Contents

[Order Management System Implementation Documentation 2](#_Toc197891043)

[Overview 2](#_Toc197891044)

[Main Components 2](#_Toc197891045)

[1. Server (server.py) 2](#_Toc197891046)

[2. Client (client.py) 3](#_Toc197891047)

[Error Handling 4](#_Toc197891048)

[Design Patterns 4](#_Toc197891049)

[Summary 4](#_Toc197891050)

# Order Management System Implementation Documentation

# Overview

This document describes the implementation of the Order Management System for the restaurant. The implementation uses a **client-server architecture** and **Façade Design Pattern** in Python. The server manages all order data and business logic, while clients provide a graphical user interface (GUI) using **tkinter** for waitstaff and kitchen staff. Communication between client and server is handled via sockets and the **Facade** design pattern is used to simplify server-side order operations.

# Main Components

## 1. Server (server.py)

The server is responsible for managing all order data, handling multiple client connections, and processing requests from clients.

#### Key Classes:

1. **OrderManagement**
   * Handles the core logic for adding, retrieving, updating, and deleting orders.
   * Stores orders in a class-level dictionary (orders\_dict).
   * Methods:
     + add\_new\_order(table\_num, items, special\_requests)
     + retrieve\_current\_orders()
     + update\_order\_progress(order\_id, new\_progress)
     + delete\_order(order\_id)
2. **OrderFacade**
   * Implements the Facade design pattern.
   * Provides a simplified interface for order operations, delegating to OrderManagement.
   * Methods:
     + add\_order(table\_num, items, special\_requests)
     + get\_orders()
     + update\_order(order\_id, new\_status)
     + delete\_order(order\_id)
3. **MessageSender**
   * Handles sending and receiving messages over sockets, including checksum calculation for data integrity.
   * Methods:
   * send\_message(client\_socket, message)
   * recv\_message()
4. **Server**
   * Listens for incoming client connections and spawns a new thread for each client.
   * Manages active client sockets.
   * Methods:
     + start()
     + client\_disconnect(conn, addr)
5. **ClientHandler**
   * Handles communication with a single client.
   * Uses OrderFacade to process client requests.
   * Methods:
     + receive()
     + handle()
     + message\_options(msg)

#### Error Handling

* All socket operations are wrapped in try-except blocks.
* Order operations check for invalid input and missing orders.
* Disconnected clients are handled gracefully.

## 2. Client (client.py)

The client provides a Tkinter-based GUI for restaurant staff to interact with the order system.

#### Key Classes:

* **MessageSender**
  + Handles sending and receiving messages to/from the server, with checksum validation.
* **Client**
  + Manages the connection to the server and launches the appropriate user interface (waitstaff or kitchen staff).
* **UI (Base Class)**
  + Sets up the main window and displays current orders in a scrollable text area.
  + Periodically refreshes the order list from the server.
* **WaitstaffUI (Inherits UI)**
  + Provides forms for adding and deleting orders.
  + Validates user input and displays confirmation or error messages.
* **KitchenStaffUI (Inherits UI)**
  + Provides a form for updating the status of existing orders.
  + Validates input and displays confirmation or error messages.

#### User Interface Flow

1. **Startup:**
   * User selects their role (waitstaff or kitchen staff).
2. **Order Display:**
   * All users see a live-updating list of current orders.
3. **Waitstaff Actions:**
   * Add new orders (table number, items, special requests).
   * Delete existing orders by ID.
4. **Kitchen Staff Actions:**
   * Update the status of orders (e.g., "In Progress", "Done").
5. **Feedback:**
   * All actions provide immediate feedback via message boxes.

#### Error Handling

* All user input is validated (e.g., required fields, correct datatypes).
* Errors are displayed using tkinter.messagebox.
* Network errors and server disconnects are handled gracefully.

## Error Handling

* All network and order operations are wrapped in try-except blocks.
* User input is validated for required fields and correct datatypes.
* Errors are displayed to users via message boxes.
* Server and client handle disconnects and unexpected errors without crashing.

## Design Patterns

* **Client-Server Architecture:**  
  The system is split into a server (central order management) and multiple clients (user interfaces).
* **Facade Pattern (Server):**  
  The OrderFacade class provides a simple interface for order operations, hiding the complexity of the underlying OrderManagement logic.

## Summary

This Order Management System provides a robust, user-friendly way for restaurant staff to manage orders in real time. The use of the Facade pattern and careful error handling ensures the system is maintainable, extensible, and reliable for both waitstaff and kitchen staff.

Feel free to copy and paste this into your Word document! If you need any further adjustments, just let me know.