label className="block text-sm font-medium text-gray-700 mb-1">Message</label>
          <textarea
            className="w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500"
            rows="3"
            value={formData.message}
            onChange={(e) => setFormData({...formData, message: e.target.value})}
            required
          />
        </div>
        <div>
          <label className="block text-sm font-medium text-gray-700 mb-1">Active Date</label>
```

```
            <input
              type="date"
              className="w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500"
              value={formData.active_date}
              onChange={(e) => setFormData({...formData, active_date: e.target.value})}
              required
            />
          </div>
          <div className="flex gap-2">
            <button
              type="submit"
              className="bg-blue-600 text-white px-4 py-2 rounded-md hover:bg-blue-700"
            >
              {offer ? 'Update' : 'Create'} Offer
            </button>
            <button
              type="button"
              onClick={onCancel}
              className="bg-gray-300 text-gray-700 px-4 py-2 rounded-md hover:bg-gray-400"
            >
              Cancel
            </button>
          </div>
        </form>
      </div>
    );
  };

  const MenuForm = ({ item = null, onSave, onCancel }) => {
    const [formData, setFormData] = useState({
      name: item?.name || '',
      category: item?.category || 'Main Course',
      price: item?.price || '',
      description: item?.description || '',
      is_available: item?.is_available !== undefined ? item.is_available : true
    });

    const handleSubmit = (e) => {
      e.preventDefault();
      onSave(formData);
    };

    return (
      <div className="bg-white p-6 rounded-lg shadow-md">
        <h3 className="text-lg font-semibold mb-4">
```

```
          {item ? 'Edit Menu Item' : 'Add New Menu Item'}
        </h3>
        <form onSubmit={handleSubmit} className="space-y-4">
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">Name</label>
            <input
              type="text"
              className="w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500"
              value={formData.name}
              onChange={(e) => setFormData({...formData, name: e.target.value})}
              required
            />
          </div>
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">Category</label>
            <select
              className="w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500"
              value={formData.category}
              onChange={(e) => setFormData({...formData, category: e.target.value})}
            >
              <option>Starters</option>
              <option>Main Course</option>
              <option>Beverages</option>
              <option>Desserts</option>
            </select>
          </div>
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">Price</label>
            <input
              type="number"
              step="0.01"
              className="w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500"
              value={formData.price}
              onChange={(e) => setFormData({...formData, price: parseFloat(e.target.value)})}
              required
            />
          </div>
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">Description</label>
            <textarea
              className="w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500"
              rows="3"
```

```
            value={formData.description}
            onChange={(e) => setFormData({...formData, description: e.target.value})}
          />
        </div>
        <div>
          <label className="flex items-center">
            <input
              type="checkbox"
              className="mr-2"
              checked={formData.is_available}
              onChange={(e) => setFormData({...formData, is_available: e.target.checked})}
            />
            Available
          </label>
        </div>
        <div className="flex gap-2">
          <button
            type="submit"
            className="bg-blue-600 text-white px-4 py-2 rounded-md hover:bg-blue-700"
          >
            {item ? 'Update' : 'Add'} Item
          </button>
          <button
            type="button"
            onClick={onCancel}
            className="bg-gray-300 text-gray-700 px-4 py-2 rounded-md hover:bg-gray-400"
          >
            Cancel
          </button>
        </div>
      </form>
    </div>
  );
};

const handleSaveOffer = (formData) => {
  if (editingItem) {
    setOffers(offers.map(offer =>
      offer.id === editingItem.id ? {...offer, ...formData} : offer
    ));
  } else {
    setOffers([...offers, {...formData, id: Date.now().toString()}]);
  }
  setShowAddForm(false);
  setEditingItem(null);
};
```

```
  const handleSaveMenuItem = (formData) => {
    if (editingItem) {
      setMenuItems(menuItems.map(item =>
        item.id === editingItem.id ? {...item, ...formData} : item
      ));
    } else {
      setMenuItems([...menuItems, {...formData, id: Date.now().toString()}]);
    }
    setShowAddForm(false);
    setEditingItem(null);
  };

  const handleDelete = (id, type) => {
    if (type === 'offer') {
      setOffers(offers.filter(offer => offer.id !== id));
    } else if (type === 'menu') {
      setMenuItems(menuItems.filter(item => item.id !== id));
    }
  };

  return (
    <div className="min-h-screen bg-gray-100">
      {/* Header */}
      <header className="bg-white shadow-sm border-b">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
          <div className="flex justify-between items-center py-4">
            <h1 className="text-2xl font-bold text-gray-900">Restaurant Admin</h1>
            <div className="flex items-center space-x-4">
              <Bell className="w-6 h-6 text-gray-400" />
              <div className="w-8 h-8 bg-blue-600 rounded-full"></div>
            </div>
          </div>
        </div>
      </header>

      {/* Navigation Tabs */}
      <div className="bg-white border-b">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
          <nav className="flex space-x-8">
            {[
              { id: 'offers', label: 'Daily Offers', icon: Bell },
              { id: 'menu', label: 'Menu Management', icon: MenuIcon },
              { id: 'orders', label: 'Orders', icon: ShoppingBag },
              { id: 'analytics', label: 'Analytics', icon: TrendingUp }
            ].map(({ id, label, icon: Icon }) => (
```

```
            <button
              key={id}
              onClick={() => setActiveTab(id)}
              className={`flex items-center space-x-2 py-4 px-1 border-b-2 font-medium text-sm ${
                activeTab === id
                  ? 'border-blue-500 text-blue-600'
                  : 'border-transparent text-gray-500 hover:text-gray-700'
              }`}
            >
              <Icon className="w-4 h-4" />
              <span>{label}</span>
            </button>
          ))}
        </nav>
      </div>
    </div>

    {/* Main Content */}
    <main className="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">
      {/* Offers Tab */}
      {activeTab === 'offers' && (
        <div className="px-4 py-6 sm:px-0">
          <div className="flex justify-between items-center mb-6">
            <h2 className="text-lg font-medium text-gray-900">Daily Offers</h2>
            <button
              onClick={() => setShowAddForm(true)}
              className="bg-blue-600 text-white px-4 py-2 rounded-md hover:bg-blue-700 flex
items-center space-x-2"
            >
              <Plus className="w-4 h-4" />
              <span>Add Offer</span>
            </button>
          </div>

          {showAddForm && (
            <div className="mb-6">
              <OfferForm
                offer={editingItem}
                onSave={handleSaveOffer}
                onCancel={() => {
                  setShowAddForm(false);
                  setEditingItem(null);
                }}
              />
            </div>
          )}
```

```
          <div className="bg-white shadow overflow-hidden sm:rounded-md">
            <ul className="divide-y divide-gray-200">
              {offers.map((offer) => (
                <li key={offer.id} className="px-6 py-4">
                  <div className="flex items-center justify-between">
                    <div className="flex-1">
                      <h3 className="text-lg font-medium text-gray-900">{offer.title}</h3>
                      <p className="text-sm text-gray-500 mt-1">{offer.message}</p>
                      <p className="text-xs text-gray-400 mt-1">Active: {offer.active_date}</p>
                    </div>
                    <div className="flex space-x-2">
                      <button
                        onClick={() => {
                          setEditingItem(offer);
                          setShowAddForm(true);
                        }}
                        className="text-blue-600 hover:text-blue-800"
                      >
                        <Edit className="w-4 h-4" />
                      </button>
                      <button
                        onClick={() => handleDelete(offer.id, 'offer')}
                        className="text-red-600 hover:text-red-800"
                      >
                        <Trash2 className="w-4 h-4" />
                      </button>
                    </div>
                  </div>
                </li>
              ))}
            </ul>
          </div>
        </div>
      )}

      {/* Menu Tab */}
      {activeTab === 'menu' && (
        <div className="px-4 py-6 sm:px-0">
          <div className="flex justify-between items-center mb-6">
            <h2 className="text-lg font-medium text-gray-900">Menu Management</h2>
            <button
              onClick={() => setShowAddForm(true)}
              className="bg-blue-600 text-white px-4 py-2 rounded-md hover:bg-blue-700 flex
items-center space-x-2"
            >
```

```
              <Plus className="w-4 h-4" />
              <span>Add Menu Item</span>
            </button>
          </div>

          {showAddForm && (
            <div className="mb-6">
              <MenuForm
                item={editingItem}
                onSave={handleSaveMenuItem}
                onCancel={() => {
                  setShowAddForm(false);
                  setEditingItem(null);
                }}
              />
            </div>
          )}

          <div className="bg-white shadow overflow-hidden sm:rounded-md">
            <table className="min-w-full divide-y divide-gray-200">
              <thead className="bg-gray-50">
                <tr>
                  <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase
tracking-wider">Name</th>
                  <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase
tracking-wider">Category</th>
                  <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase
tracking-wider">Price</th>
                  <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase
tracking-wider">Status</th>
                  <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase
tracking-wider">Actions</th>
                </tr>
              </thead>
              <tbody className="bg-white divide-y divide-gray-200">
                {menuItems.map((item) => (
                  <tr key={item.id}>
                    <td className="px-6 py-4 whitespace-nowrap text-sm font-medium
text-gray-900">{item.name}</td>
                    <td className="px-6 py-4 whitespace-nowrap text-sm
text-gray-500">{item.category}</td>
                    <td className="px-6 py-4 whitespace-nowrap text-sm
text-gray-500">${item.price}</td>
                    <td className="px-6 py-4 whitespace-nowrap">
                      <span className={`px-2 inline-flex text-xs leading-5 font-semibold rounded-full ${
                        item.is_available ? 'bg-green-100 text-green-800' : 'bg-red-100 text-red-800'
```

```
                    }`}>
                      {item.is_available ? 'Available' : 'Unavailable'}
                    </span>
                  </td>
                  <td className="px-6 py-4 whitespace-nowrap text-sm font-medium">
                    <div className="flex space-x-2">
                      <button
                        onClick={() => {
                          setEditingItem(item);
                          setShowAddForm(true);
                        }}
                        className="text-blue-600 hover:text-blue-800"
                      >
                        <Edit className="w-4 h-4" />
                      </button>
                      <button
                        onClick={() => handleDelete(item.id, 'menu')}
                        className="text-red-600 hover:text-red-800"
                      >
                        <Trash2 className="w-4 h-4" />
                      </button>
                    </div>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      </div>
    )}

