Restaurant Management System - Design Document

IMT2022003 Ritish S, IMT2022076 Mohit N,
IMT2022086 Ananthakrishna K, IMT2022103 Anurag R

November 15, 2024

Contents

1 Introduction

1.1 Overview

This document provides the design details for a Restaurant Management System, including high-level and low-level design, use cases, class diagrams, and sequence diagrams.

1.2 Objectives

The main objective of this system is to manage restaurant operations efficiently, including table reservations, order management, and billing.

2 High-Level Design

The Restaurant Management System (RMS) follows a 3-tier architecture:

2.1 Presentation Layer (Frontend)

The Command-Line Interface (CLI) serves as the user interface, visible to end users.

2.2 Business Logic Layer (Backend)

This layer handles:

- User Management
- Billing System
- Menu Management
- Reporting System
- Employee Management
- Authentication and Authorization

2.3 Data Access Layer (Database)

This layer abstracts database operations. The key components include:

- User Management System for accounts, permissions, and logging
- Billing System for creating, viewing, and archiving transactions
- Menu Management for updating menu items and tracking popularity
- Reporting System for generating revenue and sales reports
- Employee Management for tracking performance and schedules
- Authentication and Authorization for secure access and data encryption

This design aims to create a modular, maintainable, and secure restaurant management system that streamlines administrative tasks and provides data-driven insights.

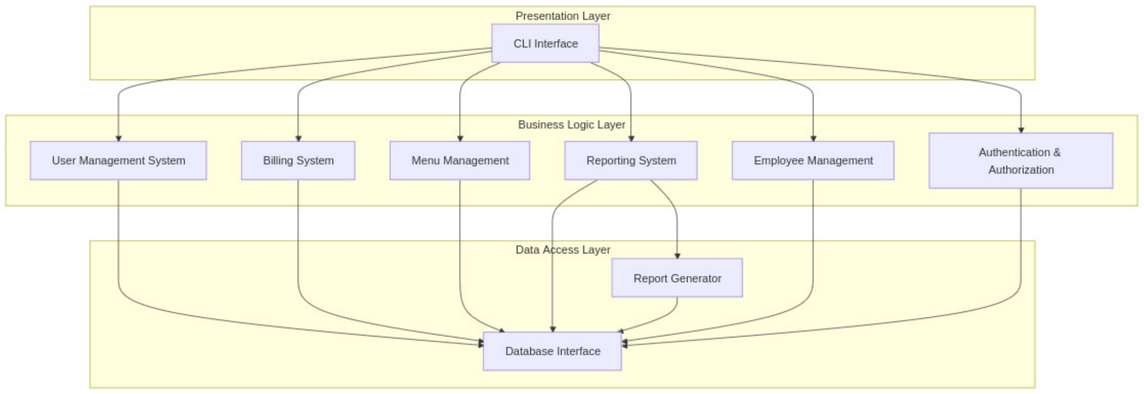

Figure 1: Architecture Diagram

2.4 Module Breakdown

The system can be broken down into the following modules -

- **User Management**: Handles user registration and authentication.
- **Menu Management**: Manages the restaurant's menu and item details.
- **Order Management**: Processes and tracks customer orders.
- **Payment and Billing**: Handles billing and payment processing.

