Technical Design Document

Restaurant OS - Phase 1

Version: 1.0
Last Updated: December 2024
Tech Team Size: 5 (AI-Assisted Development)
Timeline: 6-9 months
Stack: React + Node.js + PostgreSQL

---

1. Executive Summary

Architecture Philosophy

Build a **simple, modular, and well-documented** system that:

- Can be built by AI-assisted developers with limited experience
- Uses proven libraries and patterns
- Scales to 200+ restaurants without major rewrites
- Has clear boundaries between components

Key Technical Decisions

Decision	Choice	Rationale
Frontend Framework	React (Vite)	Best AI coding support, huge ecosystem, team preference
Backend	Node.js (Express)	JavaScript full-stack, easier for junior devs, fast iteration
Database	PostgreSQL	Proven relational model, free, excellent multi-tenancy support
Real-time	Socket.io	Battle-tested, fallbacks to polling, AI-friendly docs
Hosting	Hostinger VPS	Affordable (₹500-2000/mo), full control
Image Storage	Local filesystem (Phase 1), Cloudinary (Phase 2)	Simple, migrate later
Authentication	JWT + bcrypt	Standard, secure, stateless

---

2. System Architecture

2.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                          USERS                                  │
│  ┌──────────────────┐     ┌──────────────────┐                  │
│  │ Restaurant       │     │ Customers        │                  │
│  │ Admin/Staff      │     │ (QR Scanners)    │                  │
│  └────────┬─────────┘     └────────┬─────────┘                  │
└───────────┼────────────────────────┼────────────────────────────┘
            │                        │
            │                        │
┌───────────▼────────────────────────▼────────────────────────────┐
│                    PRESENTATION LAYER                           │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ Frontend Apps (React)                                    │   │
│  │ ├── Admin Dashboard (admin.yourapp.com)                  │   │
│  │ ├── Customer QR App (yourapp.com/order/:restaurant/:table)│  │
│  │ └── Marketing Site (yourapp.com)                         │   │
│  └──────────────────────────────────────────────────────────┘   │
└───────────┬─────────────────────────────────────────────────────┘
            │
            │ HTTPS/WSS
            │
┌───────────▼─────────────────────────────────────────────────────┐
│                    APPLICATION LAYER                            │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ API Gateway / Load Balancer (Nginx)                      │   │
│  └─────────────────────────┬────────────────────────────────┘   │
│                            │                                    │
│  ┌─────────────────────────▼────────────────────────────────┐   │
│  │ Backend Server (Node.js + Express)                       │   │
│  │ ├── REST API Endpoints                                   │   │
│  │ ├── WebSocket Server (Socket.io)                         │   │
│  │ ├── Authentication Middleware                            │   │
│  │ └── Business Logic Layer                                 │   │
│  └─────────────────────────┬────────────────────────────────┘   │
└────────────────────────────┼────────────────────────────────────┘
                             │
                             │ SQL
                             │
┌────────────────────────────▼────────────────────────────────────┐
│                       DATA LAYER                                │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ PostgreSQL Database                                      │   │
│  │  ├── Multi-tenant data model                │  │
│  │  ├── Indexed for performance                │  │
│  │  └── Automated backups                      │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                         │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ File Storage                       │  │
│  │  ├── Menu item images (local /uploads)        │  │
│  │  ├── Restaurant logos                   │  │
│  │  └── QR code images                     │  │
│  └───────────────────────────────────────────────────────────┘  │
└───────────────────────────────────────────────────────────────────┘
```

2.2 Component Responsibilities

Frontend (React)

- **Admin Dashboard:** Restaurant management interface
- **Customer App:** QR ordering interface
- **Marketing Site:** Landing pages

Key Libraries:

- React Router v6 (routing)
- TanStack Query / React Query (data fetching + caching)
- Zustand or Context API (state management)
- Tailwind CSS (styling)
- Socket.io-client (real-time updates)
- React Hook Form (forms)
- Zod (validation)

Backend (Node.js + Express)

- REST API endpoints
- WebSocket server for real-time updates
- Business logic (orders, menu, QR generation)
- Multi-tenant access control
- File upload handling
- Authentication & authorization

Key Libraries:

- Express.js (web framework)
- Socket.io (WebSockets)
- pg (PostgreSQL client)
- bcrypt (password hashing)
- jsonwebtoken (JWT)
- multer (file uploads)
- qrcode (QR generation)
- cors (CORS handling)
- helmet (security headers)
- express-rate-limit (rate limiting)

Database (PostgreSQL)

- All structured data storage
- Multi-tenant with restaurant_id isolation
- Transactions for order consistency
- Indexes for performance

---

3. Database Design

3.1 Schema Overview

Design Principles:

- Multi-tenant from day 1 (every table has `restaurant_id`)
- Soft deletes (`deleted_at` column)
- Audit trails (`created_at`, `updated_at`)
- UUID primary keys for security

3.2 Complete Database Schema

```sql
```

```sql
-- ================================================
-- CORE ENTITIES
-- ================================================

-- Restaurants (tenants)
CREATE TABLE restaurants (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    slug VARCHAR(255) UNIQUE NOT NULL, -- URL-friendly name
    email VARCHAR(255) UNIQUE NOT NULL,
    phone VARCHAR(20),

    -- Address
    address_line1 VARCHAR(255),
    address_line2 VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(100),
    pincode VARCHAR(10),

    -- Branding
    logo_url VARCHAR(500),
    primary_color VARCHAR(7) DEFAULT '#000000',
    secondary_color VARCHAR(7) DEFAULT '#FFFFFF',

    -- Settings
    currency VARCHAR(3) DEFAULT 'INR',
    tax_percentage DECIMAL(5,2) DEFAULT 0.00,
    timezone VARCHAR(50) DEFAULT 'Asia/Kolkata',

    -- Business Info
    operating_hours JSONB, -- {mon: {open: "09:00", close: "22:00"}, ...}

    -- Subscription
    plan VARCHAR(50) DEFAULT 'starter', -- starter, pro, multi-outlet
    subscription_status VARCHAR(20) DEFAULT 'trial', -- trial, active, suspended
    subscription_expires_at TIMESTAMP,

    -- Timestamps
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    deleted_at TIMESTAMP NULL
);

CREATE INDEX idx_restaurants_slug ON restaurants(slug);
CREATE INDEX idx_restaurants_email ON restaurants(email);
```

```sql
-- ================================================
-- USERS & AUTHENTICATION
-- ================================================

CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    restaurant_id UUID REFERENCES restaurants(id) ON DELETE CASCADE,

    -- Auth
    email VARCHAR(255) NOT NULL,
    phone VARCHAR(20),
    password_hash VARCHAR(255) NOT NULL,

    -- Profile
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    role VARCHAR(50) DEFAULT 'owner', -- owner, manager, waiter, kitchen

    -- Status
    is_active BOOLEAN DEFAULT TRUE,
    last_login_at TIMESTAMP,

    -- Timestamps
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    deleted_at TIMESTAMP NULL,

    UNIQUE(restaurant_id, email)
);

