# Restaurant Management System

**shahd kamal Mohamed 2101601**

**shahd ali Mohamed 2101616**

**shahd khaled Mohamed 2101620**

**shahd zakaria Soliman 2101605**

**rahma yahia abd elrahem 2101456**

## 1. Introduction

This document outlines the **Restaurant Management System (RMS)**, which aims to automate and streamline various restaurant operations. It covers order management, menu management, table reservations, customer management, payment processing, and integrates several design patterns for maintainability and scalability.

## 2. System Overview

### 2.1 Purpose

The **Restaurant Management System** automates essential tasks within a restaurant, including:

- **Order Management**
- **Menu Management** (e.g., Burgers, Drinks, Desserts, Fries)
- **Table Reservations**
- **Customer Management**
- **Payment Processing**

- **Inventory Tracking (Optional)**

### 2.2 Features

- **Order Management**: Create and manage orders, add/remove items, calculate totals, and generate bills.
- **Menu Management**: Manage menu items such as burgers, drinks, desserts, fries, and sauces.
- **Table Reservations**: Manage reservations, assign tables, and track availability.
- **Customer Management**: Store customer details and track order history.
- **Payment Processing**: Process payments and generate receipts.
- **Inventory Tracking** (Optional): Track stock levels of ingredients and supplies.

## 3. Design Considerations

### 3.1 Design Patterns

The following design patterns are used in the system to improve maintainability and flexibility:

- **Factory Pattern**: Used to create objects of different types without specifying the exact class of object that will be created.
    - **MenuItemFactory**: Factory for creating menu items such as **Burgers**, **Drinks**, **Desserts**, **Fries**, and **Sauces**.
    - **TableFactory**: Factory for creating different types of tables (e.g., Regular, VIP, Outdoor).

- **Singleton Pattern**: Ensures that a class has only one instance and provides a global point of access.
    - **OrderManager**: Manages all order operations and ensures a single instance for handling orders.
    - **PaymentSystem**: Manages payment processing with a single instance.

- **Prototype Pattern**: Allows cloning of objects to create copies with the same properties.
    - **MenuItem**: Prototype for cloning menu items (e.g., Burger, Drink, Fries).

- **Proxy Pattern**: Provides a surrogate or placeholder for another object to control access.
    - **DatabaseConnectionProxy**: Controls access to the database, ensuring efficient resource management.

- **Facade Pattern**: Provides a simplified interface to a complex subsystem.
    - **RestaurantFacade**: Simplifies access to core restaurant operations like placing orders, processing payments, and managing reservations.

## 3.2 Class Diagram

[Include a UML class diagram illustrating the relationships between the classes and design patterns.]

**4. Class Descriptions**

- **Table**: Represents a table in the restaurant.
    - **Attributes**: Table ID, type (Regular, VIP, Outdoor), number of chairs.
    - **Methods**: getType(), setType(), getNumOfChairs(), setNumOfChairs(), clone().
- **Burger**: Represents a burger in the menu.
    - **Attributes**: Name, size, price, ingredients.
    - **Methods**: getName(), setName(), getSize(), setSize(), getPrice(), setPrice(), clone().
- **Drink**: Represents a drink in the menu (e.g., soda, juice).
    - **Attributes**: Name, size, price.
    - **Methods**: getName(), setName(), getSize(), setSize(), getPrice(), setPrice(), clone().
- **Dessert**: Represents a dessert in the menu.
    - **Attributes**: Name, price, flavor.
    - **Methods**: getName(), setName(), getPrice(), setPrice(), getFlavor(), setFlavor(), clone().
- **Fries**: Represents fries in the menu.
    - **Attributes**: Name, size, price.
    - **Methods**: getName(), setName(), getSize(), setSize(), getPrice(), setPrice(), clone().
- **Sauce**: Represents a sauce in the menu.
    - **Attributes**: Name, type (e.g., Ketchup, Mayonnaise), price.
    - **Methods**: getName(), setName(), getPrice(), setPrice(), clone().
- **OrderItem**: Represents an individual item in an order.
    - **Attributes**: Menu item (e.g., Burger, Drink, Fries), quantity.

    - **Methods**: getMenuItem(), setMenuItem(), getQuantity(), setQuantity().
- **Orders**: Represents a customer's order, which contains multiple **OrderItems**.
    - **Attributes**: Order ID, list of **OrderItems**, total cost.
    - **Methods**: addOrderItem(), removeOrderItem(), calculateTotal(), generateBill().
- **Customer**: Represents a customer of the restaurant.
    - **Attributes**: Customer ID, name, contact details, order history.
    - **Methods**: getCustomerID(), setCustomerID(), getName(), setName(), getContactDetails(), setContactDetails(), addOrderToHistory().
- **Reservation**: Represents a table reservation.
    - **Attributes**: Reservation ID, **Customer**, **Table**, reservation time.
    - **Methods**: getReservationID(), setReservationID(), getCustomer(), setCustomer(), getTable(), setTable(), getReservationTime(), setReservationTime().
- **DatabaseConnectionProxy**: Proxy class that controls access to the **DatabaseConnection**.
    - **Methods**: getConnection() (controls access to the database).
- **OrderManager**: Singleton class for managing orders.
    - **Methods**: createOrder(), addOrderItem(), removeOrderItem(), calculateTotal(), generateBill(), etc.
- **PaymentSystem**: Singleton class for managing payment processing.
    - **Methods**: processPayment(), generateReceipt().
- **RestaurantFacade**: Facade class that simplifies the interface to core operations like placing orders, processing payments, and making reservations.
    - **Methods**: placeOrder(), processPayment(), makeReservation().

## 5. Implementation Details

### 5.1 Database

MySQL (or another suitable relational database)

### 5.2 Database Connection with MySQL

The system connects to a **MySQL database** to store essential data such as orders, menu items, and customer information. A **DatabaseConnectionProxy** controls access to the database, ensuring efficient resource management.

**5.53GUI Design**

The **Graphical User Interface (GUI)** is designed to provide an easy-to-use interface for managing restaurant operations. The GUI includes:

- **Login Screen**: For authentication.
- **Main Menu**: Navigation to different modules (Order Management, Menu Management, Reservations, etc.).
- **Order Management**: For adding and managing orders.
- **Menu Management**: For adding, editing, and deleting menu items.
- **Reservation Management**: For managing table reservations.
- **Payment Screen**: For processing payments.

**Example GUI Screens:**

- **Login Screen**: User enters credentials to log into the system.
- **Main Menu Screen**: Buttons for "Orders", "Menu", "Reservations", "Payments".
- **Order Management Screen**: List of current orders, buttons to create new orders, and itemized details of orders.
- **Menu Management Screen**: Interface for managing menu items (add, edit, delete).
- **Payment Screen**: Interface for processing payments and generating receipts.

## 6. Future Enhancements

- **Inventory Tracking**: Implement inventory management to track ingredient and supply stock levels.
- **POS Integration**: Integrate with a Point-of-Sale system for seamless transaction processing.
- **Online Ordering**: Expand the system to support online orders through a website or mobile app.
- **User Interface Improvements**: Enhance the overall user experience and appearance of the application.

## 7. Conclusion

This document provides an overview of the **Restaurant Management System**, including its purpose, features, and design patterns. The system employs key design patterns such as **Factory**, **Singleton**, **Prototype**, **Proxy**, and **Facade** to ensure scalability, maintainability, and flexibility. The integration of a **GUI** and **MySQL** database makes it user-friendly and efficient for restaurant staff to manage daily operations.