3 Low-Level Design (LLD)

3.1 Class Diagram

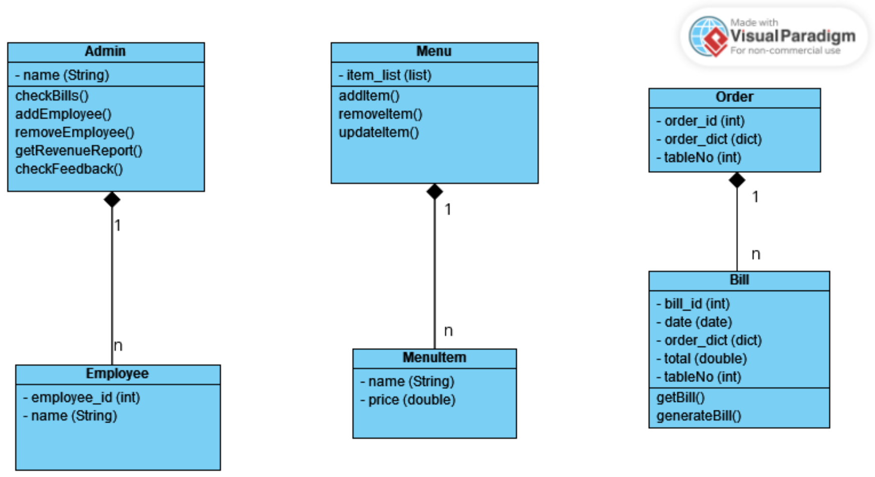

Figure 2: Class Diagram for Restaurant Management System

3.2 Class Descriptions

- **Admin**:
  - Attributes: name (String)
  - Methods: checkBills(), addEmployee(), removeEmployee(), getRevenueReport(), check-Feedback()
- **Employee**:
  - Attributes: employee_id (Integer), name (String)
- **Menu**:
  - Attributes: item_list (List¡MenuItem¿)
  - Methods: addItem(), removeItem(), updateItem()
- **MenuItem**:
  - Attributes: name (String), price (Double)
- **Order**:
  - Attributes: order_id (Integer), order_dict (Dictionary<MenuItem, Integer>), table_no (Integer)

- **Bill**:
  - Attributes: bill_id (Integer), date (Date), order_dict (Dictionary), total (Double), table_no (Integer)
  - Methods: getBill(), generateBill()

3.3 Database Schema

This section describes the conceptual database schema for the Restaurant Management System (RMS), covering key entities, their attributes, and relationships.

Entities and Attributes

1. Users :Contains information about all system users, such as admins and employees.

Attributes:

- user_id: Unique identifier for each user.
- username: Unique name for login purposes.
- password_hash: Encrypted password for authentication.
- role: Specifies user role, e.g., admin or employee.
- name: Full name of the user.
- created_at: Date and time the user was added.

Relationships: Users can create orders; employees may have assigned schedules.

2. MenuItems: Stores details of items available on the restaurant menu.

Attributes:

- item_id: Unique identifier for each menu item.
- name: Name of the menu item.
- description: Brief description of the item.
- price: Cost of the item.
- is_available: Indicates if the item is currently available.
- created_at, updated_at: Timestamps for when the item was added or last updated.

Relationships: Menu items are linked to orders through OrderItems.

3. Orders: Represents individual orders placed in the restaurant.

Attributes:

- order_id: Unique identifier for each order.
- user_id: ID of the user (employee) who created the order.
- table_no: Table number associated with the order.
- order_date: Date and time the order was placed.

Relationships: Orders contain one or more OrderItems and are associated with a Bill.

4. OrderItems: Contains specific items and quantities for each order.

Attributes:

- order_item_id: Unique identifier for each item within an order.
- order_id: ID linking this item to an order.
- item_id: ID of the menu item ordered.
- quantity: Number of units ordered for the item.

Relationships: Order items are linked to orders and menu items.

5. Bills: Contains billing information for each completed order.

Attributes:

- bill_id: Unique identifier for each bill.
- order_id: ID linking the bill to an order.
- total_amount: Total payable amount for the order.
- billing_date: Date and time the bill was generated.
- payment_status: Indicates if the bill is paid or unpaid.

Relationships: Each bill is associated with a single order.

6. Reports: Stores data for various generated reports, such as revenue or sales reports.

Attributes:

- report_id: Unique identifier for each report.
- report_type: Type of report (daily, weekly, monthly).
- start_date, end_date: Date range the report covers.
- generated_at: Date and time the report was created.
- report_data: Data for the report.

Relationships: Reports can reference data from orders, bills, and menu items for analytics.

Entity Relationships Summary

- Users to Orders: A user (employee) may create multiple orders.
- Orders to OrderItems: An order can have multiple items, linking each ordered item and its quantity.
- Orders to Bills: Each order has a corresponding bill, but the order may exist before the bill is generated.
- MenuItems to OrderItems: Each menu item can appear in multiple orders, and an order can contain multiple menu items.
- Users to EmployeeSchedules: Each employee may have multiple schedule entries.