CREATE INDEX idx_users_restaurant ON users(restaurant_id);
CREATE INDEX idx_users_email ON users(email);

-- ================================================
-- MENU MANAGEMENT
-- ================================================

-- Menu Categories
CREATE TABLE menu_categories (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    restaurant_id UUID NOT NULL REFERENCES restaurants(id) ON DELETE CASCADE,

    name VARCHAR(255) NOT NULL,
    description TEXT,
    display_order INT DEFAULT 0,
    is_active BOOLEAN DEFAULT TRUE,
    -- Timestamps
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    deleted_at TIMESTAMP NULL,

    UNIQUE(restaurant_id, name)
);

CREATE INDEX idx_categories_restaurant ON menu_categories(restaurant_id);
CREATE INDEX idx_categories_active ON menu_categories(restaurant_id, is_active);

-- Menu Items
CREATE TABLE menu_items (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    restaurant_id UUID NOT NULL REFERENCES restaurants(id) ON DELETE CASCADE,
    category_id UUID REFERENCES menu_categories(id) ON DELETE SET NULL,

    -- Basic Info
    name VARCHAR(255) NOT NULL,
    description TEXT,
    image_url VARCHAR(500),

    -- Pricing
    base_price DECIMAL(10,2) NOT NULL,

    -- Attributes
    item_type VARCHAR(20) DEFAULT 'veg', -- veg, non-veg, egg, vegan
    spice_level INT DEFAULT 0, -- 0-3
    preparation_time_minutes INT DEFAULT 15,

    -- Tags
    tags TEXT[], -- ['bestseller', 'new', 'seasonal']
    allergens TEXT[], -- ['nuts', 'dairy', 'gluten']

    -- Availability
    is_available BOOLEAN DEFAULT TRUE,
    available_from TIME,
    available_to TIME,

    -- Display
    display_order INT DEFAULT 0,

    -- Timestamps
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    deleted_at TIMESTAMP NULL
);
```

```
CREATE INDEX idx_items_restaurant ON menu_items(restaurant_id);
CREATE INDEX idx_items_category ON menu_items(category_id);
CREATE INDEX idx_items_available ON menu_items(restaurant_id, is_available);

-- Item Variants (sizes, bases, etc.)
CREATE TABLE item_variants (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  item_id UUID NOT NULL REFERENCES menu_items(id) ON DELETE CASCADE,

  variant_group VARCHAR(100) NOT NULL, -- "Size", "Base", "Style"
  option_name VARCHAR(100) NOT NULL, -- "Small", "Thin Crust", "Grilled"
  price_adjustment DECIMAL(10,2) DEFAULT 0.00, -- +/- from base price

  is_available BOOLEAN DEFAULT TRUE,
  display_order INT DEFAULT 0,

  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_variants_item ON item_variants(item_id);

-- Addon Groups (for customization)
CREATE TABLE addon_groups (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  restaurant_id UUID NOT NULL REFERENCES restaurants(id) ON DELETE CASCADE,

  name VARCHAR(100) NOT NULL, -- "Toppings", "Sauces", "Extras"
  selection_type VARCHAR(20) DEFAULT 'multi', -- single, multi
  is_required BOOLEAN DEFAULT FALSE,
  max_selections INT, -- NULL = unlimited

  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_addon_groups_restaurant ON addon_groups(restaurant_id);

-- Addons
CREATE TABLE addons (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  addon_group_id UUID NOT NULL REFERENCES addon_groups(id) ON DELETE CASCADE,

  name VARCHAR(100) NOT NULL,
  price DECIMAL(10,2) DEFAULT 0.00,
  is_available BOOLEAN DEFAULT TRUE,
```

```sql
    display_order INT DEFAULT 0,

    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_addons_group ON addons(addon_group_id);

-- Item-AddonGroup mapping (which items have which addon groups)
CREATE TABLE item_addon_groups (
    item_id UUID REFERENCES menu_items(id) ON DELETE CASCADE,
    addon_group_id UUID REFERENCES addon_groups(id) ON DELETE CASCADE,

    PRIMARY KEY (item_id, addon_group_id)
);

-- ============================================
-- TABLE MANAGEMENT
-- ============================================

CREATE TABLE restaurant_tables (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    restaurant_id UUID NOT NULL REFERENCES restaurants(id) ON DELETE CASCADE,

    table_number VARCHAR(50) NOT NULL, -- "T1", "Patio-3", "Counter-A"
    section VARCHAR(100), -- "Indoor", "Outdoor", "Rooftop"
    capacity INT, -- number of seats

    -- QR Code
    qr_code_url VARCHAR(500), -- URL of generated QR image
    qr_unique_token VARCHAR(100) UNIQUE, -- unique token in QR URL

    -- Status
    is_active BOOLEAN DEFAULT TRUE,
    current_status VARCHAR(20) DEFAULT 'available', -- available, occupied, reserved

    -- Timestamps
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    deleted_at TIMESTAMP NULL,

    UNIQUE(restaurant_id, table_number)
);

CREATE INDEX idx_tables_restaurant ON restaurant_tables(restaurant_id);
CREATE INDEX idx_tables_token ON restaurant_tables(qr_unique_token);
```

```sql
-- ================================================
-- ORDER MANAGEMENT
-- ================================================

CREATE TABLE orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    restaurant_id UUID NOT NULL REFERENCES restaurants(id) ON DELETE CASCADE,
    table_id UUID REFERENCES restaurant_tables(id) ON DELETE SET NULL,

    -- Order Info
    order_number VARCHAR(50) UNIQUE NOT NULL, -- Human-readable: ORD-20241204-001
    order_type VARCHAR(20) DEFAULT 'dine-in', -- dine-in, takeaway, delivery

    -- Status
    status VARCHAR(20) DEFAULT 'pending', -- pending, preparing, ready, served, completed, cancelled

    -- Special requests
    special_instructions TEXT,

    -- Amounts
    subtotal DECIMAL(10,2) NOT NULL,
    tax_amount DECIMAL(10,2) DEFAULT 0.00,
    discount_amount DECIMAL(10,2) DEFAULT 0.00,
    total_amount DECIMAL(10,2) NOT NULL,

    -- Payment (Phase 2)
    payment_status VARCHAR(20) DEFAULT 'pending', -- pending, paid, failed
    payment_method VARCHAR(50),

    -- Timestamps
    placed_at TIMESTAMP DEFAULT NOW(),
    preparing_started_at TIMESTAMP,
    ready_at TIMESTAMP,
    served_at TIMESTAMP,
    completed_at TIMESTAMP,
    cancelled_at TIMESTAMP,
    cancellation_reason TEXT,

    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_orders_restaurant ON orders(restaurant_id);
CREATE INDEX idx_orders_table ON orders(table_id);
CREATE INDEX idx_orders_status ON orders(restaurant_id, status);
CREATE INDEX idx_orders_placed_at ON orders(placed_at);
```

```sql
-- Order Items (line items)
CREATE TABLE order_items (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_id UUID NOT NULL REFERENCES orders(id) ON DELETE CASCADE,
    item_id UUID REFERENCES menu_items(id) ON DELETE SET NULL,

