Taste of Tradition Management System

Technical Documentation

❖Project Overview

- Project Name: Taste of Tradition Management System
- Team Number: 05 INDIXPERT ACADEMY
- Team Members Name: Khushi, Anand
- Date: 2024-25

❖Project Summary

The **Taste of Tradition Management Restaurant System** is a comprehensive software designed to simplify and streamline restaurant operations. It integrates modules for user authentication, menu management, order processing, table reservations, billing, and inventory tracking. This modular design ensures scalability, security, and efficiency.

❖Table of Contents

1. **System Overview**
   Overview of the restaurant management system, including its structure and purpose. (E.g., Authentication, Menu, Orders, Staff, etc.)
2. **System Requirements**
   Necessary components such as Python dependencies (uuid, datetime, json) and configuration details.
3. **Role-Based Access**
   Explanation of user roles (Owner, Staff) and their respective permissions (e.g., managing menu, bookings, orders, billing).
4. **How to Start the Project**
   Entry point details (main() function in Src. Dashboard. manage_dashboard) and instructions for initializing the application.
5. **Detailed Module Descriptions**
   - **Authentication**: Managing user signup, login, and access.
   - **Menu Management**: Adding, updating, and deleting menu items.
   - **Order Management**: Handling customer orders, stock checks, and cancellations.
   - **Billing**: Generating and managing bills.

- **Table Booking**: Reservations and availability checks.
- **Staff Management**: Employee profiles, salaries, and status.
- **Reports**: Generating billing, reservation, and order reports.

❖Introduction

The system has been developed with an emphasis on modularity and role-based access, enabling tailored functionalities for Owners, Staff, and Customers. Key features include:

- **Role-Based Dashboards:** Personalized views for users based on their roles.
- **Inventory Management:** Real-time updates on stock levels.
- **Table Booking:** Seamless reservation system with availability tracking.

❖Key Features Preview

1. **Authentication Module**
    - Role-based login with secure password handling.
    - Automatically assigns "Owner" role to the first registered user.
2. **Order Management**
    - View, update, or cancel orders.
    - Detailed billing with GST calculation (18%).
3. **Table Booking Module**
    - Reserve tables with unique Booking IDs.
    - Check availability and modify reservations.

❖Benefits

- **Efficiency:** Streamlines operations, reducing manual effort.
- **Security:** Role-based access control ensures data safety.
- **Scalability:** Modular structure supports future growth and enhancements.

❖Installation Steps

Step	Description	Command/Details
1. Prerequisites	Ensure Python and pip are installed on your system.	- Check Python: python --version or python3 --version - Download Python
2. Install VS Code	Download and install Visual Studio Code on your system.	- Download VS Code: VS Code Official Site - Follow the platform-specific installation guide below.
	For Windows:	- Run the downloaded installer. - Select **"Add to PATH"** during installation. - Complete installation and launch VS Code.
	For macOS:	- Open the .dmg file and drag VS Code to the Applications folder. - Launch VS Code from the Applications folder or Spotlight.
	For Linux:	- Install using the terminal: sudo apt install code (Debian/Ubuntu) sudo dnf install code (Fedora)
3. Download Project	Clone your project repository.	- Clone the repository: git clone https://github.com/khus759/Restaurant_Management_System - Extract if downloaded as a zip file.
4. Open Project in VS Code	Open the project directory in Visual Studio Code.	- Launch VS Code. - Use File > Open Folder to select the project directory or open it from the terminal: code <project_directory>
5. Create Virtual Environment	Set up an isolated environment	- In VS Code Terminal: python -m venv venv - **Activate Virtual Environment**:

Step	Description	Command/Details
	for dependencies.	- Linux/macOS: source venv/bin/activate - Windows: venv\Scripts\activate
6. Configure Virtual Environment in VS Code	Set VS Code to use the virtual environment.	- Press Ctrl+Shift+P (or Cmd+Shift+P on macOS). - Search for Python: Select Interpreter. - Select the Python interpreter from the venv folder.
7. Install Dependencies	Install required Python libraries.	- If requirements.txt exists: pip install -r requirements.txt - For missing libraries (e.g., uuid, maskpass): pip install <library_name>
8. Set Up Database Files	Ensure required JSON files (e.g., users_file, menu_file) exist in the correct paths.	- Check file paths in the code. - If missing, create them with initial content, e.g., an empty list [].
9. Run the Project	Execute the main script to start the system.	- In VS Code terminal: python main.py (Replace main.py with the actual entry point, e.g., Src/Dashboard/manage_dashboard.py).