4 Key Design Patterns Used

4.1 Singleton Pattern

The Singleton pattern is used in the RMS for:

- Database connection management - Ensuring a single, shared database connection is used across the system.
- User session management - Maintaining a single user session object for tracking logged-in users and their permissions.

4.2 Factory Pattern

The Factory pattern is used in the RMS for:

- Creating different types of reports - Allowing the system to generate various report formats (daily, weekly, monthly) without exposing the creation logic.
- Creating different types of users - Simplifying the process of instantiating users with different roles (admin, employee).

4.3 Observer Pattern

The Observer pattern is used in the RMS for:

- Updating menu items - Allowing the system to notify relevant components (e.g., reporting, billing) when menu items are added, removed, or modified.
- Bill notifications - Enabling the system to alert users (admins, employees) when new bills are created or updated.

4.4 Command Pattern

The Command pattern is used in the RMS for implementing the CLI commands. This pattern encapsulates each CLI command as a separate object, allowing for:

- Easy addition or modification of commands without impacting the overall CLI structure.
- Improved testability and maintainability of the command-handling logic.
- Support for features like undo/redo for CLI operations.

The use of these design patterns helps the RMS achieve a modular, extensible, and maintainable architecture, making it easier to implement new features and evolve the system over time.

5 Use Case Diagrams

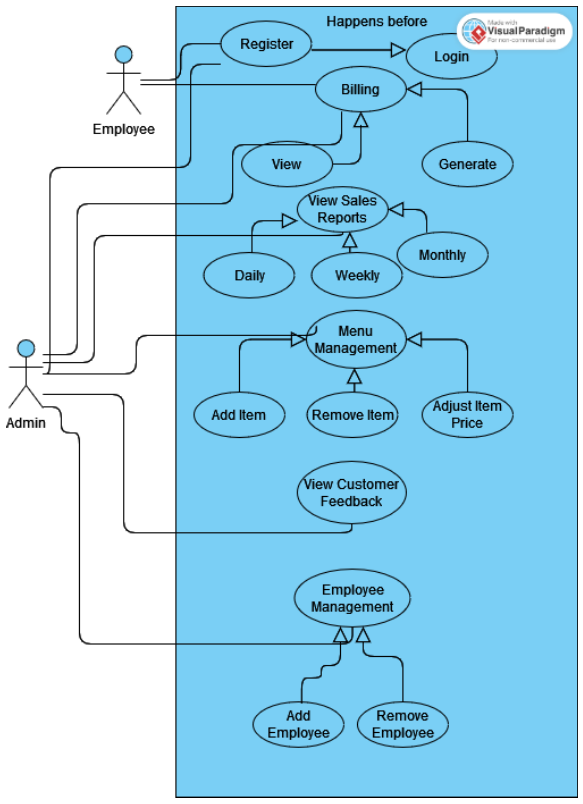

Figure 3: Use Case Diagram

5.1 Use Case Descriptions

- **Place Order**: The customer selects menu items and places an order.

- **Make Payment**: The customer completes payment for the order.
- **Manage Menu**: The restaurant staff updates the menu items.