    -- Snapshot data (in case menu item changes/deleted)
    item_name VARCHAR(255) NOT NULL,
    item_price DECIMAL(10,2) NOT NULL,

    -- Quantity
    quantity INT NOT NULL DEFAULT 1,

    -- Variant selected
    variant_group VARCHAR(100),
    variant_option VARCHAR(100),
    variant_price_adjustment DECIMAL(10,2) DEFAULT 0.00,

    -- Special instructions for this item
    special_instructions TEXT,

    -- Amounts
    item_total DECIMAL(10,2) NOT NULL, -- (base_price + variant_adj + addons) * qty

    created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_order_items_order ON order_items(order_id);

-- Order Item Addons
CREATE TABLE order_item_addons (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_item_id UUID NOT NULL REFERENCES order_items(id) ON DELETE CASCADE,
    addon_id UUID REFERENCES addons(id) ON DELETE SET NULL,

    -- Snapshot
    addon_name VARCHAR(100) NOT NULL,
    addon_price DECIMAL(10,2) NOT NULL,

    created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_order_item_addons_item ON order_item_addons(order_item_id);

-- ============================================
-- ANALYTICS & REPORTING (Phase 1 - basic)
-- ============================================
-- Can be queried from orders table directly
-- Phase 2: Create materialized views for performance

-- ================================================
-- SYSTEM TABLES
-- ================================================

-- Audit Log (optional, Phase 2)
CREATE TABLE audit_logs (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  restaurant_id UUID REFERENCES restaurants(id) ON DELETE CASCADE,
  user_id UUID REFERENCES users(id) ON DELETE SET NULL,

  entity_type VARCHAR(50), -- 'menu_item', 'order', etc.
  entity_id UUID,
  action VARCHAR(50), -- 'created', 'updated', 'deleted'
  old_data JSONB,
  new_data JSONB,

  created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_audit_restaurant ON audit_logs(restaurant_id);
CREATE INDEX idx_audit_entity ON audit_logs(entity_type, entity_id);
```

3.3 Multi-Tenancy Strategy

Every query MUST include `restaurant_id` filter to ensure data isolation.

Example Safe Query:

```javascript
// CORRECT - filters by restaurant_id
const items = await db.query(
  'SELECT * FROM menu_items WHERE restaurant_id = $1',
  [restaurantId]
);

// DANGEROUS - no filter, could expose all data
const items = await db.query('SELECT * FROM menu_items');
```

Middleware Pattern:

```javascript
```

```
// Extract restaurant from JWT token
function extractRestaurant(req, res, next) {
  const token = req.headers.authorization?.split(' ')[1];
  const decoded = jwt.verify(token, SECRET);
  req.restaurantId = decoded.restaurantId;
  req.userId = decoded.userId;
  next();
}

// All protected routes use this
app.use('/api/*', extractRestaurant);
```

---

4. API Design

4.1 API Structure

Base URL: `https://api.yourapp.com/v1`

Authentication: JWT Bearer token in `Authorization` header

Response Format:

json

```json
{
  "success": true,
  "data": { ... },
  "message": "Operation successful",
  "timestamp": "2024-12-04T10:30:00Z"
}
```

Error Format:

json

```
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": [
      {"field": "email", "message": "Invalid email format"}
    ]
  },
  "timestamp": "2024-12-04T10:30:00Z"
}
```

4.2 API Endpoints (Complete)

Authentication

```
POST   /auth/register          - Register new restaurant
POST   /auth/login             - Login user
POST   /auth/refresh           - Refresh JWT token
POST   /auth/logout            - Logout (invalidate token)
POST   /auth/forgot-password   - Send reset email
POST   /auth/reset-password    - Reset password with token
```

Restaurant Management

```
GET    /restaurant             - Get current restaurant details
PUT    /restaurant             - Update restaurant details
PUT    /restaurant/branding    - Update logo, colors
GET    /restaurant/settings    - Get all settings
PUT    /restaurant/settings    - Update settings
```

Menu Categories

```
GET    /categories             - List all categories (with items count)
POST   /categories             - Create category
GET    /categories/:id         - Get category details
PUT    /categories/:id         - Update category
DELETE /categories/:id         - Soft delete category
PUT    /categories/reorder     - Update display order of multiple
```

Menu Items

```
GET    /items                   - List items (paginated, filterable)
                                Query params: ?category=uuid&search=pizza&type=veg
POST   /items                   - Create item
GET    /items/:id               - Get item details (with variants, addons)
PUT    /items/:id               - Update item
DELETE /items/:id               - Soft delete item
PUT    /items/:id/availability  - Toggle availability
POST   /items/:id/image         - Upload item image
PUT    /items/reorder           - Bulk reorder items
```

Item Variants

```
GET    /items/:itemId/variants     - List variants for item
POST   /items/:itemId/variants     - Add variant
PUT    /variants/:id               - Update variant
DELETE /variants/:id               - Delete variant
```

Addon Groups & Addons

```
GET    /addon-groups               - List addon groups
POST   /addon-groups               - Create addon group
GET    /addon-groups/:id           - Get group with addons
PUT    /addon-groups/:id           - Update group
DELETE /addon-groups/:id           - Delete group

POST   /addon-groups/:id/addons    - Add addon to group
PUT    /addons/:id                 - Update addon
DELETE /addons/:id                 - Delete addon

POST   /items/:itemId/addon-groups - Link addon group to item
DELETE /items/:itemId/addon-groups/:groupId - Unlink
```

Tables

```
GET    /tables                     - List all tables
POST   /tables                     - Create table
GET    /tables/:id                 - Get table details
PUT    /tables/:id                 - Update table
DELETE /tables/:id                 - Delete table
POST   /tables/:id/generate-qr     - Generate QR code
GET    /tables/download-qrs        - Download all QRs as ZIP
```

Orders (Admin)

```
GET    /orders                    - List orders (filterable, paginated)
                           Query: ?status=pending&table=uuid&date=2024-12-04
GET    /orders/:id                - Get order details
PUT    /orders/:id/status         - Update order status
PUT    /orders/:id/cancel         - Cancel order
GET    /orders/stats              - Get statistics (today, bestsellers, etc.)
```

Orders (Customer)

```
GET    /public/menu/:restaurantSlug/:tableToken  - Get menu for QR scan
POST   /public/orders                            - Place order (no auth)
GET    /public/orders/:id                        - Get order status (no auth)
```

Analytics

```
GET    /analytics/dashboard       - Dashboard summary
GET    /analytics/sales           - Sales over time
GET    /analytics/items           - Item performance
GET    /analytics/tables          - Table performance
GET    /analytics/export          - Export data as CSV
```

4.3 WebSocket Events

Connection:

```javascript
// Client connects
socket.emit('join_restaurant', { restaurantId });

// Server acknowledges
socket.on('joined', { message: 'Connected to restaurant' });
```

Events:

```javascript
```

```
// Server → Client (broadcast to all staff of restaurant)
'new_order'          - New order placed
'order_updated'      - Order status changed
'menu_updated'       - Menu item changed