    {/* Orders Tab */}
    {activeTab === 'orders' && (
      <div className="px-4 py-6 sm:px-0">
        <h2 className="text-lg font-medium text-gray-900 mb-6">Recent Orders</h2>
        <div className="bg-white shadow overflow-hidden sm:rounded-md">
          <ul className="divide-y divide-gray-200">
            {orders.map((order) => (
              <li key={order.id} className="px-6 py-4">
                <div className="flex items-center justify-between">
                  <div className="flex-1">
                    <div className="flex items-center space-x-4">
                      <h3 className="text-lg font-medium text-gray-900">#{order.id}</h3>
                      <span className="text-sm text-gray-500">{order.location}</span>
                      <span className={`px-2 inline-flex text-xs leading-5 font-semibold rounded-full ${
                        order.status === 'completed' ? 'bg-green-100 text-green-800' :
```

```
                    order.status === 'ready' ? 'bg-yellow-100 text-yellow-800' :
                    'bg-blue-100 text-blue-800'
                  }`}>
                    {order.status}
                  </span>
                </div>
                <div className="mt-2 flex items-center space-x-4">
                  <span className="text-sm text-gray-500">${order.total}</span>
                  <span className="text-sm text-gray-500">{order.time}</span>
                </div>
              </div>
            </div>
          </li>
        ))}
      </ul>
    </div>
  </div>
))}

{/* Analytics Tab */}
{activeTab === 'analytics' && (
  <div className="px-4 py-6 sm:px-0">
    <h2 className="text-lg font-medium text-gray-900 mb-6">Analytics Dashboard</h2>
    <div className="grid grid-cols-1 gap-6 sm:grid-cols-2 lg:grid-cols-4">
      <div className="bg-white overflow-hidden shadow rounded-lg">
        <div className="p-5">
          <div className="flex items-center">
            <div className="flex-shrink-0">
              <Users className="h-6 w-6 text-gray-400" />
            </div>
            <div className="ml-5 w-0 flex-1">
              <dl>
                <dt className="text-sm font-medium text-gray-500 truncate">Total Orders</dt>
                <dd className="text-lg font-medium text-gray-900">432</dd>
              </dl>
            </div>
          </div>
        </div>
      </div>
      <div className="bg-white overflow-hidden shadow rounded-lg">
        <div className="p-5">
          <div className="flex items-center">
            <div className="flex-shrink-0">
              <TrendingUp className="h-6 w-6 text-gray-400" />
            </div>
            <div className="ml-5 w-0 flex-1">
```

```
                  <dl>
                    <dt className="text-sm font-medium text-gray-500 truncate">Revenue</dt>
                    <dd className="text-lg font-medium text-gray-900">$12,480</dd>
                  </dl>
                </div>
              </div>
            </div>
          </div>
          <div className="bg-white overflow-hidden shadow rounded-lg">
            <div className="p-5">
              <div className="flex items-center">
                <div className="flex-shrink-0">
                  <MenuIcon className="h-6 w-6 text-gray-400" />
                </div>
                <div className="ml-5 w-0 flex-1">
                  <dl>
                    <dt className="text-sm font-medium text-gray-500 truncate">Menu Items</dt>
                    <dd className="text-lg font-medium text-gray-900">{menuItems.length}</dd>
                  </dl>
                </div>
              </div>
            </div>
          </div>
        </div>
      ))}
    </main>
  </div>
 );
};

export default AdminDashboard; truncate">Total Customers</dt>
                    <dd className="text-lg font-medium text-gray-900">1,245</dd>
                  </dl>
                </div>
              </div>
            </div>
          </div>
          <div className="bg-white overflow-hidden shadow rounded-lg">
            <div className="p-5">
              <div className="flex items-center">
                <div className="flex-shrink-0">
                  <ShoppingBag className="h-6 w-6 text-gray-400" />
                </div>
                <div className="ml-5 w-0 flex-1">
                  <dl>
```

```
                    <dt className="text-sm font-medium text-gray-500
```

## Kitchen Dashboard API

```
import React, { useState, useEffect } from 'react';
import { Clock, MapPin, CheckCircle, AlertCircle, Users, Utensils } from 'lucide-react';

const KitchenDashboard = () => {
  const [orders, setOrders] = useState([]);
  const [stats, setStats] = useState({
    pending: 0,
    preparing: 0,
    ready: 0,
    avgPrepTime: 0
  });

  // Mock data - replace with real-time WebSocket connection
  useEffect(() => {
    const mockOrders = [
      {
        id: 'ORD001',
        location_type: 'table',
        location_id: 'TBL-001',
        status: 'pending',
        items: [
          { name: 'Chicken Biryani', quantity: 2, notes: 'Extra spicy' },
          { name: 'Raita', quantity: 1, notes: '' }
        ],
        placed_at: '2025-08-06T10:30:00Z',
        total: 25.0,
        customer_notes: 'No onions please'
      },
      {
        id: 'ORD002',
        location_type: 'flat',
        location_id: 'FLAT-A201',
        status: 'preparing',
        items: [
          { name: 'Paneer Butter Masala', quantity: 1, notes: 'Medium spice' },
          { name: 'Naan', quantity: 2, notes: 'Butter naan' },
          { name: 'Dal Tadka', quantity: 1, notes: '' }
        ],
        placed_at: '2025-08-06T10:15:00Z',
        total: 18.5,
        customer_notes: 'Deliver to 2nd floor'
      },
      {
```

```
      id: 'ORD003',
      location_type: 'house',
      location_id: 'HSE-101',
      status: 'preparing',
      items: [
        { name: 'Mutton Curry', quantity: 1, notes: 'Well cooked' },
        { name: 'Rice', quantity: 1, notes: 'Basmati rice' }
      ],
      placed_at: '2025-08-06T10:00:00Z',
      total: 22.0,
      customer_notes: 'Call before delivery'
    },
    {
      id: 'ORD004',
      location_type: 'table',
      location_id: 'TBL-005',
      status: 'ready',
      items: [
        { name: 'Veg Thali', quantity: 1, notes: '' }
      ],
      placed_at: '2025-08-06T09:45:00Z',
      total: 12.0,
      customer_notes: ''
    }
  ];

  setOrders(mockOrders);

  // Calculate stats
  const pending = mockOrders.filter(o => o.status === 'pending').length;
  const preparing = mockOrders.filter(o => o.status === 'preparing').length;
  const ready = mockOrders.filter(o => o.status === 'ready').length;

  setStats({ pending, preparing, ready, avgPrepTime: 18 });
}, []);

const updateOrderStatus = async (orderId, newStatus) => {
  try {
    // API call to update status
    const response = await fetch(`/kitchen/orders/${orderId}/status`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${localStorage.getItem('kitchen_token')}`
      },
      body: JSON.stringify({ status: newStatus })
    });

    if (response.ok) {
```

```
        setOrders(orders.map(order =>
          order.id === orderId ? { ...order, status: newStatus } : order
        ));
      }
    } catch (error) {
      console.error('Failed to update order status:', error);
    }
  };

  const formatTime = (timestamp) => {
    const date = new Date(timestamp);
    const now = new Date();
    const diff = Math.floor((now - date) / (1000 * 60)); // minutes

    if (diff < 60) {
      return `${diff}m ago`;
    } else {
      return `${Math.floor(diff / 60)}h ${diff % 60}m ago`;
    }
  };

  const getStatusColor = (status) => {
    switch (status) {
      case 'pending': return 'bg-red-100 text-red-800 border-red-200';
      case 'preparing': return 'bg-yellow-100 text-yellow-800 border-yellow-200';
      case 'ready': return 'bg-green-100 text-green-800 border-green-200';
      default: return 'bg-gray-100 text-gray-800 border-gray-200';
    }
  };

  const getLocationIcon = (type) => {
    switch (type) {
      case 'table': return '🪑';
      case 'flat': return '🏢';
      case 'house': return '🏠';
      default: return '📍';
    }
  };

  const OrderCard = ({ order }) => (
    <div className={`bg-white rounded-lg shadow-md p-6 border-l-4 ${
      order.status === 'pending' ? 'border-l-red-500' :
      order.status === 'preparing' ? 'border-l-yellow-500' :
      'border-l-green-500'
    }`}>
      <div className="flex justify-between items-start mb-4">
        <div className="flex items-center space-x-3">
          <h3 className="text-xl font-bold text-gray-900">#{order.id}</h3>
```

```
          <span className={`px-3 py-1 rounded-full text-sm font-medium border
${getStatusColor(order.status)}`}>
            {order.status.toUpperCase()}
          </span>
        </div>
        <div className="text-right">
          <div className="flex items-center text-gray-500 mb-1">
            <Clock className="w-4 h-4 mr-1" />
            <span className="text-sm">{formatTime(order.placed_at)}</span>
          </div>
          <div className="text-lg font-bold text-gray-900">${order.total}</div>
        </div>
      </div>

      <div className="flex items-center text-gray-600 mb-4">
        <span className="mr-2">{getLocationIcon(order.location_type)}</span>
        <MapPin className="w-4 h-4 mr-1" />
        <span className="font-medium">{order.location_id}</span>
      </div>

      <div className="space-y-2 mb-4">
        {order.items.map((item, index) => (
          <div key={index} className="flex justify-between items-center py-2 border-b border-gray-100
last:border-b-0">
            <div className="flex-1">
              <div className="flex items-center space-x-2">
                <span className="font-medium text-gray-900">{item.name}</span>
                <span className="bg-blue-100 text-blue-800 text-xs px-2 py-1 rounded-full">
                  x{item.quantity}
                </span>
              </div>
              {item.notes && (
                <p className="text-sm text-orange-600 mt-1">📝 {item.notes}</p>
              )}
            </div>
          </div>
        ))}
      </div>

      {order.customer_notes && (
        <div className="bg-blue-50 p-3 rounded-md mb-4">
          <p className="text-sm text-blue-800">
            <strong>Customer Notes:</strong> {order.customer_notes}
          </p>
        </div>
      )}

      <div className="flex space-x-2">
        {order.status === 'pending' && (
```

```
          <button
            onClick={() => updateOrderStatus(order.id, 'preparing')}
            className="flex-1 bg-yellow-600 text-white px-4 py-2 rounded-md hover:bg-yellow-700
transition-colors flex items-center justify-center"
          >
            <Utensils className="w-4 h-4 mr-2" />
            Start Preparing
          </button>
        )}

        {order.status === 'preparing' && (
          <button
            onClick={() => updateOrderStatus(order.id, 'ready')}
            className="flex-1 bg-green-600 text-white px-4 py-2 rounded-md hover:bg-green-700
transition-colors flex items-center justify-center"
          >
            <CheckCircle className="w-4 h-4 mr-2" />
            Mark Ready
          </button>
        )}