6 Sequence Diagrams

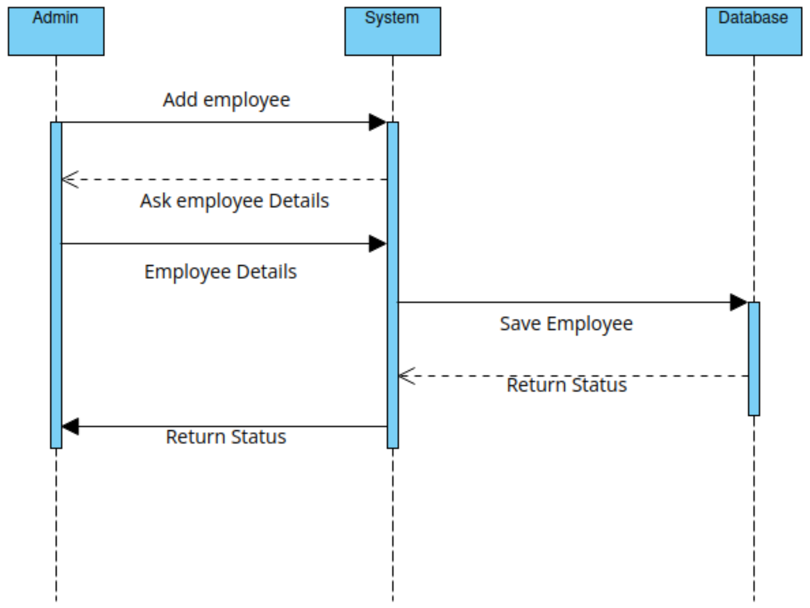

Figure 4: Sequence Diagram for adding an employee

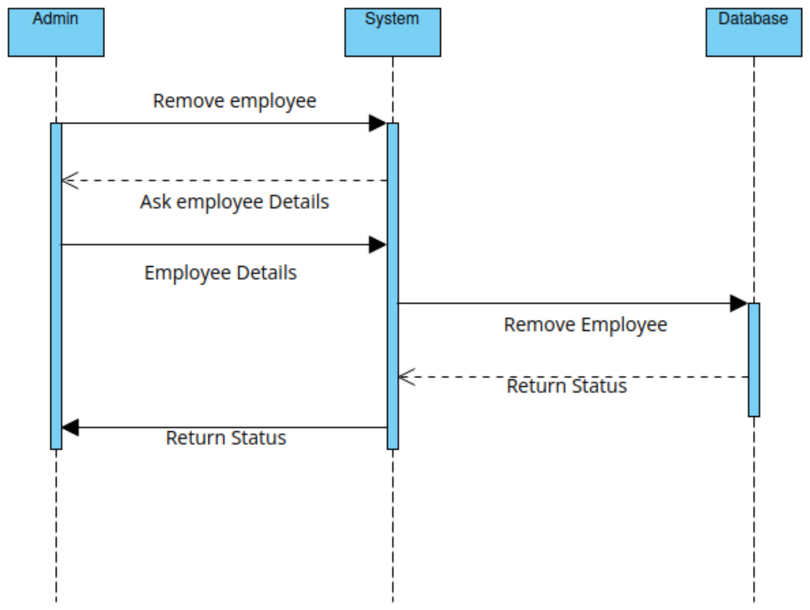

Figure 5: Sequence Diagram for removing an employee

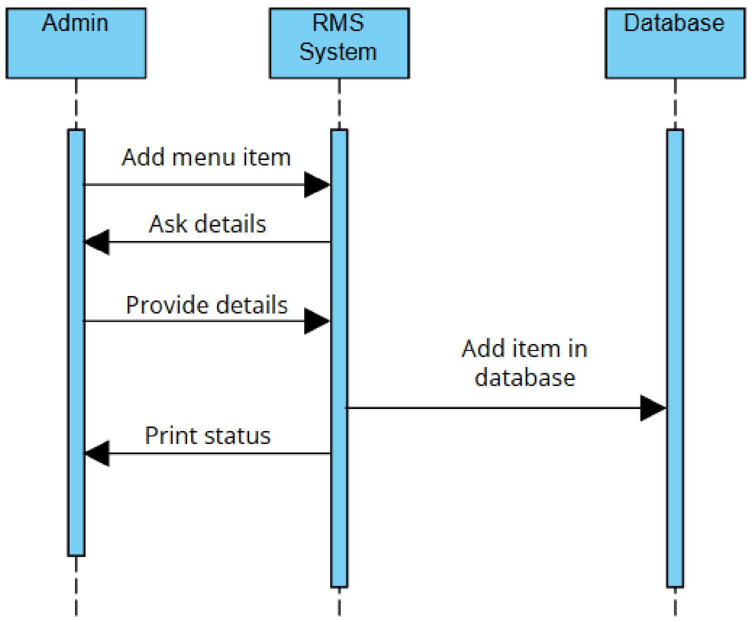