// Client → Server
'update_order'       - Staff updates order status
'ping'               - Keep-alive
```

---

5. Frontend Architecture

5.1 Project Structure

```
frontend/
├── admin/                    # Admin Dashboard
│   ├── src/
│   │   ├── components/
│   │   │   ├── common/       # Reusable components
│   │   │   │   ├── Button.jsx
│   │   │   │   ├── Input.jsx
│   │   │   │   ├── Modal.jsx
│   │   │   │   └── Table.jsx
│   │   │   ├── menu/         # Menu-specific components
│   │   │   │   ├── CategoryList.jsx
│   │   │   │   ├── ItemForm.jsx
│   │   │   │   └── ItemCard.jsx
│   │   │   ├── orders/
│   │   │   │   ├── OrderCard.jsx
│   │   │   │   ├── OrderDetail.jsx
│   │   │   │   └── StatusBadge.jsx
│   │   │   └── tables/
│   │   │       ├── TableGrid.jsx
│   │   │       └── QRViewer.jsx
│   │   ├── pages/
│   │   │   ├── auth/
│   │   │   │   ├── Login.jsx
│   │   │   │   └── Register.jsx
│   │   │   ├── dashboard/
│   │   │   │   └── Dashboard.jsx
│   │   │   ├── menu/
│   │   │   │   ├── MenuList.jsx
│   │   │   │   ├── ItemEdit.jsx
│   │   │   │   └── CategoryManager.jsx
│   │   │   ├── orders/
│   │   │   │   ├── OrdersLive.jsx
│   │   │   │   └── OrderHistory.jsx
│   │   │   ├── tables/
│   │   │   │   └── TablesManager.jsx
│   │   │   └── analytics/
│   │   │       └── Analytics.jsx
│   │   ├── hooks/           # Custom hooks
│   │   │   ├── useAuth.js
│   │   │   ├── useMenu.js
│   │   │   ├── useOrders.js
│   │   │   └── useSocket.js
│   │   ├── services/        # API calls
│   │   │   ├── api.js       # Axios instance
│   │   │   ├── auth.service.js
│   │   │   ├── menu.service.js
│   │   │   ├── order.service.js
│   │   │   └── table.service.js
│   │   ├── store/           # State management (Zustand)
│   │   │   ├── authStore.js
│   │   │   ├── menuStore.js
│   │   │   └── orderStore.js
│   │   ├── utils/
│   │   │   ├── constants.js
│   │   │   ├── formatters.js
│   │   │   └── validators.js
│   │   ├── App.jsx
│   │   └── main.jsx
│   └── package.json
│
├── customer/                # Customer QR Ordering App
│   ├── src/
│   │   ├── components/
│   │   │   ├── MenuBrowser.jsx
│   │   │   ├── ItemCard.jsx
│   │   │   ├── ItemDetail.jsx
│   │   │   ├── Cart.jsx
│   │   │   └── OrderStatus.jsx
│   │   ├── pages/
│   │   │   ├── TableLanding.jsx
│   │   │   ├── MenuView.jsx
│   │   │   └── OrderConfirm.jsx
│   │   ├── hooks/
│   │   │   ├── useMenu.js
│   │   │   └── useCart.js
│   │   ├── services/
│   │   │   └── customer.service.js
│   │   │
│   │   ├── store/
│   │   │   └── cartStore.js
│   │   ├── App.jsx
│   │   └── main.jsx
│   └── package.json
│
└── marketing/              # Landing Page
    ├── src/
    │   ├── components/
    │   │   ├── Hero.jsx
    │   │   ├── Features.jsx
    │   │   ├── Pricing.jsx
    │   │   └── FAQ.jsx
    │   ├── pages/
    │   │   ├── Home.jsx
    │   │   └── ForCafes.jsx
    │   ├── App.jsx
    │   └── main.jsx
    └── package.json
```

5.2 State Management Strategy

Use Zustand (simple, AI-friendly):

```
javascript
```

```
// Example: orderStore.js
import { create } from 'zustand';

const useOrderStore = create((set, get) => ({
  // State
  orders: [],
  loading: false,
  error: null,

  // Actions
  setOrders: (orders) => set({ orders }),

  addOrder: (order) => set((state) => ({
    orders: [order, ...state.orders]
  })),

  updateOrderStatus: (orderId, status) => set((state) => ({
    orders: state.orders.map(o =>
      o.id === orderId ? { ...o, status } : o
    )
  })),

  removeOrder: (orderId) => set((state) => ({
    orders: state.orders.filter(o => o.id !== orderId)
  })),
}));

export default useOrderStore;
```

When to use:

- Global state: User auth, restaurant data
- Shared state: Cart, orders list
- Real-time data: Socket updates

When NOT to use:

- Form state (use React Hook Form)
- Server state (use React Query)
- Component-local state (use useState)

5.3 Data Fetching with React Query

```
javascript
```

```
// hooks/useMenu.js
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import menuService from '../services/menu.service';

export function useMenuItems() {
  return useQuery({
    queryKey: ['menu-items'],
    queryFn: menuService.getItems,
    staleTime: 30000, // 30 seconds
  });
}

export function useCreateMenuItem() {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: menuService.createItem,
    onSuccess: () => {
      // Invalidate and refetch
      queryClient.invalidateQueries({ queryKey: ['menu-items'] });
    },
  });
}
```

Benefits:

- Automatic caching
- Background refetching
- Optimistic updates
- Error handling

5.4 Real-time Updates with Socket.io

javascript

```
// hooks/useSocket.js
import { useEffect } from 'react';
import { io } from 'socket.io-client';
import useOrderStore from '../store/orderStore';

export function useSocket(restaurantId) {
  useEffect(() => {
    const socket = io('https://api.yourapp.com', {
      auth: { token: localStorage.getItem('token') }
    });

    socket.emit('join_restaurant', { restaurantId });

    socket.on('new_order', (order) => {
      useOrderStore.getState().addOrder(order);
      // Show notification
      new Notification('New Order!', {
        body: `Table ${order.table_number}`,
      });
    });

    socket.on('order_updated', ({ orderId, status }) => {
      useOrderStore.getState().updateOrderStatus(orderId, status);
    });

    return () => socket.disconnect();
  }, [restaurantId]);
}

// Usage in component
function OrdersLive() {
  const { restaurantId } = useAuth();
  useSocket(restaurantId); // Automatically handles real-time updates

  // ... rest of component
}
```

---

6. Backend Architecture

6.1 Project Structure