        {order.status === 'ready' && (
          <div className="flex-1 bg-green-100 text-green-800 px-4 py-2 rounded-md text-center
font-medium">
            ✅ Ready for Pickup/Delivery
          </div>
        )}
      </div>
    </div>
  );

  return (
    <div className="min-h-screen bg-gray-100">
      {/* Header */}
      <header className="bg-white shadow-sm">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
          <div className="flex justify-between items-center py-6">
            <div>
              <h1 className="text-3xl font-bold text-gray-900">Kitchen Dashboard</h1>
              <p className="text-gray-600 mt-1">Manage incoming orders and track preparation status</p>
            </div>
            <div className="text-right">
              <div className="text-sm text-gray-500">Current Time</div>
              <div className="text-lg font-semibold text-gray-900">
                {new Date().toLocaleTimeString()}
              </div>
            </div>
          </div>
        </div>
```

```
      </header>

      {/* Stats Cards */}
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-6">
        <div className="grid grid-cols-1 md:grid-cols-4 gap-6 mb-8">
          <div className="bg-white rounded-lg shadow p-6">
            <div className="flex items-center">
              <div className="p-3 rounded-full bg-red-100">
                <AlertCircle className="w-6 h-6 text-red-600" />
              </div>
              <div className="ml-4">
                <p className="text-sm font-medium text-gray-500">Pending Orders</p>
                <p className="text-2xl font-bold text-gray-900">{stats.pending}</p>
              </div>
            </div>
          </div>

          <div className="bg-white rounded-lg shadow p-6">
            <div className="flex items-center">
              <div className="p-3 rounded-full bg-yellow-100">
                <Utensils className="w-6 h-6 text-yellow-600" />
              </div>
              <div className="ml-4">
                <p className="text-sm font-medium text-gray-500">Preparing</p>
                <p className="text-2xl font-bold text-gray-900">{stats.preparing}</p>
              </div>
            </div>
          </div>

          <div className="bg-white rounded-lg shadow p-6">
            <div className="flex items-center">
              <div className="p-3 rounded-full bg-green-100">
                <CheckCircle className="w-6 h-6 text-green-600" />
              </div>
              <div className="ml-4">
                <p className="text-sm font-medium text-gray-500">Ready</p>
                <p className="text-2xl font-bold text-gray-900">{stats.ready}</p>
              </div>
            </div>
          </div>

          <div className="bg-white rounded-lg shadow p-6">
            <div className="flex items-center">
              <div className="p-3 rounded-full bg-blue-100">
                <Clock className="w-6 h-6 text-blue-600" />
              </div>
              <div className="ml-4">
                <p className="text-sm font-medium text-gray-500">Avg Prep Time</p>
                <p className="text-2xl font-bold text-gray-900">{stats.avgPrepTime}min</p>
```

```
            </div>
          </div>
        </div>
      </div>

      {/* Orders Grid */}
      <div className="space-y-6">
        <h2 className="text-xl font-semibold text-gray-900">Active Orders</h2>

        {/* Pending Orders */}
        {orders.filter(order => order.status === 'pending').length > 0 && (
          <div>
            <h3 className="text-lg font-medium text-red-700 mb-4 flex items-center">
              <AlertCircle className="w-5 h-5 mr-2" />
              Pending Orders - Needs Immediate Attention
            </h3>
            <div className="grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-6">
              {orders.filter(order => order.status === 'pending').map(order => (
                <OrderCard key={order.id} order={order} />
              ))}
            </div>
          </div>
        )}

        {/* Preparing Orders */}
        {orders.filter(order => order.status === 'preparing').length > 0 && (
          <div>
            <h3 className="text-lg font-medium text-yellow-700 mb-4 flex items-center">
              <Utensils className="w-5 h-5 mr-2" />
              Currently Preparing
            </h3>
            <div className="grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-6">
              {orders.filter(order => order.status === 'preparing').map(order => (
                <OrderCard key={order.id} order={order} />
              ))}
            </div>
          </div>
        )}

        {/* Ready Orders */}
        {orders.filter(order => order.status === 'ready').length > 0 && (
          <div>
            <h3 className="text-lg font-medium text-green-700 mb-4 flex items-center">
              <CheckCircle className="w-5 h-5 mr-2" />
              Ready for Pickup/Delivery
            </h3>
            <div className="grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-6">
              {orders.filter(order => order.status === 'ready').map(order => (
                <OrderCard key={order.id} order={order} />
```

```
            ))}
          </div>
        </div>
      ))}

      {orders.length === 0 && (
        <div className="text-center py-12">
          <Utensils className="w-12 h-12 text-gray-400 mx-auto mb-4" />
          <h3 className="text-lg font-medium text-gray-900 mb-2">No Active Orders</h3>
          <p className="text-gray-500">All orders are completed. Great job!</p>
        </div>
      )}
    </div>
  </div>
 );
};

export default KitchenDashboard;
```

## Sales reports and analytics API

```
// Sales Reports & Analytics API
const express = require('express');
const { verifyAdmin } = require('./auth'); // From previous admin auth
const app = express();

// 1. Daily Sales Summary
app.get('/admin/analytics/daily-sales', verifyAdmin, async (req, res) => {
 const { date = new Date().toISOString().split('T')[0] } = req.query;

 try {
  const startDate = new Date(`${date}T00:00:00.000Z`);
  const endDate = new Date(`${date}T23:59:59.999Z`);

  // Get orders for the day
  const orders = await db.orders.findMany({
   where: {
    placed_at: {
     gte: startDate,
     lte: endDate
    },
    status: 'completed'
   },
   include: {
    items: {
     include: {
```

```
          menu_item: true
        }
      }
    }
  });

  // Calculate metrics
  const totalOrders = orders.length;
  const totalRevenue = orders.reduce((sum, order) =>
    sum + order.items.reduce((itemSum, item) =>
      itemSum + (item.price * item.quantity), 0
    ), 0
  );

  const avgOrderValue = totalOrders > 0 ? totalRevenue / totalOrders : 0;

  // Orders by location type
  const locationBreakdown = orders.reduce((acc, order) => {
    acc[order.location_type] = (acc[order.location_type] || 0) + 1;
    return acc;
  }, {});

  // Top-selling items
  const itemSales = {};
  orders.forEach(order => {
    order.items.forEach(item => {
      if (!itemSales[item.menu_item.name]) {
        itemSales[item.menu_item.name] = { quantity: 0, revenue: 0 };
      }
      itemSales[item.menu_item.name].quantity += item.quantity;
      itemSales[item.menu_item.name].revenue += item.price * item.quantity;
    });
  });

  const topSellingItems = Object.entries(itemSales)
    .map(([name, data]) => ({ name, ...data }))
    .sort((a, b) => b.quantity - a.quantity)
    .slice(0, 10);

  res.json({
    date,
    summary: {
      totalOrders,
      totalRevenue: parseFloat(totalRevenue.toFixed(2)),
      avgOrderValue: parseFloat(avgOrderValue.toFixed(2))
    },
    locationBreakdown,
    topSellingItems
  });
```

```
  } catch (error) {
    console.error('Daily sales error:', error);
    res.status(500).json({ error: 'Failed to fetch daily sales' });
  }
});

// 2. Weekly/Monthly Revenue Trends
app.get('/admin/analytics/revenue-trends', verifyAdmin, async (req, res) => {
  const { period = 'weekly', weeks = 4 } = req.query; // weekly or monthly

  try {
    const endDate = new Date();
    const startDate = new Date();

    if (period === 'weekly') {
      startDate.setDate(endDate.getDate() - (weeks * 7));
    } else {
      startDate.setMonth(endDate.getMonth() - weeks);
    }

    const orders = await db.orders.findMany({
      where: {
        placed_at: {
          gte: startDate,
          lte: endDate
        },
        status: 'completed'
      },
      include: {
        items: true
      }
    });

    // Group by date
    const revenueByDate = {};
    orders.forEach(order => {
      const dateKey = order.placed_at.toISOString().split('T')[0];
      if (!revenueByDate[dateKey]) {
        revenueByDate[dateKey] = { orders: 0, revenue: 0 };
      }
      revenueByDate[dateKey].orders += 1;
      revenueByDate[dateKey].revenue += order.items.reduce((sum, item) =>
        sum + (item.price * item.quantity), 0
      );
    });

    // Fill missing dates with 0
    const trends = [];
    let currentDate = new Date(startDate);
```

```
    while (currentDate <= endDate) {
      const dateKey = currentDate.toISOString().split('T')[0];
      trends.push({
        date: dateKey,
        orders: revenueByDate[dateKey]?.orders || 0,
        revenue: parseFloat((revenueByDate[dateKey]?.revenue || 0).toFixed(2))
      });
      currentDate.setDate(currentDate.getDate() + 1);
    }

    res.json({ period, trends });
  } catch (error) {
    console.error('Revenue trends error:', error);
    res.status(500).json({ error: 'Failed to fetch revenue trends' });
  }
});

// 3. Top Customers Analysis
app.get('/admin/analytics/top-customers', verifyAdmin, async (req, res) => {
  const { limit = 10, days = 30 } = req.query;

  try {
    const startDate = new Date();
    startDate.setDate(startDate.getDate() - days);

    const customerStats = await db.orders.groupBy({
      by: ['user_id'],
      where: {
        placed_at: { gte: startDate },
        status: 'completed'
      },
      _count: {
        id: true
      },
      _sum: {
        // You'll need to add a total_amount field to orders table
        total_amount: true
      }
    });

    // Get user details
    const topCustomers = await Promise.all(
      customerStats
        .sort((a, b) => b._sum.total_amount - a._sum.total_amount)
        .slice(0, parseInt(limit))
        .map(async (stat) => {
          const user = await db.users.findUnique({
            where: { id: stat.user_id },
            select: { phone_number: true, whatsapp_number: true, joined_at: true }
```

```
        });

        return {
          user_id: stat.user_id,
          phone_number: user?.phone_number,
          whatsapp_number: user?.whatsapp_number,
          total_orders: stat._count.id,
          total_spent: parseFloat((stat._sum.total_amount || 0).toFixed(2)),
          avg_order_value: parseFloat((stat._sum.total_amount / stat._count.id).toFixed(2))
        };
      })
    );

    res.json({
      period_days: days,
      top_customers: topCustomers
    });
  } catch (error) {
    console.error('Top customers error:', error);
    res.status(500).json({ error: 'Failed to fetch top customers' });
  }
});

