Code Logic & Implementation Approach

GitHub Repository : https://github.com/hussain-2004/Restaurant-Management-System.git

Project Architecture Overview

Our Restaurant Management System follows a layered architecture pattern using Java and PostgreSQL, designed to handle real-time restaurant operations with multiple concurrent users. The system is built around role-based access control and status-driven workflows to ensure seamless coordination between different stakeholders.

Core Logic Implementation

1. DATABASE - First Design Approach

We started with a comprehensive database schema that captures all restaurant entities and their relationships:

1. **Tables Management :** Tables table tracks availability status (OCCUPIED/AVAILABLE)
2. **Order Lifecycle :** Orders and order_items tables manage the complete order flow
3. **User Roles:** Separate authentication and role management for Customers, Waiters, Chefs, Managers, and Admins
4. **Financial Tracking:** Bills table handles payment processing and revenue tracking

2. State-Based Workflow Logic

The entire system operates on status transitions that drive the workflow:

Table Booking: AVAILABLE → OCCUPIED → AVAILABLE
Order Process: PENDING → READY → SERVED
Payment Flow: UNPAID → PAID

This state-based approach ensures data consistency and enables real-time updates across all user interfaces.

3. Concurrent User Management

Our logic handles multiple users working simultaneously through:

- **Queue-Based Table Assignment :** First-come-first-served logic for table bookings when no tables are available
- **Role Isolation:** Each user type sees only relevant data and functions based on their role

4. Real-Time Synchronization Logic

The system implements instant updates through:

- **Event-Driven Updates :** When waiters place orders, kitchen staff immediately see them
- **Status Broadcasting :** Order status changes are reflected across waiter and chef interfaces
- **Automatic Calculations :** Bills are generated dynamically from order data with real-time pricing

Implementation Strategy

Phase 1: Core Infrastructure

- Database schema design and PostgreSQL setup
- Basic CRUD operations for all entities
- User authentication and role management system

Phase 2: Workflow Implementation

- Table booking logic with queue management
- Order placement and kitchen notification system
- Status update mechanisms between different roles

Phase 3: Business Logic Integration

- Automated billing calculations
- Payment processing and table release
- Admin dashboard for menu and sales management

Phase 4: Optimization & Concurrency

- Real-time data synchronization
- Error handling and system reliability

Key Problem-Solving Logic

Queue Management Algorithm

When all tables are occupied, customers are added to a waiting queue. Our logic processes this queue in FIFO order, automatically assigning tables as they become available, eliminating manual coordination and ensuring fairness.

Order Synchronization Logic

The moment a waiter submits an order, it's immediately visible to kitchen staff. As chefs mark individual items as completed, the system automatically updates order status to READY when all items are done, triggering notifications to waiters.

Dynamic Billing Engine

Our billing logic fetches all order items, applies current menu prices, calculates taxes/discounts, and generates accurate bills automatically. This eliminates manual calculation errors and speeds up the checkout process.

Role-Based Access Control

Each user type has specific permissions and views:

- **Customers :** Can only book tables, make orders and view their orders
- **Waiters :** Access assigned tables and order management
- **Chefs :** View pending orders and update cooking status
- **Managers :** Handle billing and payment processing
- **Admins :** Complete system control and reporting access

System Flow Logic

The entire system follows a circular workflow:

1. Customer books table → System assigns or queues
2. Waiter takes order → Kitchen receives instantly
3. Chef prepares food → Status updates to waiter
4. Manager processes bill → Table becomes available
5. System assigns table to next queued customer

This logic ensures continuous operation without manual intervention, reducing errors and improving efficiency.

Restaurant Management System - Setup Instructions

Prerequisites

- Java 21+
- PostgreSQL 14+
- Maven 3.8+
- IDE (IntelliJ IDEA recommended)

Setup Steps

1. Clone Repository

git clone https://github.com/hussain-2004/Restaurant-Management-System.git
cd Restaurant-Management-System

2. Database Setup

Step 1: Create a PostgreSQL database named **restaurant_management_db**
Step 2: Navigate to **src/main/resources/**
Step 3: Execute **db_script.sql** in your PostgreSQL terminal or pgAdmin

3. Configure Database

Edit **src/main/resources/application.properties**:

database_url= your_database_url
username= your_username
password= your_password

4. Run Application

1. Open project in IntelliJ IDEA
2. Navigate to **src/main/java/tech/zeta/commandInterface/Main.java**
3. Click the Run button
4. The command-line interface will start