```
backend/
├── src/
```

```
│   ├── config/
│   │   ├── database.js       # PostgreSQL connection
│   │   ├── jwt.js            # JWT config
│   │   └── env.js            # Environment variables
│   ├── middleware/
│   │   ├── auth.middleware.js       # JWT verification
│   │   ├── validate.middleware.js   # Request validation
│   │   ├── error.middleware.js      # Error handling
│   │   └── upload.middleware.js     # File upload (multer)
│   ├── models/               # Database models (if using ORM)
│   │   ├── Restaurant.js
│   │   ├── User.js
│   │   ├── MenuItem.js
│   │   └── Order.js
│   ├── routes/
│   │   ├── auth.routes.js
│   │   ├── restaurant.routes.js
│   │   ├── menu.routes.js
│   │   ├── table.routes.js
│   │   ├── order.routes.js
│   │   ├── customer.routes.js       # Public routes
│   │   └── analytics.routes.js
│   ├── controllers/
│   │   ├── auth.controller.js
│   │   ├── menu.controller.js
│   │   ├── table.controller.js
│   │   ├── order.controller.js
│   │   └── analytics.controller.js
│   ├── services/
│   │   ├── auth.service.js
│   │   ├── menu.service.js
│   │   ├── order.service.js
│   │   ├── qr.service.js            # QR generation
│   │   └── email.service.js         # Email (optional)
│   ├── utils/
│   │   ├── logger.js                # Winston logger
│   │   ├── errors.js                # Custom error classes
│   │   └── helpers.js
│   ├── socket/
│   │   └── socket.handler.js        # Socket.io logic
│   ├── app.js                       # Express app setup
│   └── server.js                    # Entry point
├── uploads/                         # Uploaded files
├── .env
├── package.json
└── README.md
```

6.2 Example Controller Pattern

javascript

```
// controllers/menu.controller.js
const menuService = require('../services/menu.service');
const { AppError } = require('../utils/errors');

class MenuController {
  // Get all items for a restaurant
  async getItems(req, res, next) {
    try {
      const { restaurantId } = req; // From auth middleware
      const { category, search, type } = req.query;

      const items = await menuService.getItems(restaurantId, {
        category,
        search,
        type
      });

      res.json({
        success: true,
        data: items
      });
    } catch (error) {
      next(error);
    }
  }

  // Create new item
  async createItem(req, res, next) {
    try {
      const { restaurantId } = req;
      const itemData = req.body;

      // Validate
      if (!itemData.name || !itemData.base_price) {
        throw new AppError('Name and price are required', 400);
      }

      const newItem = await menuService.createItem(restaurantId, itemData);

      res.status(201).json({
        success: true,
        data: newItem,
        message: 'Item created successfully'
      });
    } catch (error) {
      next(error);
    }
  }

  // ... other methods
}

module.exports = new MenuController();
```

6.3 Example Service Pattern

```
javascript
```

```
// services/menu.service.js
const db = require('../config/database');

class MenuService {
  async getItems(restaurantId, filters = {}) {
    let query = `
      SELECT
        mi.*,
        mc.name as category_name,
        COUNT(oi.id) as times_ordered
      FROM menu_items mi
      LEFT JOIN menu_categories mc ON mi.category_id = mc.id
      LEFT JOIN order_items oi ON mi.id = oi.item_id
      WHERE mi.restaurant_id = $1 AND mi.deleted_at IS NULL
    `;

    const params = [restaurantId];
    let paramCount = 1;

    // Add filters
    if (filters.category) {
      paramCount++;
      query += ` AND mi.category_id = $${paramCount}`;
      params.push(filters.category);
    }

    if (filters.search) {
      paramCount++;
      query += ` AND mi.name ILIKE $${paramCount}`;
      params.push(`%${filters.search}%`);
    }

    if (filters.type) {
      paramCount++;
      query += ` AND mi.item_type = $${paramCount}`;
      params.push(filters.type);
    }

    query += ` GROUP BY mi.id, mc.name ORDER BY mi.display_order`;

    const result = await db.query(query, params);
    return result.rows;
  }

  async createItem(restaurantId, itemData) {
    const query = `
      INSERT INTO menu_items (
        restaurant_id, category_id, name, description,
        base_price, item_type, is_available
      )
      VALUES ($1, $2, $3, $4, $5, $6, $7)
      RETURNING *
    `;

    const values = [
      restaurantId,
      itemData.category_id,
      itemData.name,
      itemData.description || null,
      itemData.base_price,
      itemData.item_type || 'veg',
      itemData.is_available !== undefined ? itemData.is_available : true
    ];

    const result = await db.query(query, values);
    return result.rows[0];
  }

  // ... other methods
}

module.exports = new MenuService();
```

6.4 Authentication Middleware

javascript

```javascript
// middleware/auth.middleware.js
const jwt = require('jsonwebtoken');
const { AppError } = require('../utils/errors');

function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1]; // Bearer TOKEN

  if (!token) {
    return next(new AppError('Access token required', 401));
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.userId = decoded.userId;
    req.restaurantId = decoded.restaurantId;
    req.role = decoded.role;
    next();
  } catch (error) {
    return next(new AppError('Invalid or expired token', 403));
  }
}

function requireRole(roles) {
  return (req, res, next) => {
    if (!roles.includes(req.role)) {
      return next(new AppError('Insufficient permissions', 403));
    }
    next();
  };
}

module.exports = { authenticateToken, requireRole };
```

6.5 QR Code Generation

javascript

```
// services/qr.service.js
const QRCode = require('qrcode');
const fs = require('fs').promises;
const path = require('path');
const { v4: uuidv4 } = require('uuid');

class QRService {
  async generateTableQR(restaurant, table) {
    // Generate unique token
    const token = uuidv4();

    // Create URL
    const qrUrl = `https://yourapp.com/order/${restaurant.slug}/${token}`;

    // Generate QR code as PNG
    const qrPath = path.join(__dirname, '../../uploads/qr/', `${token}.png`);

    await QRCode.toFile(qrPath, qrUrl, {
      width: 500,
      margin: 2,
      color: {
        dark: restaurant.primary_color || '#000000',
        light: '#FFFFFF'
      }
    });

    return {
      token,
      url: qrUrl,
      imagePath: `/uploads/qr/${token}.png`
    };
  }

  async generateBulkQRs(restaurant, tables) {
    const results = [];

    for (const table of tables) {
      const qr = await this.generateTableQR(restaurant, table);
      results.push({ tableId: table.id, ...qr });
    }

    return results;
  }
}
```

```
module.exports = new QRService();
```

---

7. Deployment Strategy

7.1 Hosting Setup (Hostinger VPS)

Server Specs Recommendation:

- **Phase 1 (0-50 restaurants):** VPS 2 - 2 vCPU, 4GB RAM, 50GB SSD (₹699/month)
- **Phase 2 (50-200 restaurants):** VPS 4 - 4 vCPU, 8GB RAM, 100GB SSD (₹1,399/month)

Software Stack:

- Ubuntu 22.04 LTS
- Node.js 20 LTS
- PostgreSQL 15
- Nginx (reverse proxy + static files)
- PM2 (process manager)
- Let's Encrypt (free SSL)

7.2 Directory Structure on Server

```
/var/www/
├── restaurant-os/
│   ├── backend/           # Node.js backend
│   ├── admin-frontend/    # Admin dashboard (built)
│   ├── customer-frontend/ # Customer app (built)
│   ├── marketing-frontend/ # Landing page (built)
│   └── uploads/           # User uploads
└── logs/
```

7.3 Nginx Configuration