// 4. Menu Item Performance
app.get('/admin/analytics/menu-performance', verifyAdmin, async (req, res) => {
  const { days = 30 } = req.query;

  try {
    const startDate = new Date();
    startDate.setDate(startDate.getDate() - days);

    const itemPerformance = await db.orderItems.groupBy({
      by: ['item_id'],
      where: {
        order: {
          placed_at: { gte: startDate },
          status: 'completed'
        }
      },
      _sum: {
        quantity: true
      },
      _count: {
        id: true
      }
    });

    // Get menu item details and calculate revenue
    const performanceWithDetails = await Promise.all(
```

```
      itemPerformance.map(async (item) => {
        const menuItem = await db.menuItems.findUnique({
          where: { id: item.item_id },
          select: { name: true, price: true, category: true }
        });

        const totalRevenue = (item._sum.quantity || 0) * (menuItem?.price || 0);

        return {
          item_id: item.item_id,
          name: menuItem?.name,
          category: menuItem?.category,
          price: menuItem?.price,
          total_quantity_sold: item._sum.quantity || 0,
          total_orders: item._count.id,
          total_revenue: parseFloat(totalRevenue.toFixed(2))
        };
      })
    );

    // Sort by revenue
    performanceWithDetails.sort((a, b) => b.total_revenue - a.total_revenue);

    res.json({
      period_days: days,
      menu_performance: performanceWithDetails
    });
  } catch (error) {
    console.error('Menu performance error:', error);
    res.status(500).json({ error: 'Failed to fetch menu performance' });
  }
});

// 5. Location-wise Analysis
app.get('/admin/analytics/location-analysis', verifyAdmin, async (req, res) => {
  const { days = 30 } = req.query;

  try {
    const startDate = new Date();
    startDate.setDate(startDate.getDate() - days);

    const locationStats = await db.orders.groupBy({
      by: ['location_type', 'location_id'],
      where: {
        placed_at: { gte: startDate },
        status: 'completed'
      },
      _count: {
        id: true
```

```
      },
      _sum: {
        total_amount: true
      }
    });

    // Group by location type
    const locationAnalysis = locationStats.reduce((acc, stat) => {
      if (!acc[stat.location_type]) {
        acc[stat.location_type] = {
          type: stat.location_type,
          total_orders: 0,
          total_revenue: 0,
          locations: []
        };
      }

      acc[stat.location_type].total_orders += stat._count.id;
      acc[stat.location_type].total_revenue += stat._sum.total_amount || 0;
      acc[stat.location_type].locations.push({
        location_id: stat.location_id,
        orders: stat._count.id,
        revenue: parseFloat((stat._sum.total_amount || 0).toFixed(2))
      });

      return acc;
    }, {});

    // Sort locations within each type
    Object.values(locationAnalysis).forEach(typeData => {
      typeData.locations.sort((a, b) => b.revenue - a.revenue);
      typeData.total_revenue = parseFloat(typeData.total_revenue.toFixed(2));
    });

    res.json({
      period_days: days,
      location_analysis: Object.values(locationAnalysis)
    });
  } catch (error) {
    console.error('Location analysis error:', error);
    res.status(500).json({ error: 'Failed to fetch location analysis' });
  }
});

// 6. Peak Hours Analysis
app.get('/admin/analytics/peak-hours', verifyAdmin, async (req, res) => {
  const { days = 30 } = req.query;

  try {
```

```
    const startDate = new Date();
    startDate.setDate(startDate.getDate() - days);

    const orders = await db.orders.findMany({
      where: {
        placed_at: { gte: startDate },
        status: 'completed'
      },
      select: {
        placed_at: true,
        total_amount: true
      }
    });

    // Group by hour
    const hourlyStats = {};
    orders.forEach(order => {
      const hour = order.placed_at.getHours();
      if (!hourlyStats[hour]) {
        hourlyStats[hour] = { orders: 0, revenue: 0 };
      }
      hourlyStats[hour].orders += 1;
      hourlyStats[hour].revenue += order.total_amount || 0;
    });

    // Fill all 24 hours
    const peakHours = [];
    for (let hour = 0; hour < 24; hour++) {
      peakHours.push({
        hour,
        time: `${hour.toString().padStart(2, '0')}:00`,
        orders: hourlyStats[hour]?.orders || 0,
        revenue: parseFloat((hourlyStats[hour]?.revenue || 0).toFixed(2))
      });
    }

    // Sort by orders to find peak hours
    const sortedByOrders = [...peakHours].sort((a, b) => b.orders - a.orders);

    res.json({
      period_days: days,
      peak_hours: peakHours,
      top_3_hours: sortedByOrders.slice(0, 3)
    });
  } catch (error) {
    console.error('Peak hours error:', error);
    res.status(500).json({ error: 'Failed to fetch peak hours analysis' });
  }
});
```

```
// 7. Order Status Analytics
app.get('/admin/analytics/order-status', verifyAdmin, async (req, res) => {
  const { days = 7 } = req.query;

  try {
    const startDate = new Date();
    startDate.setDate(startDate.getDate() - days);

    const statusStats = await db.orders.groupBy({
      by: ['status'],
      where: {
        placed_at: { gte: startDate }
      },
      _count: {
        id: true
      }
    });

    // Calculate average preparation time for completed orders
    const completedOrders = await db.orders.findMany({
      where: {
        placed_at: { gte: startDate },
        status: 'completed'
      },
      select: {
        placed_at: true,
        updated_at: true
      }
    });

    const avgPrepTime = completedOrders.length > 0
      ? completedOrders.reduce((sum, order) => {
          const prepTime = (order.updated_at - order.placed_at) / (1000 * 60); // minutes
          return sum + prepTime;
        }, 0) / completedOrders.length
      : 0;

    const statusBreakdown = statusStats.reduce((acc, stat) => {
      acc[stat.status] = stat._count.id;
      return acc;
    }, {});

    res.json({
      period_days: days,
      status_breakdown: statusBreakdown,
      avg_preparation_time: parseFloat(avgPrepTime.toFixed(2)),
      total_orders: statusStats.reduce((sum, stat) => sum + stat._count.id, 0)
    });
```

```
  } catch (error) {
    console.error('Order status error:', error);
    res.status(500).json({ error: 'Failed to fetch order status analytics' });
  }
});

module.exports = app;
```

## Invoice/Receipt Generation APi

```
// Invoice/Receipt Generation API
const express = require('express');
const PDFDocument = require('pdfkit');
const nodemailer = require('nodemailer');
const { verifyUser } = require('./auth');
const app = express();

// Configure email transporter
const emailTransporter = nodemailer.createTransporter({
  service: 'gmail', // or your preferred email service
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS
  }
});

// Restaurant details (can be stored in config/database)
const RESTAURANT_INFO = {
  name: "Your Restaurant Name",
  address: "123 Restaurant Street, City, State 12345",
  phone: "+1-234-567-8900",
  email: "orders@yourrestaurant.com",
  gst_number: "GST123456789", // if applicable
  logo_url: "https://yourrestaurant.com/logo.png"
};

// 1. Generate Invoice PDF
function generateInvoicePDF(orderData) {
  return new Promise((resolve, reject) => {
    try {
      const doc = new PDFDocument({ margin: 50 });
      const buffers = [];

      doc.on('data', buffers.push.bind(buffers));
      doc.on('end', () => {
        const pdfBuffer = Buffer.concat(buffers);
        resolve(pdfBuffer);
```

```
    });

    // Header
    doc.fontSize(20).text(RESTAURANT_INFO.name, { align: 'center' });
    doc.fontSize(10).text(RESTAURANT_INFO.address, { align: 'center' });
    doc.text(`Phone: ${RESTAURANT_INFO.phone} | Email: ${RESTAURANT_INFO.email}`, { align:
'center' });

    if (RESTAURANT_INFO.gst_number) {
      doc.text(`GST No: ${RESTAURANT_INFO.gst_number}`, { align: 'center' });
    }

    doc.moveDown(2);

    // Invoice details
    doc.fontSize(16).text('INVOICE', { align: 'center' });
    doc.moveDown();

    const invoiceY = doc.y;
    doc.fontSize(10)
       .text(`Invoice #: ${orderData.invoice_number}`, 50, invoiceY)
       .text(`Order #: ${orderData.id}`, 50, invoiceY + 15)
       .text(`Date: ${new Date(orderData.placed_at).toLocaleDateString()}`, 50, invoiceY + 30)
       .text(`Time: ${new Date(orderData.placed_at).toLocaleTimeString()}`, 50, invoiceY + 45);

    // Customer details
    doc.text(`Customer: ${orderData.customer_phone || 'Walk-in'}`, 300, invoiceY)
       .text(`Location: ${orderData.location_type.toUpperCase()} ${orderData.location_id}`, 300, invoiceY
+ 15);

    if (orderData.customer_notes) {
      doc.text(`Notes: ${orderData.customer_notes}`, 300, invoiceY + 30);
    }

    doc.moveDown(4);

    // Table header
    const tableTop = doc.y;
    doc.fontSize(10)
       .text('Item', 50, tableTop)
       .text('Qty', 250, tableTop)
       .text('Price', 300, tableTop)
       .text('Amount', 450, tableTop);

    doc.moveTo(50, tableTop + 15)
       .lineTo(550, tableTop + 15)
       .stroke();

    // Items
```

```
    let itemY = tableTop + 25;
    let subtotal = 0;

    orderData.items.forEach(item => {
      const amount = item.price * item.quantity;
      subtotal += amount;

      doc.text(item.item_name, 50, itemY)
         .text(item.quantity.toString(), 250, itemY)
         .text(`$${item.price.toFixed(2)}`, 300, itemY)
         .text(`$${amount.toFixed(2)}`, 450, itemY);

      if (item.notes) {
        itemY += 15;
        doc.fontSize(8).text(`  Note: ${item.notes}`, 50, itemY).fontSize(10);
      }

      itemY += 20;
    });

    // Totals
    doc.moveTo(50, itemY)
       .lineTo(550, itemY)
       .stroke();

    itemY += 10;