Figure 6: Add menu item

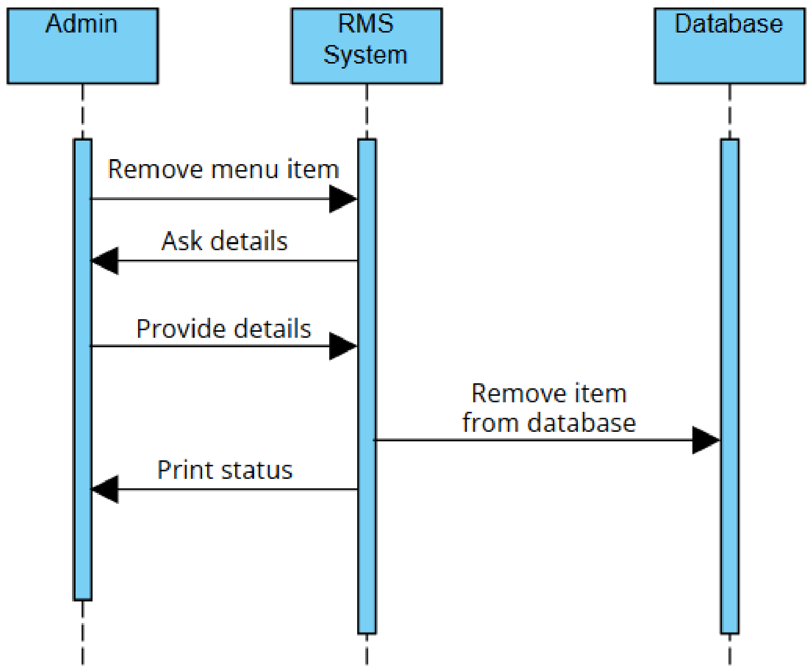

Figure 7: Remove menu item

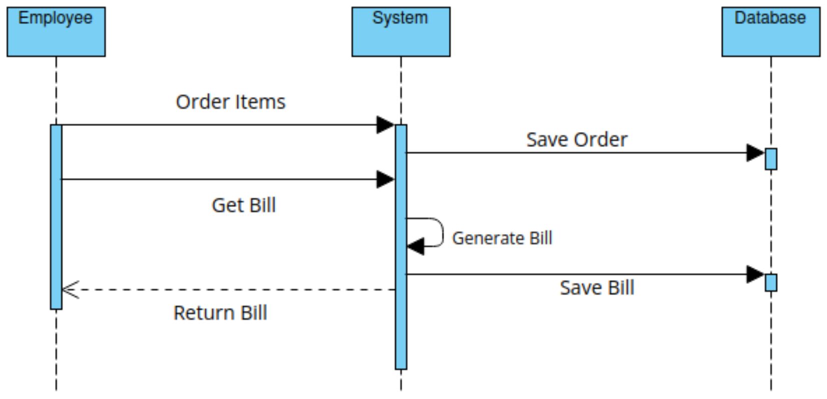

Figure 8: Generate bill

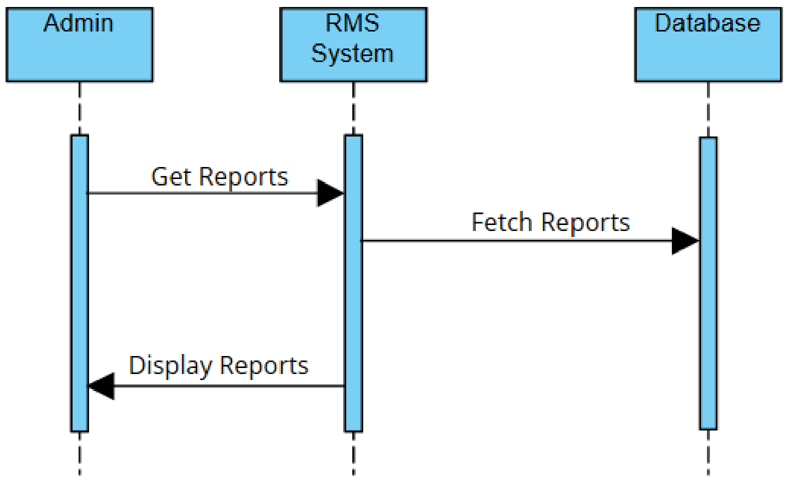

Figure 9: Fetch Reports