```
nginx
```

```nginx
# /etc/nginx/sites-available/restaurant-os

# Admin Dashboard
server {
    listen 80;
    server_name admin.yourapp.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name admin.yourapp.com;

    ssl_certificate /etc/letsencrypt/live/admin.yourapp.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/admin.yourapp.com/privkey.pem;

    root /var/www/restaurant-os/admin-frontend;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}

# Customer App + Marketing
server {
    listen 80;
    server_name yourapp.com www.yourapp.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name yourapp.com www.yourapp.com;

    ssl_certificate /etc/letsencrypt/live/yourapp.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/yourapp.com/privkey.pem;
    # Customer ordering
    location /order {
        root /var/www/restaurant-os/customer-frontend;
        try_files $uri $uri/ /index.html;
    }

    # Marketing site
    location / {
        root /var/www/restaurant-os/marketing-frontend;
        try_files $uri $uri/ /index.html;
    }

    # API
    location /api {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    # Uploads
    location /uploads {
        alias /var/www/restaurant-os/uploads;
        expires 7d;
    }
}

# API Server (optional separate subdomain)
server {
    listen 80;
    server_name api.yourapp.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name api.yourapp.com;

    ssl_certificate /etc/letsencrypt/live/api.yourapp.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/api.yourapp.com/privkey.pem;

    location / {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_cache_bypass $http_upgrade;
    }
}
```

7.4 PM2 Configuration

javascript

```javascript
// ecosystem.config.js
module.exports = {
  apps: [{
    name: 'restaurant-os-api',
    script: './src/server.js',
    instances: 2, // Use multiple cores
    exec_mode: 'cluster',
    env: {
      NODE_ENV: 'production',
      PORT: 5000
    },
    error_file: '/var/www/logs/api-error.log',
    out_file: '/var/www/logs/api-out.log',
    log_date_format: 'YYYY-MM-DD HH:mm:ss Z',
    merge_logs: true,
    autorestart: true,
    max_memory_restart: '500M',
  }]
};
```

PM2 Commands:

bash

```
# Start
pm2 start ecosystem.config.js

# Monitor
pm2 monit

# Logs
pm2 logs

# Restart
pm2 restart all

# Auto-start on server reboot
pm2 startup
pm2 save
```

7.5 Database Backups

```bash
# Automated daily backup script
#!/bin/bash
# /var/www/scripts/backup-db.sh

BACKUP_DIR="/var/backups/postgres"
DB_NAME="restaurant_os"
DATE=$(date +%Y%m%d_%H%M%S)

# Create backup
pg_dump $DB_NAME | gzip > $BACKUP_DIR/$DB_NAME-$DATE.sql.gz

# Keep only last 30 days
find $BACKUP_DIR -name "*.sql.gz" -mtime +30 -delete

echo "Backup completed: $DB_NAME-$DATE.sql.gz"
```

Cron Job (daily at 2 AM):

```bash
crontab -e

# Add:
0 2 * * * /var/www/scripts/backup-db.sh
```

7.6 Monitoring & Logging

Winston Logger Setup:

```javascript
// utils/logger.js
const winston = require('winston');

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.File({ filename: 'logs/error.log', level: 'error' }),
    new winston.transports.File({ filename: 'logs/combined.log' }),
  ],
});

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.simple(),
  }));
}

module.exports = logger;
```

Usage:

```javascript
logger.info('Order placed', { orderId: '123', restaurantId: 'abc' });
logger.error('Database error', { error: err.message, stack: err.stack });
```

---

8. Security Considerations

8.1 Authentication & Authorization

Password Security:

- Use bcrypt with salt rounds = 12
- Minimum 8 characters

- Store only hashed passwords

JWT Security:

- Short expiry: 24 hours
- Refresh token mechanism
- Store secret in environment variables
- Include: userId, restaurantId, role in payload

Example:

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

// Hash password
const hashedPassword = await bcrypt.hash(password, 12);

// Generate JWT
const token = jwt.sign(
  { userId, restaurantId, role },
  process.env.JWT_SECRET,
  { expiresIn: '24h' }
);

// Verify password
const isValid = await bcrypt.compare(password, hashedPassword);
```

8.2 Data Protection

Multi-tenancy Isolation:

- ALWAYS filter by `restaurant_id`
- Database-level row security (Phase 2)
- No shared data between restaurants

SQL Injection Prevention:

- Use parameterized queries
- Never concatenate user input into SQL

```javascript
```

```
// DANGEROUS
const query = `SELECT * FROM orders WHERE id = '${req.params.id}'`;

// SAFE
const query = 'SELECT * FROM orders WHERE id = $1 AND restaurant_id = $2';
const result = await db.query(query, [req.params.id, req.restaurantId]);
```

XSS Prevention:

- Sanitize all user inputs
- Use helmet.js for security headers
- Set Content-Security-Policy

CSRF Protection:

- Not needed for pure API (no cookies)
- Use CORS properly

8.3 API Security

Rate Limiting:

```javascript
const rateLimit = require('express-rate-limit');

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // 100 requests per window
  message: 'Too many requests, please try again later'
});

app.use('/api/', limiter);
```

CORS Configuration:

```javascript
```

```
const cors = require('cors');

app.use(cors({
  origin: [
    'https://yourapp.com',
    'https://admin.yourapp.com'
  ],
  credentials: true
}));
```

Input Validation:

```javascript
const { body, validationResult } = require('express-validator');

// Validation middleware
const validateMenuItem = [
  body('name').trim().notEmpty().isLength({ max: 255 }),
  body('base_price').isFloat({ min: 0 }),
  body('item_type').isIn(['veg', 'non-veg', 'egg', 'vegan']),

  (req, res, next) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    next();
  }
];

router.post('/items', validateMenuItem, menuController.createItem);
```

8.4 File Upload Security

```javascript
```

```
const multer = require('multer');
const path = require('path');

// Configure multer
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/items/');
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    cb(null, uniqueSuffix + path.extname(file.originalname));
  }
});

const upload = multer({
  storage,
  limits: { fileSize: 5 * 1024 * 1024 }, // 5MB
  fileFilter: (req, file, cb) => {
    const allowedTypes = /jpeg|jpg|png|webp/;
    const extname = allowedTypes.test(path.extname(file.originalname).toLowerCase());
    const mimetype = allowedTypes.test(file.mimetype);

    if (extname && mimetype) {
      return cb(null, true);
    }
    cb(new Error('Only images (JPEG, PNG, WEBP) allowed'));
  }
});

module.exports = upload;
```

---

9. Testing Strategy

9.1 Testing Pyramid

```
       ┌─────────────┐
       │  E2E Tests  │  (5% - Critical flows)
       │  (Cypress)  │
       └─────────────┘
      ┌───────────────────┐
      │ Integration Tests │ (25% - API endpoints)
      │  (Jest + Supertest)│
      └───────────────────┘
```

```
┌─────────────────────────────┐
│    Unit Tests        │ (70% - Business logic)
│      (Jest)        │
└─────────────────────────────┘
```