    // Calculate tax (example: 10%)
    const taxRate = 0.10;
    const taxAmount = subtotal * taxRate;
    const total = subtotal + taxAmount;

    doc.text('Subtotal:', 350, itemY)
       .text(`$${subtotal.toFixed(2)}`, 450, itemY);

    itemY += 15;
    doc.text(`Tax (${(taxRate * 100)}%):`, 350, itemY)
       .text(`$${taxAmount.toFixed(2)}`, 450, itemY);

    itemY += 15;
    doc.fontSize(12)
       .text('Total:', 350, itemY)
       .text(`$${total.toFixed(2)}`, 450, itemY);

    doc.fontSize(10);

    // Payment status
    itemY += 30;
    const paymentStatus = orderData.payment_status === 'paid' ? 'PAID' : 'PENDING';
```

```
      doc.text(`Payment Status: ${paymentStatus}`, 50, itemY);

      // Footer
      doc.text('Thank you for your order!', 50, doc.page.height - 100, { align: 'center' });
      doc.text('Visit us again soon!', 50, doc.page.height - 85, { align: 'center' });

      doc.end();
    } catch (error) {
      reject(error);
    }
  });
}

// 2. Get Invoice for Order
app.get('/orders/:orderId/invoice', verifyUser, async (req, res) => {
  try {
    const { orderId } = req.params;

    const order = await db.orders.findUnique({
      where: { id: orderId },
      include: {
        items: true,
        user: {
          select: { phone_number: true, whatsapp_number: true }
        }
      }
    });

    if (!order) {
      return res.status(404).json({ error: 'Order not found' });
    }

    // Check if user owns this order
    if (order.user_id !== req.userId) {
      return res.status(403).json({ error: 'Access denied' });
    }

    // Generate invoice number if not exists
    const invoiceNumber = order.invoice_number || `INV-${Date.now()}`;

    if (!order.invoice_number) {
      await db.orders.update({
        where: { id: orderId },
        data: { invoice_number: invoiceNumber }
      });
    }

    const orderData = {
      ...order,
```

```
      invoice_number: invoiceNumber,
      customer_phone: order.user.phone_number || order.user.whatsapp_number
    };

    const pdfBuffer = await generateInvoicePDF(orderData);

    res.setHeader('Content-Type', 'application/pdf');
    res.setHeader('Content-Disposition', `attachment; filename="invoice-${invoiceNumber}.pdf"`);
    res.send(pdfBuffer);

  } catch (error) {
    console.error('Invoice generation error:', error);
    res.status(500).json({ error: 'Failed to generate invoice' });
  }
});

// 3. Email Invoice to Customer
app.post('/orders/:orderId/email-invoice', verifyUser, async (req, res) => {
  try {
    const { orderId } = req.params;
    const { email } = req.body;

    if (!email) {
      return res.status(400).json({ error: 'Email address required' });
    }

    const order = await db.orders.findUnique({
      where: { id: orderId },
      include: {
        items: true,
        user: {
          select: { phone_number: true, whatsapp_number: true }
        }
      }
    });

    if (!order || order.user_id !== req.userId) {
      return res.status(404).json({ error: 'Order not found or access denied' });
    }

    const invoiceNumber = order.invoice_number || `INV-${Date.now()}`;

    const orderData = {
      ...order,
      invoice_number: invoiceNumber,
      customer_phone: order.user.phone_number || order.user.whatsapp_number
    };

    const pdfBuffer = await generateInvoicePDF(orderData);
```

```
    // Send email with PDF attachment
    const mailOptions = {
      from: RESTAURANT_INFO.email,
      to: email,
      subject: `Invoice for Order #${order.id} - ${RESTAURANT_INFO.name}`,
      html: `
        <h2>${RESTAURANT_INFO.name}</h2>
        <p>Dear Customer,</p>
        <p>Thank you for your order! Please find your invoice attached.</p>
        <p><strong>Order Details:</strong></p>
        <ul>
          <li>Order #: ${order.id}</li>
          <li>Date: ${new Date(order.placed_at).toLocaleDateString()}</li>
          <li>Location: ${order.location_type.toUpperCase()} ${order.location_id}</li>
          <li>Total: $${order.items.reduce((sum, item) => sum + (item.price * item.quantity),
0).toFixed(2)}</li>
        </ul>
        <p>We appreciate your business!</p>
        <br>
        <p>Best regards,<br>${RESTAURANT_INFO.name}<br>${RESTAURANT_INFO.phone}</p>
      `,
      attachments: [
        {
          filename: `invoice-${invoiceNumber}.pdf`,
          content: pdfBuffer,
          contentType: 'application/pdf'
        }
      ]
    };

    await emailTransporter.sendMail(mailOptions);

    res.json({
      success: true,
      message: 'Invoice emailed successfully',
      invoice_number: invoiceNumber
    });

  } catch (error) {
    console.error('Email invoice error:', error);
    res.status(500).json({ error: 'Failed to email invoice' });
  }
});

// 4. Generate Receipt (Simplified version for quick printing)
app.get('/orders/:orderId/receipt', verifyUser, async (req, res) => {
  try {
    const { orderId } = req.params;
```

```
    const order = await db.orders.findUnique({
      where: { id: orderId },
      include: {
        items: true
      }
    });

    if (!order || order.user_id !== req.userId) {
      return res.status(404).json({ error: 'Order not found or access denied' });
    }

    const doc = new PDFDocument({ size: 'A6', margin: 20 }); // Smaller receipt size
    const buffers = [];

    doc.on('data', buffers.push.bind(buffers));
    doc.on('end', () => {
      const pdfBuffer = Buffer.concat(buffers);
      res.setHeader('Content-Type', 'application/pdf');
      res.setHeader('Content-Disposition', `inline; filename="receipt-${order.id}.pdf"`);
      res.send(pdfBuffer);
    });

    // Receipt header
    doc.fontSize(14).text(RESTAURANT_INFO.name, { align: 'center' });
    doc.fontSize(8).text(RESTAURANT_INFO.phone, { align: 'center' });
    doc.moveDown();

    // Order info
    doc.fontSize(10)
       .text(`Receipt #: ${order.id}`)
       .text(`Date: ${new Date(order.placed_at).toLocaleString()}`)
       .text(`Location: ${order.location_type.toUpperCase()} ${order.location_id}`);

    doc.moveDown();
    doc.text('Items:', { underline: true });

    let total = 0;
    order.items.forEach(item => {
      const amount = item.price * item.quantity;
      total += amount;
      doc.text(`${item.item_name} x${item.quantity} - $${amount.toFixed(2)}`);
    });

    doc.moveDown();
    doc.fontSize(12).text(`Total: $${total.toFixed(2)}`, { align: 'right' });

    doc.fontSize(8).text('Thank you!', { align: 'center' });
```

```
    doc.end();

  } catch (error) {
    console.error('Receipt generation error:', error);
    res.status(500).json({ error: 'Failed to generate receipt' });
  }
});

// 5. Bulk Invoice Generation (Admin)
app.post('/admin/generate-bulk-invoices', async (req, res) => {
  const { date, email_to_customers = false } = req.body;

  try {
    const startDate = new Date(`${date}T00:00:00.000Z`);
    const endDate = new Date(`${date}T23:59:59.999Z`);

    const orders = await db.orders.findMany({
      where: {
        placed_at: { gte: startDate, lte: endDate },
        status: 'completed',
        invoice_number: null // Only orders without invoices
      },
      include: {
        items: true,
        user: {
          select: { phone_number: true, whatsapp_number: true }
        }
      }
    });

    const results = [];

    for (const order of orders) {
      const invoiceNumber = `INV-${Date.now()}-${order.id}`;

      await db.orders.update({
        where: { id: order.id },
        data: { invoice_number: invoiceNumber }
      });

      results.push({
        order_id: order.id,
        invoice_number: invoiceNumber,
        status: 'generated'
      });

      // Optional: Email to customers
      if (email_to_customers && order.user.email) {
        try {
```

```
        const orderData = {
          ...order,
          invoice_number: invoiceNumber,
          customer_phone: order.user.phone_number || order.user.whatsapp_number
        };

        const pdfBuffer = await generateInvoicePDF(orderData);

        await emailTransporter.sendMail({
          from: RESTAURANT_INFO.email,
          to: order.user.email,
          subject: `Invoice for Order #${order.id}`,
          text: 'Please find your invoice attached.',
          attachments: [{
            filename: `invoice-${invoiceNumber}.pdf`,
            content: pdfBuffer,
            contentType: 'application/pdf'
          }]
        });

        results[results.length - 1].email_status = 'sent';
      } catch (emailError) {
        results[results.length - 1].email_status = 'failed';
      }
    }
  }

  res.json({
    success: true,
    processed: results.length,
    results
  });

 } catch (error) {
  console.error('Bulk invoice error:', error);
  res.status(500).json({ error: 'Failed to generate bulk invoices' });
 }
});

module.exports = app;
```

## Realtime Order updates

```
// Real-time Order Updates with WebSocket
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
```

```
const jwt = require('jsonwebtoken');

const app = express();
const server = http.createServer(app);
const io = socketIo(server, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"]
  }
});

const JWT_SECRET = process.env.JWT_SECRET || 'your_jwt_secret';

// Store connected clients by role
const connectedClients = {
  kitchen: new Map(),
  counter: new Map(),
  customers: new Map(),
  admin: new Map()
};

// WebSocket Authentication Middleware
const authenticateSocket = (socket, next) => {
  const token = socket.handshake.auth.token;
  const role = socket.handshake.auth.role; // 'kitchen', 'counter', 'customer', 'admin'

  if (!token) {
    return next(new Error('Authentication token required'));
  }

  try {
    const payload = jwt.verify(token, JWT_SECRET);
    socket.userId = payload.userId || payload.adminId;
    socket.role = role;
    socket.userPhone = payload.phone_number;
    next();
  } catch (error) {
    next(new Error('Invalid authentication token'));
  }
};

io.use(authenticateSocket);

// Handle WebSocket connections
io.on('connection', (socket) => {
  console.log(`${socket.role} connected: ${socket.id}`);

  // Store client connection
  if (connectedClients[socket.role]) {
```

```
    connectedClients[socket.role].set(socket.userId, {
      socketId: socket.id,
      socket: socket,
      connectedAt: new Date()
    });
  }

  // Send initial data based on role
  handleInitialConnection(socket);

  // Handle disconnection
  socket.on('disconnect', () => {
    console.log(`${socket.role} disconnected: ${socket.id}`);
    if (connectedClients[socket.role]) {
      connectedClients[socket.role].delete(socket.userId);
    }
  });

  // Handle role-specific events
  handleSocketEvents(socket);
});

