INSTITUTE OF SPACE TECHNOLOGY
KICSIT, Kahuta Campus

# Project Report

## Restaurant Management System

**Group Members:**

- Safa Waseem — 242201054
- Sania Shabbir — 242201055
- Misbah Saeed — 242201059

**Submitted to: Sir Uzair**

Department of Computer Science
KICSIT

June 15, 2025

## Contents

## 1 Introduction

The **Restaurant Management System** is a console-based application developed in C++ to manage various operations within a restaurant. This includes order placement, billing, employee tracking, table management, and daily sales reporting. The system implements core **Object-Oriented Programming (OOP)** principles such as abstraction, encapsulation, inheritance, and polymorphism.

## 2 Objectives

- Enable customers to place orders from a menu.
- Manage restaurant tables effectively.
- Maintain records of employees.
- Generate bills and daily sales reports.
- Apply OOP principles to ensure clean and maintainable code.

## 3 Features

- Order placement by customers.
- Dynamic assignment and release of tables.
- Display of employee roles (Waiter, Chef).
- Billing system with price calculation.
- Daily sales tracking and reporting.
- File saving for orders and sales.

## 4 Technologies Used

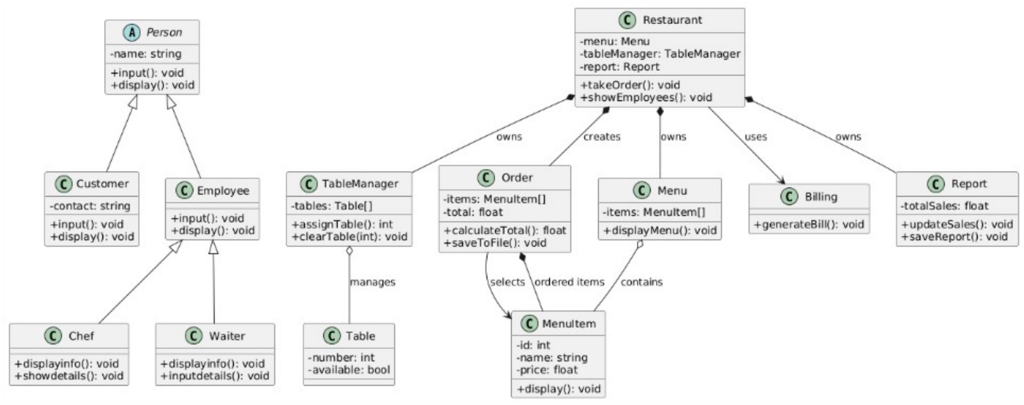
- **Programming Language:** C++
- **OOP Concepts:**
  - **Abstraction:** Abstract class `Person`
  - **Encapsulation:** Use of private class members
  - **Inheritance:** `Customer`, `Employee`, `Chef`, `Waiter`
  - **Polymorphism:** Virtual functions in `Person`
- **File Handling:** For order and sales persistence

## 5 Class Overview

| Class | Purpose |
|---|---|
| `Person` | Abstract base class for common attributes like `name`. |
| `Customer` | Inherits from `Person`; captures customer name and contact. |
| `Employee` | Inherits from `Person`; base class for restaurant staff. |
| `Waiter`, `Chef` | Derived from `Employee`; represents roles in the restaurant. |
| `MenuItem` | Represents food items with ID, name, and price. |
| `Menu` | Holds and displays all `MenuItems`. |
| `Table` | Represents a single table's number and availability. |
| `TableManager` | Manages all tables and their status. |
| `Order` | Stores a customer's order and calculates the total amount. |
| `Billing` | Handles bill generation and display. |
| `Report` | Tracks and reports total daily sales. |
| `Restaurant` | Central class that coordinates all operations. |

## 6 UML Diagram

The UML (Unified Modeling Language) class diagram below illustrates the structure of the Restaurant Management System. It shows key classes such as `Person`, `Customer`, `Employee`, and how inheritance is applied. It also displays associations between classes like `Order`, `Menu`, `Table`, and `Restaurant`.



## 7 Program Flow

1. Start program with main menu.
2. User chooses an action (e.g., Take Order, Show Employees).
3. For orders:
    - Input customer details.
    - Assign available table.
    - Show menu and accept item IDs.
    - Generate bill and update sales.
    - Save to `orders.txt`.
4. Tables can be manually cleared.
5. Daily sales saved to `sales.txt` before exiting.

## 8 Code Snippets

Example of polymorphism using virtual functions:

```
class Person {
protected:
    string name;

public:
    Person(string n) : name(n) {}
    virtual void showRole() = 0; // Abstract method
};

class Waiter : public Person {
public:
    Waiter(string n) : Person(n) {}
    void showRole() override {
        cout << "I am a Waiter. Name: " << name << endl;
    }
};
```

## 9 Object-Oriented Design Explanation

- **Abstraction:** The abstract class `Person` provides a base for different people (e.g., customers, staff), hiding implementation details.
- **Encapsulation:** Class data members are kept private or protected to ensure data integrity and accessed via public methods.

- **Inheritance:** `Employee` and `Customer` inherit from `Person`; `Chef` and `Waiter` inherit from `Employee`.
- **Polymorphism:** Overridden virtual functions like `showRole()` allow derived classes to behave differently at runtime.

## 10 Testing and Validation

The system was tested with multiple use cases:

- **Order Entry:** Valid and invalid item IDs tested.
- **Table Management:** Full and available table conditions handled correctly.
- **Billing:** Total amount calculation confirmed.
- **File Output:** Checked for accurate writing to `orders.txt` and `sales.txt`.

## 11 User Manual

### Running the Program

Compile using any standard C++ compiler and run the executable to start the system.

### Main Menu Options

- **1. Take Order:** Enter customer details, assign table, and select menu items.
- **2. Show Employees:** Lists chef and waiter details.
- **3. Show Sales Report:** Displays daily total revenue.
- **4. Clear Table:** Frees up a table for new customers.
- **5. Show Table Status:** Shows current availability of tables.
- **6. Exit:** Saves sales data and exits the program.

## 12 Sample Output

```
======= Restaurant Management =======
1. Take Order
2. Show Employees
3. Show Sales Report
4. Clear Table
5. Show Table Status
6. Exit
Enter your choice: 1

Enter name: Ali
Enter contact number: 0300-1234567
Assigned Table No: 3

--- Menu ---
1. Burger - Rs. 250
2. Pizza - Rs. 500
...

Enter item ID to add to order (0 to finish): 1
Burger added to order.
Enter item ID to add to order (0 to finish): 2
Pizza added to order.
Enter item ID to add to order (0 to finish): 0

----- Order Summary -----
Burger - Rs. 250
Pizza - Rs. 500
Total Amount: Rs. 750
```

## 13 Files Created

- **orders.txt**: Stores customer name, table number, ordered items, and total.
- **sales.txt**: Logs total daily sales when exiting the program.

## 14 Strengths and Advantages

- Fully modular design using OOP.
- Easy to understand and maintain.
- Data persistence through file handling.
- Can be extended for future requirements.

## 15 Limitations

- Uses fixed-size arrays instead of dynamic containers.
- No error handling for file read/write failures.
- Console-based; lacks graphical interface.

## 16 Future Enhancements

- Use STL containers like `vector`.
- Add GUI with Qt or Web interface.

- Integrate database for robust storage.
- Implement discount, tax, and login systems.

## 17 Conclusion

This project successfully demonstrates how OOP concepts can be applied to real-world problems. It provides an effective simulation of restaurant operations and serves as a solid foundation for more advanced systems.