9.2 Backend Testing

Unit Tests (Jest):

javascript

```javascript
// tests/services/menu.service.test.js
const menuService = require('../../src/services/menu.service');
const db = require('../../src/config/database');

jest.mock('../../src/config/database');

describe('MenuService', () => {
  describe('getItems', () => {
    it('should return items for a restaurant', async () => {
      const mockItems = [
        { id: '1', name: 'Pizza', base_price: 299 }
      ];

      db.query.mockResolvedValue({ rows: mockItems });

      const result = await menuService.getItems('restaurant-123');

      expect(result).toEqual(mockItems);
      expect(db.query).toHaveBeenCalledWith(
        expect.stringContaining('restaurant_id = $1'),
        ['restaurant-123']
      );
    });
  });
});
```

API Tests (Supertest):

javascript

```
// tests/routes/menu.routes.test.js
const request = require('supertest');
const app = require('../../src/app');

describe('Menu API', () => {
  let authToken;

  beforeAll(async () => {
    // Get auth token
    const res = await request(app)
      .post('/api/auth/login')
      .send({ email: 'test@example.com', password: 'password' });
    authToken = res.body.data.token;
  });

  describe('GET /api/items', () => {
    it('should return menu items', async () => {
      const res = await request(app)
        .get('/api/items')
        .set('Authorization', `Bearer ${authToken}`)
        .expect(200);

      expect(res.body.success).toBe(true);
      expect(Array.isArray(res.body.data)).toBe(true);
    });

    it('should require authentication', async () => {
      await request(app)
        .get('/api/items')
        .expect(401);
    });
  });
});
```

9.3 Frontend Testing

Component Tests (React Testing Library):

```
javascript
```

```jsx
// tests/components/ItemCard.test.jsx
import { render, screen, fireEvent } from '@testing-library/react';
import ItemCard from '../src/components/menu/ItemCard';

describe('ItemCard', () => {
  const mockItem = {
    id: '1',
    name: 'Margherita Pizza',
    base_price: 299,
    item_type: 'veg',
    image_url: '/images/pizza.jpg'
  };

  const mockOnAdd = jest.fn();

  it('renders item details correctly', () => {
    render(<ItemCard item={mockItem} onAdd={mockOnAdd} />);

    expect(screen.getByText('Margherita Pizza')).toBeInTheDocument();
    expect(screen.getByText('₹299')).toBeInTheDocument();
  });

  it('calls onAdd when Add button clicked', () => {
    render(<ItemCard item={mockItem} onAdd={mockOnAdd} />);

    const addButton = screen.getByRole('button', { name: /add/i });
    fireEvent.click(addButton);

    expect(mockOnAdd).toHaveBeenCalledWith(mockItem);
  });
});
```

9.4 Manual Testing Checklist (Phase 1)

Admin Dashboard:

- ☐ User can register and login
- ☐ User can add menu categories
- ☐ User can add menu items with photos
- ☐ User can add variants and addons
- ☐ User can create tables
- ☐ User can generate QR codes
- ☐ User can view live orders
- ☐ User can update order status
- ☐ Real-time updates work without refresh

- [ ] Dashboard works on tablet

Customer App:

- [ ] QR code scan opens correct restaurant + table
- [ ] Menu loads within 2 seconds
- [ ] Can browse all categories
- [ ] Can view item details
- [ ] Can select variants
- [ ] Can select addons
- [ ] Can add to cart
- [ ] Cart persists during session
- [ ] Can place order
- [ ] Order confirmation shows
- [ ] Can view order status
- [ ] Works on old Android phones (Android 8+)

Marketing Site:

- [ ] Landing page loads quickly
- [ ] All CTAs work
- [ ] Forms submit correctly
- [ ] Mobile responsive
- [ ] SEO meta tags present

---

10. Performance Optimization

10.1 Database Optimization

Indexing Strategy:

```sql
```

```
-- Already included in schema, but important to highlight:

-- Multi-tenant queries
CREATE INDEX idx_items_restaurant ON menu_items(restaurant_id);
CREATE INDEX idx_orders_restaurant ON orders(restaurant_id);

-- Frequent lookups
CREATE INDEX idx_tables_token ON restaurant_tables(qr_unique_token);
CREATE INDEX idx_orders_status ON orders(restaurant_id, status);

-- Time-based queries
CREATE INDEX idx_orders_placed_at ON orders(placed_at);

-- Search queries
CREATE INDEX idx_items_name ON menu_items(name text_pattern_ops);
```

Query Optimization:

- Use EXPLAIN ANALYZE for slow queries
- Limit result sets with pagination
- Use JOINs instead of multiple queries
- Cache frequently accessed data

Connection Pooling:

```javascript
// config/database.js
const { Pool } = require('pg');

const pool = new Pool({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  max: 20, // max connections
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,
});

module.exports = {
  query: (text, params) => pool.query(text, params)
};
```

10.2 API Performance

Response Caching:

```javascript
const redis = require('redis');
const client = redis.createClient();

async function cacheMiddleware(req, res, next) {
  const key = `cache:${req.originalUrl}`;

  const cached = await client.get(key);
  if (cached) {
    return res.json(JSON.parse(cached));
  }

  // Override res.json to cache response
  const originalJson = res.json.bind(res);
  res.json = (data) => {
    client.setEx(key, 300, JSON.stringify(data)); // 5 min cache
    originalJson(data);
  };

  next();
}

// Use for read-heavy endpoints
router.get('/items', cacheMiddleware, menuController.getItems);
```

Pagination:

```javascript
```

```javascript
async function getOrders(restaurantId, page = 1, limit = 50) {
  const offset = (page - 1) * limit;

  const query = `
    SELECT * FROM orders
    WHERE restaurant_id = $1
    ORDER BY placed_at DESC
    LIMIT $2 OFFSET $3
  `;

  const result = await db.query(query, [restaurantId, limit, offset]);

  return {
    data: result.rows,
    page,
    limit,
    total: result.rowCount
  };
}
```

10.3 Frontend Performance

Code Splitting:

javascript

```javascript
// Lazy load routes
const Dashboard = lazy(() => import('./pages/dashboard/Dashboard'));
const MenuList = lazy(() => import('./pages/menu/MenuList'));

function App() {
  return (
    <Suspense fallback={<Loading />}>
      <Routes>
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="/menu" element={<MenuList />} />
      </Routes>
    </Suspense>
  );
}
```

Image Optimization:

javascript

```
// Lazy load images
import { LazyLoadImage } from 'react-lazy-load-image-component';

function ItemCard({ item }) {
  return (
    <LazyLoadImage
      src={item.image_url}
      alt={item.name}
      effect="blur"
      placeholderSrc="/placeholder.jpg"
    />
  );
}
```

Bundle Optimization:

```javascript
// vite.config.js
export default {
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          'react-vendor': ['react', 'react-dom', 'react-router-dom'],
          'ui-vendor': ['@headlessui/react', 'react-icons'],
        }
      }
    }
  }
};
```

---

11. AI-Assisted Development Guidelines

11.1 Effective Prompting for Your Team

General Tips:

- Be specific about the framework/library (e.g., "React with Vite and Tailwind")
- Provide context (e.g., "This is for a multi-tenant restaurant app")
- Ask for complete, working code with all imports
- Request error handling and validation