// Initial connection data
async function handleInitialConnection(socket) {
  try {
    switch (socket.role) {
      case 'kitchen':
        const activeOrders = await getActiveOrdersForKitchen();
        socket.emit('initial_orders', activeOrders);
        break;

      case 'counter':
        const paymentPendingOrders = await getPaymentPendingOrders();
        socket.emit('initial_orders', paymentPendingOrders);
        break;

      case 'customer':
        const customerOrders = await getCustomerActiveOrders(socket.userId);
        socket.emit('order_updates', customerOrders);
        break;

      case 'admin':
        const adminStats = await getRealtimeStats();
        socket.emit('admin_stats', adminStats);
        break;
    }
  } catch (error) {
    console.error('Initial connection error:', error);
  }
```

```
}

// Handle role-specific socket events
function handleSocketEvents(socket) {
  // Kitchen events
  if (socket.role === 'kitchen') {
    socket.on('update_order_status', async (data) => {
      const { orderId, status } = data;
      try {
        await updateOrderStatusInDB(orderId, status);

        // Notify all relevant parties
        notifyOrderStatusChange(orderId, status);

        socket.emit('status_updated', { orderId, status, success: true });
      } catch (error) {
        socket.emit('status_updated', { orderId, status, success: false, error: error.message });
      }
    });

    socket.on('request_order_details', async (orderId) => {
      try {
        const orderDetails = await getOrderDetails(orderId);
        socket.emit('order_details', orderDetails);
      } catch (error) {
        socket.emit('error', { message: 'Failed to fetch order details' });
      }
    });
  }

  // Counter events
  if (socket.role === 'counter') {
    socket.on('update_payment_status', async (data) => {
      const { orderId, paymentStatus } = data;
      try {
        await updatePaymentStatusInDB(orderId, paymentStatus);

        // Notify customer and kitchen
        notifyPaymentStatusChange(orderId, paymentStatus);

        socket.emit('payment_updated', { orderId, paymentStatus, success: true });
      } catch (error) {
        socket.emit('payment_updated', { orderId, paymentStatus, success: false, error: error.message });
      }
    });
  }

  // Customer events
  if (socket.role === 'customer') {
```

```
    socket.on('track_order', async (orderId) => {
      try {
        const orderStatus = await getOrderStatus(orderId);
        socket.emit('order_status', orderStatus);
      } catch (error) {
        socket.emit('error', { message: 'Failed to track order' });
      }
    });
  }
}

// Database operations
async function getActiveOrdersForKitchen() {
  return await db.orders.findMany({
    where: {
      status: { in: ['pending', 'preparing'] }
    },
    include: {
      items: true
    },
    orderBy: {
      placed_at: 'asc'
    }
  });
}

async function getPaymentPendingOrders() {
  return await db.orders.findMany({
    where: {
      payment_status: 'unpaid',
      status: { in: ['ready', 'completed'] }
    },
    include: {
      items: true
    },
    orderBy: {
      placed_at: 'asc'
    }
  });
}

async function getCustomerActiveOrders(userId) {
  return await db.orders.findMany({
    where: {
      user_id: userId,
      status: { in: ['pending', 'preparing', 'ready'] }
    },
    include: {
      items: true
```

```
    },
    orderBy: {
      placed_at: 'desc'
    }
  });
}

async function updateOrderStatusInDB(orderId, status) {
  return await db.orders.update({
    where: { id: orderId },
    data: {
      status,
      updated_at: new Date()
    }
  });
}

async function updatePaymentStatusInDB(orderId, paymentStatus) {
  return await db.orders.update({
    where: { id: orderId },
    data: {
      payment_status: paymentStatus,
      updated_at: new Date()
    }
  });
}

async function getOrderDetails(orderId) {
  return await db.orders.findUnique({
    where: { id: orderId },
    include: {
      items: true,
      user: {
        select: { phone_number: true, whatsapp_number: true }
      }
    }
  });
}

async function getOrderStatus(orderId) {
  return await db.orders.findUnique({
    where: { id: orderId },
    select: {
      id: true,
      status: true,
      payment_status: true,
      placed_at: true,
      updated_at: true
    }
```

```
  });
}

async function getRealtimeStats() {
  const today = new Date();
  today.setHours(0, 0, 0, 0);

  const [totalOrders, pendingOrders, revenue, activeCustomers] = await Promise.all([
    db.orders.count({
      where: { placed_at: { gte: today } }
    }),
    db.orders.count({
      where: { status: 'pending' }
    }),
    db.orders.aggregate({
      where: {
        placed_at: { gte: today },
        status: 'completed'
      },
      _sum: { total_amount: true }
    }),
    db.orders.groupBy({
      by: ['user_id'],
      where: { placed_at: { gte: today } }
    }).then(result => result.length)
  ]);

  return {
    totalOrders,
    pendingOrders,
    revenue: revenue._sum.total_amount || 0,
    activeCustomers
  };
}

// Notification functions
function notifyOrderStatusChange(orderId, newStatus) {
  // Notify kitchen staff
  broadcastToRole('kitchen', 'order_status_changed', { orderId, status: newStatus });

  // Notify counter staff
  broadcastToRole('counter', 'order_status_changed', { orderId, status: newStatus });

  // Notify customer (find the customer who owns this order)
  notifyCustomerOfOrderUpdate(orderId, newStatus);

  // Notify admin
  broadcastToRole('admin', 'order_status_changed', { orderId, status: newStatus });
}
```

```
function notifyPaymentStatusChange(orderId, paymentStatus) {
  broadcastToRole('kitchen', 'payment_status_changed', { orderId, paymentStatus });
  broadcastToRole('admin', 'payment_status_changed', { orderId, paymentStatus });
  notifyCustomerOfPaymentUpdate(orderId, paymentStatus);
}

async function notifyCustomerOfOrderUpdate(orderId, status) {
  try {
    const order = await db.orders.findUnique({
      where: { id: orderId },
      select: { user_id: true }
    });

    if (order && connectedClients.customers.has(order.user_id)) {
      const customer = connectedClients.customers.get(order.user_id);
      customer.socket.emit('order_status_update', { orderId, status });
    }
  } catch (error) {
    console.error('Error notifying customer:', error);
  }
}

async function notifyCustomerOfPaymentUpdate(orderId, paymentStatus) {
  try {
    const order = await db.orders.findUnique({
      where: { id: orderId },
      select: { user_id: true }
    });

    if (order && connectedClients.customers.has(order.user_id)) {
      const customer = connectedClients.customers.get(order.user_id);
      customer.socket.emit('payment_status_update', { orderId, paymentStatus });
    }
  } catch (error) {
    console.error('Error notifying customer of payment:', error);
  }
}

function broadcastToRole(role, event, data) {
  if (connectedClients[role]) {
    connectedClients[role].forEach((client) => {
      client.socket.emit(event, data);
    });
  }
}

// REST API integration - call these from your existing APIs
function notifyNewOrder(order) {
```

```
  // Notify kitchen of new order
  broadcastToRole('kitchen', 'new_order', order);

  // Notify counter if payment is required
  if (order.payment_status === 'unpaid') {
    broadcastToRole('counter', 'new_order', order);
  }

  // Update admin dashboard
  broadcastToRole('admin', 'new_order', order);
}

// Export functions for use in other modules
module.exports = {
  io,
  server,
  notifyNewOrder,
  notifyOrderStatusChange,
  notifyPaymentStatusChange,
  broadcastToRole
};

// Example integration with existing order placement API:
/*
// In your order placement API:
app.post('/place-order', async (req, res) => {
  // ... existing order creation logic ...

  // After creating order:
  const { notifyNewOrder } = require('./realtime-updates');
  notifyNewOrder(createdOrder);

  res.json({ success: true, order_id: createdOrder.id });
});
*/

// Start the server
const PORT = process.env.PORT || 3001;
server.listen(PORT, () => {
  console.log(`Real-time server running on port ${PORT}`);
});
```

# Multi Restaurant Support Api

```
// Multi-Restaurant Support API
const express = require('express');
const { verifyAdmin, verifyUser } = require('./auth');
const app = express();

// Updated Database Schema (add restaurant_id to existing tables)
/*
restaurants {
  id: string,
  name: string,
  address: string,
  phone: string,
  email: string,
  logo_url: string,
  is_active: boolean,
  created_at: Date
}

// Add restaurant_id to existing tables:
- orders.restaurant_id
- menu_items.restaurant_id
- offers.restaurant_id
- sessions.restaurant_id
- qr_codes.restaurant_id (new table for tracking QR codes)
*/

// 1. Create Restaurant (Super Admin)
app.post('/admin/restaurants', verifyAdmin, async (req, res) => {
  const { name, address, phone, email, logo_url } = req.body;

  try {
    const restaurant = await db.restaurants.create({
      data: {
        name,
        address,
        phone,
        email,
        logo_url,
        is_active: true,
        created_at: new Date()
      }
    });

    res.json({ success: true, restaurant });
  } catch (error) {
    console.error('Create restaurant error:', error);
```

```
    res.status(500).json({ error: 'Failed to create restaurant' });
  }
});

// 2. List All Restaurants
app.get('/restaurants', async (req, res) => {
  try {
    const restaurants = await db.restaurants.findMany({
      where: { is_active: true },
      select: {
        id: true,
        name: true,
        address: true,
        phone: true,
        logo_url: true
      }
    });

    res.json(restaurants);
  } catch (error) {
    console.error('List restaurants error:', error);
    res.status(500).json({ error: 'Failed to fetch restaurants' });
  }
});

// 3. Get Restaurant Details
app.get('/restaurants/:restaurantId', async (req, res) => {
  const { restaurantId } = req.params;

  try {
    const restaurant = await db.restaurants.findUnique({
      where: { id: restaurantId, is_active: true }
    });

    if (!restaurant) {
      return res.status(404).json({ error: 'Restaurant not found' });
    }

    // Get restaurant stats
    const [menuCount, todayOrders, totalCustomers] = await Promise.all([
      db.menuItems.count({ where: { restaurant_id: restaurantId } }),
      db.orders.count({
        where: {
          restaurant_id: restaurantId,
          placed_at: {
            gte: new Date(new Date().setHours(0, 0, 0, 0))
          }
        }
      }),
```

```
      db.orders.groupBy({
        by: ['user_id'],
        where: { restaurant_id: restaurantId }
      }).then(result => result.length)
    ]);

    res.json({
      ...restaurant,
      stats: {
        menu_items: menuCount,
        today_orders: todayOrders,
        total_customers: totalCustomers
      }
    });
  } catch (error) {
    console.error('Get restaurant error:', error);
    res.status(500).json({ error: 'Failed to fetch restaurant details' });
  }
});

// 4. Update QR Scan Flow to Support Multiple Restaurants
app.post('/qr-scan', verifyUser, async (req, res) => {
  const { qr_data } = req.body;
  const userId = req.userId;

  // QR data now includes restaurant_id
  // Format: { type: "table", location_id: "TBL001", restaurant_id: "REST001" }

  if (!qr_data.restaurant_id) {
    return res.status(400).json({ error: 'Restaurant ID required in QR code' });
  }

  try {
    // Verify restaurant exists and is active
    const restaurant = await db.restaurants.findUnique({
      where: { id: qr_data.restaurant_id, is_active: true }
    });

    if (!restaurant) {
      return res.status(400).json({ error: 'Invalid or inactive restaurant' });
    }

    // Create session with restaurant context
    const session = await db.sessions.create({
      data: {
        user_id: userId,
        restaurant_id: qr_data.restaurant_id,
        location_type: qr_data.type,
        location_id: qr_data.location_id,
```

```
      started_at: new Date()
    }
  });

  res.json({
    success: true,
    session_id: session.id,
    restaurant: {
      id: restaurant.id,
      name: restaurant.name,
      address: restaurant.address
    }
  });
 } catch (error) {
  console.error('Multi-restaurant QR scan error:', error);
  res.status(500).json({ error: 'QR scan failed' });
 }
});

// 5. Restaurant-specific Menu
app.get('/restaurants/:restaurantId/menu', async (req, res) => {
 const { restaurantId } = req.params;
 const { category } = req.query;

 try {
  const whereClause = {
    restaurant_id: restaurantId,
    is_available: true
  };

  if (category) {
    whereClause.category = category;
  }

  const menuItems = await db.menuItems.findMany({
    where: whereClause,
    orderBy: [
      { category: 'asc' },
      { name: 'asc' }
    ]
  });

  // Group by category
  const menuByCategory = menuItems.reduce((acc, item) => {
    if (!acc[item.category]) {
      acc[item.category] = [];
    }
    acc[item.category].push(item);
    return acc;
```

```
    }, {});

    res.json({
      restaurant_id: restaurantId,
      menu: menuByCategory,
      categories: Object.keys(menuByCategory)
    });
  } catch (error) {
    console.error('Restaurant menu error:', error);
    res.status(500).json({ error: 'Failed to fetch menu' });
  }
});

// 6. Restaurant-specific Order Placement
app.post('/restaurants/:restaurantId/place-order', verifyUser, async (req, res) => {
  const { restaurantId } = req.params;
  const { session_id, items, notes } = req.body;
  const userId = req.userId;

  try {
    // Verify session belongs to this restaurant
    const session = await db.sessions.findUnique({
      where: { id: session_id }
    });

    if (!session || session.restaurant_id !== restaurantId) {
      return res.status(400).json({ error: 'Invalid session for this restaurant' });
    }

    // Create order with restaurant context
    const order = await db.orders.create({
      data: {
        user_id: userId,
        restaurant_id: restaurantId,
        session_id,
        location_type: session.location_type,
        location_id: session.location_id,
        notes,
        status: 'pending',
        payment_status: 'unpaid',
        placed_at: new Date()
      }
    });

    // Add order items
    for (const item of items) {
      // Verify item belongs to this restaurant
      const menuItem = await db.menuItems.findUnique({
        where: { id: item.item_id }
```

```
      });

      if (!menuItem || menuItem.restaurant_id !== restaurantId) {
        await db.orders.delete({ where: { id: order.id } });
        return res.status(400).json({ error: 'Invalid menu item for this restaurant' });
      }

      await db.orderItems.create({
        data: {
          order_id: order.id,
          item_id: item.item_id,
          item_name: menuItem.name,
          quantity: item.quantity,
          price: menuItem.price
        }
      });
    }

    // Notify restaurant-specific kitchen/counter
    notifyRestaurantStaff(restaurantId, order.id);

    res.json({ success: true, order_id: order.id });
  } catch (error) {
    console.error('Multi-restaurant order error:', error);
    res.status(500).json({ error: 'Order placement failed' });
  }
});

// 7. Restaurant-specific Admin APIs
app.get('/admin/restaurants/:restaurantId/orders', verifyAdmin, async (req, res) => {
  const { restaurantId } = req.params;
  const { status, date, limit = 50 } = req.query;

  try {
    const whereClause = { restaurant_id: restaurantId };

    if (status) {
      whereClause.status = status;
    }

    if (date) {
      const startDate = new Date(`${date}T00:00:00.000Z`);
      const endDate = new Date(`${date}T23:59:59.999Z`);
      whereClause.placed_at = { gte: startDate, lte: endDate };
    }

    const orders = await db.orders.findMany({
      where: whereClause,
      include: {
```

```
        items: true,
        user: {
          select: { phone_number: true, whatsapp_number: true }
        }
      },
      orderBy: { placed_at: 'desc' },
      take: parseInt(limit)
    });

    res.json({
      restaurant_id: restaurantId,
      orders
    });
  } catch (error) {
    console.error('Restaurant orders error:', error);
    res.status(500).json({ error: 'Failed to fetch orders' });
  }
});

// 8. Restaurant-specific Analytics
app.get('/admin/restaurants/:restaurantId/analytics', verifyAdmin, async (req, res) => {
  const { restaurantId } = req.params;
  const { period = 'today' } = req.query;

  try {
    let startDate, endDate;

    switch (period) {
      case 'today':
        startDate = new Date(new Date().setHours(0, 0, 0, 0));
        endDate = new Date(new Date().setHours(23, 59, 59, 999));
        break;
      case 'week':
        startDate = new Date();
        startDate.setDate(startDate.getDate() - 7);
        endDate = new Date();
        break;
      case 'month':
        startDate = new Date();
        startDate.setMonth(startDate.getMonth() - 1);
        endDate = new Date();
        break;
      default:
        startDate = new Date(new Date().setHours(0, 0, 0, 0));
        endDate = new Date(new Date().setHours(23, 59, 59, 999));
    }

    const [totalOrders, completedOrders, revenue, topItems] = await Promise.all([
      // Total orders
```

```
    db.orders.count({
      where: {
        restaurant_id: restaurantId,
        placed_at: { gte: startDate, lte: endDate }
      }
    }),

    // Completed orders
    db.orders.count({
      where: {
        restaurant_id: restaurantId,
        status: 'completed',
        placed_at: { gte: startDate, lte: endDate }
      }
    }),

    // Revenue calculation
    db.orders.findMany({
      where: {
        restaurant_id: restaurantId,
        status: 'completed',
        placed_at: { gte: startDate, lte: endDate }
      },
      include: { items: true }
    }).then(orders =>
      orders.reduce((total, order) =>
        total + order.items.reduce((orderTotal, item) =>
          orderTotal + (item.price * item.quantity), 0
        ), 0
      )
    ),

    // Top selling items
    db.orderItems.groupBy({
      by: ['item_id'],
      where: {
        order: {
          restaurant_id: restaurantId,
          status: 'completed',
          placed_at: { gte: startDate, lte: endDate }
        }
      },
      _sum: { quantity: true },
      orderBy: { _sum: { quantity: 'desc' } },
      take: 5
    }).then(items =>
      Promise.all(items.map(async item => {
        const menuItem = await db.menuItems.findUnique({
          where: { id: item.item_id },
```

```
          select: { name: true, price: true }
        });
        return {
          name: menuItem?.name,
          quantity_sold: item._sum.quantity,
          revenue: (item._sum.quantity * menuItem?.price) || 0
        };
      }))
    )
  ]);

  res.json({
    restaurant_id: restaurantId,
    period,
    analytics: {
      total_orders: totalOrders,
      completed_orders: completedOrders,
      completion_rate: totalOrders > 0 ? (completedOrders / totalOrders * 100).toFixed(1) : 0,
      revenue: parseFloat(revenue.toFixed(2)),
      avg_order_value: completedOrders > 0 ? parseFloat((revenue / completedOrders).toFixed(2)) : 0,
      top_items: topItems
    }
  });
 } catch (error) {
  console.error('Restaurant analytics error:', error);
  res.status(500).json({ error: 'Failed to fetch analytics' });
 }
});

// 9. Generate Restaurant-specific QR Codes
app.post('/admin/restaurants/:restaurantId/generate-qr-codes', verifyAdmin, async (req, res) => {
 const { restaurantId } = req.params;
 const { locations } = req.body;

 // locations format: [{ type: "table", id: "TBL001" }, ...]

 if (!locations || !Array.isArray(locations)) {
  return res.status(400).json({ error: 'Locations array required' });
 }

 try {
  const restaurant = await db.restaurants.findUnique({
    where: { id: restaurantId }
  });

  if (!restaurant) {
    return res.status(404).json({ error: 'Restaurant not found' });
  }
```

```
    const qrCodes = [];
    const QRCode = require('qrcode');
    const BASE_URL = process.env.APP_BASE_URL || 'https://yourapp.com';

    for (const location of locations) {
      const qrData = {
        type: location.type,
        location_id: location.id,
        restaurant_id: restaurantId
      };

      const qrUrl = `${BASE_URL}/order?data=${encodeURIComponent(JSON.stringify(qrData))}`;
      const qrCodeDataUrl = await QRCode.toDataURL(qrUrl);

      // Save QR code record
      const qrRecord = await db.qrCodes.create({
        data: {
          restaurant_id: restaurantId,
          location_type: location.type,
          location_id: location.id,
          qr_url: qrUrl,
          created_at: new Date()
        }
      });

      qrCodes.push({
        id: qrRecord.id,
        location: `${location.type.toUpperCase()}-${location.id}`,
        qr_data_url: qrCodeDataUrl,
        qr_url: qrUrl
      });
    }

    res.json({
      success: true,
      restaurant_id: restaurantId,
      restaurant_name: restaurant.name,
      generated_codes: qrCodes.length,
      qr_codes: qrCodes
    });
  } catch (error) {
    console.error('QR generation error:', error);
    res.status(500).json({ error: 'Failed to generate QR codes' });
  }
});

// Helper function to notify restaurant-specific staff
async function notifyRestaurantStaff(restaurantId, orderId) {
  // This would integrate with your WebSocket notification system
```

```
  // to send notifications only to staff of the specific restaurant

  const { broadcastToRestaurantRole } = require('./realtime-updates');

  const order = await db.orders.findUnique({
    where: { id: orderId },
    include: { items: true }
  });