- Ask for comments explaining complex logic

Example Good Prompts:

```
"Create a React component called MenuItemCard that displays a menu item with:
- Image (if available, otherwise placeholder)
- Name, description, price
- Veg/non-veg indicator
- Add to cart button
Use Tailwind CSS for styling. Make it mobile-responsive."
```

```
"Write a Node.js Express API endpoint to create a new menu item. Include:
- Input validation (name, price required)
- Multi-tenant check (filter by restaurant_id from JWT)
- Error handling
- Returns created item
Use PostgreSQL with pg library."
```

```
"Create a Zustand store for managing cart state with:
- addItem, removeItem, updateQuantity actions
- Calculate total
- Persist to localStorage
- TypeScript types"
```

11.2 Development Workflow

For Each Feature:

1. **Understand Requirements** (5 min)
   - Read PRD section
   - Check API design
   - Understand data flow
2. **Break into Subtasks** (5 min)
   - Database changes (if any)
   - Backend API
   - Frontend component
   - Integration
3. **Use AI for Each Subtask** (per task)
   - Prompt with specific requirements

    - Review generated code
    - Test immediately
    - Fix errors with AI help
4. **Test Integration** (10 min)
    - Manual testing
    - Check edge cases
    - Fix bugs
5. **Code Review** (10 min)
    - Check security (multi-tenant isolation)
    - Check error handling
    - Check performance
    - Refactor if needed

11.3 Common AI Tools & Usage

Cursor / Claude Code:

- Best for: Full-file generation, refactoring
- Use: Cmd+K to edit, Cmd+L to chat

GitHub Copilot:

- Best for: Auto-completion, small functions
- Use: Write comment, let it generate

ChatGPT / Claude:

- Best for: Planning, debugging, learning
- Use: Explain concepts, review architecture

Recommended AI Workflow:

1. Plan feature with ChatGPT/Claude
2. Generate code with Cursor/Copilot
3. Debug with ChatGPT if errors
4. Refactor with Cursor if needed

11.4 Code Quality Checklist

Before committing code, verify:

Security:

- [ ] All queries include `restaurant_id` filter
- [ ] User input is validated
- [ ] No SQL injection risks
- [ ] Authentication is checked

Functionality:

- [ ] Feature works as expected
- [ ] Edge cases handled
- [ ] Error messages are clear
- [ ] Loading states shown

Code Quality:

- [ ] No hardcoded values
- [ ] Functions are small and focused
- [ ] Comments explain "why", not "what"
- [ ] Consistent naming conventions

Performance:

- [ ] No unnecessary re-renders (React)
- [ ] Queries are optimized
- [ ] Images are compressed
- [ ] API responses are paginated

---

12. Development Environment Setup

12.1 Prerequisites

Install on all developer machines:

```bash
```

```
# Node.js 20 LTS
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt-get install -y nodejs

# PostgreSQL 15
sudo apt install postgresql postgresql-contrib

# Git
sudo apt install git

# VS Code (recommended)
# Download from https://code.visualstudio.com/
```

VS Code Extensions:

- ESLint
- Prettier
- Tailwind CSS IntelliSense
- PostgreSQL (for database management)
- Thunder Client (API testing)
- GitLens

12.2 Project Setup

Backend:

```
bash
```

```bash
cd backend
npm install

# Create .env file
cp .env.example .env
# Edit .env with your values

# Setup database
npm run db:setup

# Run migrations
npm run db:migrate

# Seed sample data (optional)
npm run db:seed

# Start development server
npm run dev
```

Frontend (Admin):

```bash
cd frontend/admin
npm install
npm run dev
# Opens on http://localhost:5173
```

Frontend (Customer):

```bash
cd frontend/customer
npm install
npm run dev
# Opens on http://localhost:5174
```

12.3 Environment Variables

Backend (.env):

```env
```

```
# Server
NODE_ENV=development
PORT=5000

# Database
DB_HOST=localhost
DB_PORT=5432
DB_NAME=restaurant_os_dev
DB_USER=postgres
DB_PASSWORD=your_password

# JWT
JWT_SECRET=your-super-secret-jwt-key-change-in-production
JWT_EXPIRY=24h

# File Upload
UPLOAD_DIR=./uploads
MAX_FILE_SIZE=5242880

# Frontend URLs (for CORS)
ADMIN_URL=http://localhost:5173
CUSTOMER_URL=http://localhost:5174

# Email (Phase 2)
# SMTP_HOST=
# SMTP_PORT=
# SMTP_USER=
# SMTP_PASS=
```

Frontend (.env):

```env
VITE_API_URL=http://localhost:5000/api
VITE_SOCKET_URL=http://localhost:5000
```

---

13. Monitoring & Analytics (Phase 2)

Tools to Add Later:

- **Sentry:** Error tracking
- **Mixpanel/Amplitude:** User analytics

- **LogRocket:** Session replay
- **Grafana:** Server metrics

---

14. Appendix

14.1 Technology Choices Rationale

Technology	Why Chosen	Alternatives Considered
React	Best AI tooling support, huge ecosystem	Vue (smaller), Angular (complex)
Node.js	JavaScript everywhere, fast iteration	Python Django (slower dev), Go (harder)
PostgreSQL	Reliable, free, great multi-tenancy	MySQL (weaker features), MongoDB (no relational)
Socket.io	Battle-tested, auto-fallbacks	Native WebSockets (harder), Pusher (paid)
Tailwind CSS	Fast styling, AI-friendly	Bootstrap (less flexible), Emotion (more setup)
Hostinger VPS	Affordable, full control	AWS (expensive), Heroku (limited)

14.2 Third-Party Services Budget

Service	Purpose	Cost (Phase 1)
Hostinger VPS	Server hosting	₹700/month
Domain	yourapp.com	₹500/year
SSL Certificate	Free (Let's Encrypt)	₹0
Email Service	(Phase 2)	₹0 initially
Total		**₹750/month**

14.3 Useful Resources

Documentation:

- React: https://react.dev
- Node.js: https://nodejs.org/docs
- PostgreSQL: https://www.postgresql.org/docs/
- Socket.io: https://socket.io/docs/
- Tailwind: https://tailwindcss.com/docs

Learning:

- Fireship (YouTube): Quick tech overviews
- Web Dev Simplified: React/Node tutorials
- Traversy Media: Full-stack projects

AI Coding:

- Cursor Docs: https://cursor.sh/docs
- GitHub Copilot: https://github.com/features/copilot

---

15. Next Steps

1. **Week 1:** Setup development environment for all team members
2. **Week 1-2:** Build authentication + basic restaurant setup
3. **Week 2-3:** Build menu management (categories, items)
4. **Week 3-4:** Build table management + QR generation
5. **Week 4-5:** Build customer ordering app
6. **Week 5-6:** Build order dashboard + real-time updates
7. **Week 6:** Testing, bug fixes, deployment prep
8. **Week 7:** Deploy to production, alpha testing

END OF TECHNICAL DESIGN DOCUMENT