  // Notify kitchen staff of this restaurant
  broadcastToRestaurantRole(restaurantId, 'kitchen', 'new_order', order);

  // Notify counter staff of this restaurant
  broadcastToRestaurantRole(restaurantId, 'counter', 'new_order', order);
}

// 10. Restaurant Staff Authentication
app.post('/admin/restaurants/:restaurantId/staff/login', async (req, res) => {
  const { restaurantId } = req.params;
  const { email, password, role } = req.body; // role: 'kitchen' or 'counter'

  try {
    const staff = await db.restaurantStaff.findFirst({
      where: {
        email,
        restaurant_id: restaurantId,
        role,
        is_active: true
      }
    });

    if (!staff || staff.password !== password) {
      return res.status(401).json({ error: 'Invalid credentials' });
    }

    const token = jwt.sign({
      staffId: staff.id,
      restaurantId: restaurantId,
      role: role
    }, JWT_SECRET, { expiresIn: '8h' });

    res.json({
      success: true,
      token,
      staff: {
        id: staff.id,
        name: staff.name,
        role: staff.role,
        restaurant_id: restaurantId
```

```
    }
  });
 } catch (error) {
  console.error('Staff login error:', error);
  res.status(500).json({ error: 'Login failed' });
 }
});

module.exports = app;
```

## AI Based Recommendation API

```
// AI-based Recommendations API
const express = require('express');
const { verifyUser } = require('./auth');
const app = express();

// 1. Get Personalized Recommendations for User
app.get('/recommendations/personalized', verifyUser, async (req, res) => {
 const userId = req.userId;
 const { restaurant_id, limit = 5 } = req.query;

 try {
  // Get user's order history
  const userOrders = await getUserOrderHistory(userId, restaurant_id);

  // Get user preferences based on past orders
  const userPreferences = analyzeUserPreferences(userOrders);

  // Get similar users (collaborative filtering)
  const similarUsers = await findSimilarUsers(userId, restaurant_id);

  // Get menu items for recommendations
  const menuItems = await getMenuItems(restaurant_id);

  // Generate recommendations
  const recommendations = await generatePersonalizedRecommendations(
   userId,
   userPreferences,
   similarUsers,
   menuItems,
   parseInt(limit)
  );

  res.json({
   user_id: userId,
   restaurant_id,
   recommendations,
```

```
      recommendation_type: 'personalized'
    });
  } catch (error) {
    console.error('Personalized recommendations error:', error);
    res.status(500).json({ error: 'Failed to generate recommendations' });
  }
});

// 2. Get Popular Items (Trending)
app.get('/recommendations/popular', async (req, res) => {
  const { restaurant_id, days = 7, limit = 10 } = req.query;

  try {
    const startDate = new Date();
    startDate.setDate(startDate.getDate() - parseInt(days));

    const popularItems = await db.orderItems.groupBy({
      by: ['item_id'],
      where: {
        order: {
          restaurant_id,
          status: 'completed',
          placed_at: { gte: startDate }
        }
      },
      _count: { id: true },
      _sum: { quantity: true },
      orderBy: { _sum: { quantity: 'desc' } },
      take: parseInt(limit)
    });

    // Get item details
    const recommendations = await Promise.all(
      popularItems.map(async (item, index) => {
        const menuItem = await db.menuItems.findUnique({
          where: { id: item.item_id },
          select: {
            id: true,
            name: true,
            description: true,
            price: true,
            category: true,
            image_url: true
          }
        });

        return {
          ...menuItem,
          popularity_score: item._sum.quantity,
```

```
        total_orders: item._count.id,
        rank: index + 1,
        recommendation_reason: `${item._sum.quantity} orders in last ${days} days`
      };
    })
  );

  res.json({
    restaurant_id,
    period_days: parseInt(days),
    recommendations,
    recommendation_type: 'popular'
  });
 } catch (error) {
  console.error('Popular recommendations error:', error);
  res.status(500).json({ error: 'Failed to get popular items' });
 }
});

// 3. Get Recommendations Based on Current Time
app.get('/recommendations/time-based', async (req, res) => {
 const { restaurant_id, limit = 5 } = req.query;

 try {
  const currentHour = new Date().getHours();
  let timeCategory;

  if (currentHour >= 6 && currentHour < 11) {
    timeCategory = 'breakfast';
  } else if (currentHour >= 11 && currentHour < 16) {
    timeCategory = 'lunch';
  } else if (currentHour >= 16 && currentHour < 20) {
    timeCategory = 'snacks';
  } else {
    timeCategory = 'dinner';
  }

  // Get items popular at this time in the past
  const timeBasedOrders = await db.orders.findMany({
    where: {
      restaurant_id,
      status: 'completed',
      placed_at: {
        gte: new Date(Date.now() - 30 * 24 * 60 * 60 * 1000) // Last 30 days
      }
    },
    include: {
      items: {
        include: {
```

```
          menu_item: true
        }
      }
    }
  });

  // Filter orders by hour range
  const relevantOrders = timeBasedOrders.filter(order => {
    const orderHour = order.placed_at.getHours();
    return Math.abs(orderHour - currentHour) <= 2; // Within 2 hours
  });

  // Count item frequency during this time
  const itemCounts = {};
  relevantOrders.forEach(order => {
    order.items.forEach(item => {
      if (!itemCounts[item.item_id]) {
        itemCounts[item.item_id] = {
          count: 0,
          item: item.menu_item
        };
      }
      itemCounts[item.item_id].count += item.quantity;
    });
  });

  // Sort and get top items
  const recommendations = Object.values(itemCounts)
    .sort((a, b) => b.count - a.count)
    .slice(0, parseInt(limit))
    .map((item, index) => ({
      ...item.item,
      popularity_score: item.count,
      rank: index + 1,
      time_category: timeCategory,
      recommendation_reason: `Popular at ${timeCategory} time`
    }));

  res.json({
    restaurant_id,
    current_time_category: timeCategory,
    recommendations,
    recommendation_type: 'time_based'
  });
 } catch (error) {
  console.error('Time-based recommendations error:', error);
  res.status(500).json({ error: 'Failed to get time-based recommendations' });
 }
});
```

```
// 4. Get Location-based Recommendations
app.get('/recommendations/location-based', verifyUser, async (req, res) => {
  const { restaurant_id, location_type, location_id, limit = 5 } = req.query;
  const userId = req.userId;

  try {
    // Get orders from similar locations
    const locationOrders = await db.orders.findMany({
      where: {
        restaurant_id,
        location_type,
        status: 'completed',
        placed_at: {
          gte: new Date(Date.now() - 30 * 24 * 60 * 60 * 1000) // Last 30 days
        }
      },
      include: {
        items: {
          include: {
            menu_item: true
          }
        }
      }
    });

    // If specific location_id provided, prioritize exact matches
    let exactLocationOrders = [];
    let similarLocationOrders = [];

    locationOrders.forEach(order => {
      if (order.location_id === location_id) {
        exactLocationOrders.push(order);
      } else {
        similarLocationOrders.push(order);
      }
    });

    // Count item frequency
    const itemCounts = {};
    const ordersToAnalyze = exactLocationOrders.length > 0 ?
      exactLocationOrders : similarLocationOrders;

    ordersToAnalyze.forEach(order => {
      order.items.forEach(item => {
        if (!itemCounts[item.item_id]) {
          itemCounts[item.item_id] = {
            count: 0,
            exact_location_count: 0,
```

```
          item: item.menu_item
        };
      }
      itemCounts[item.item_id].count += item.quantity;

      if (order.location_id === location_id) {
        itemCounts[item.item_id].exact_location_count += item.quantity;
      }
    });
  });

  // Exclude items user has ordered recently to encourage variety
  const recentUserOrders = await db.orders.findMany({
    where: {
      user_id: userId,
      restaurant_id,
      placed_at: {
        gte: new Date(Date.now() - 7 * 24 * 60 * 60 * 1000) // Last 7 days
      }
    },
    include: { items: true }
  });

  const recentItemIds = new Set();
  recentUserOrders.forEach(order => {
    order.items.forEach(item => {
      recentItemIds.add(item.item_id);
    });
  });

  // Sort and filter recommendations
  const recommendations = Object.values(itemCounts)
    .filter(item => !recentItemIds.has(item.item.id)) // Exclude recently ordered
    .sort((a, b) => {
      // Prioritize exact location matches
      if (a.exact_location_count !== b.exact_location_count) {
        return b.exact_location_count - a.exact_location_count;
      }
      return b.count - a.count;
    })
    .slice(0, parseInt(limit))
    .map((item, index) => ({
      ...item.item,
      popularity_score: item.count,
      location_popularity: item.exact_location_count,
      rank: index + 1,
      recommendation_reason: item.exact_location_count > 0
        ? `Popular at ${location_type} ${location_id}`
        : `Popular at similar ${location_type} locations`
```

```
    }));

    res.json({
      user_id: userId,
      restaurant_id,
      location: { type: location_type, id: location_id },
      recommendations,
      recommendation_type: 'location_based'
    });
  } catch (error) {
    console.error('Location-based recommendations error:', error);
    res.status(500).json({ error: 'Failed to get location-based recommendations' });
  }
});

// 5. Get "Complete Your Meal" Recommendations (Frequently Bought Together)
app.get('/recommendations/complete-meal', verifyUser, async (req, res) => {
  const { restaurant_id, current_items, limit = 3 } = req.query;

  if (!current_items) {
    return res.status(400).json({ error: 'current_items parameter required' });
  }

  try {
    const currentItemIds = current_items.split(',');

    // Find orders that contain any of the current items
    const relatedOrders = await db.orders.findMany({
      where: {
        restaurant_id,
        status: 'completed',
        items: {
          some: {
            item_id: { in: currentItemIds }
          }
        }
      },
      include: {
        items: {
          include: {
            menu_item: true
          }
        }
      },
      take: 1000 // Limit for performance
    });

    // Count co-occurrence of items
    const coOccurrence = {};
```

```
    relatedOrders.forEach(order => {
      const orderItemIds = order.items.map(item => item.item_id);
      const hasCurrentItem = currentItemIds.some(id => orderItemIds.includes(id));

      if (hasCurrentItem) {
        order.items.forEach(item => {
          if (!currentItemIds.includes(item.item_id)) {
            if (!coOccurrence[item.item_id]) {
              coOccurrence[item.item_id] = {
                count: 0,
                item: item.menu_item,
```