Detailed VS Code Installation Guide:

Windows:

1. Go to the VS Code Download page and download the Windows installer.
2. Run the installer and follow the instructions:
    - Select the **"Add to PATH"** option during installation for easy terminal access.
3. Launch VS Code after installation.

macOS:

1. Download the .dmg file from the VS Code Download page.

2. Open the downloaded file and drag **Visual Studio Code** to the Applications folder.
3. Launch VS Code from the Applications folder or Spotlight.

Linux:

1. Use the terminal to install VS Code.
    - For Debian/Ubuntu: sudo apt install code.
    - For Fedora: sudo dnf install code.
    - For other distributions, refer to Linux-specific instructions.
2. Alternatively, download the .deb or .rpm file from the VS Code Download page and install it manually.

DASHBOARD

Component	Class / Function	Description
Main Entry	main()	Entry point for the application, displays welcome message, and prompts for Signup, Login, or Exit.
User Authentication	AuthSystem	Manages user authentication (Signup, Login) and role-based access control. Provides user role to be passed to the appropriate dashboard.
Dashboard Selector	show_dashboard(user_role)	Displays either the Owner or Staff dashboard based on user_role.
Owner Dashboard	Owner Dashboard	Contains methods for handling all operations available to the Owner, such as Menu Management, Order Management, Invoice Management, Table Booking Management, Stock Management, Staff Management, and Report Management.
Staff Dashboard	Staff Dashboard	Contains methods for handling operations available to Staff, such as Order Management, Invoice Management, Table Booking

Component	Class / Function	Description
		Management, Stock Management, and accessing their Profile.
Menu Management	Menu Management	Allows adding, updating, deleting menu items and ingredients, viewing the menu, and checking stock of ingredients. Called within the Owner dashboard.
Order Management	Order Management System	Manages taking, updating, canceling orders, checking specific orders, and showing all orders. Available for both Owner and Staff dashboards.
Invoice Management	Billing System	Handles generating and viewing bills, marking as paid, and viewing all bills. Available in both dashboards.
Table Booking Management	Table Booking System	Provides table reservation, cancellation, update, and viewing availability or all reservations. Used by both Owner and Staff dashboards.
Stock Management	Report, Expiration Report, Reservation Report, OrderReport, BillReport	Provides various report generation functionalities, including soon-to-expire items, reservation summaries, order summaries, and bill reports. Available in both dashboards but with role-specific options.
Staff Management	StaffManagementSystem, SalaryManagement	Allows the Owner to add, view, update, and delete staff profiles and manage salaries. Not available in Staff dashboard; only Owner can access.
Utility Functions	ask_for_dashboard, display_* functions	Various utility functions for displaying menus and choices across modules, ensuring code modularity and readability.

AUTHENTICATION

Component	Details
Class Name	AuthSystem
Purpose	Manages user authentication (signup, login) and role-based functionalities (Owner/Staff). Handles user data storage in users.json and validation against employee.json for Staff users.
Attributes	- **users_file**: Path to users.json, storing user credentials and information. - **employee_file**: Path to employee.json, used for validating Staff user IDs. - **current_user**: Holds the currently logged-in user's data. Default is None. - **message_handler**: Instance of AuthHandler for displaying user-friendly messages (e.g., success, errors). - **color**: Instance of Colors for terminal text formatting.
Initialization (__init__)	Initializes the class attributes by setting paths for user and employee files, creating instances for message handling and terminal text formatting, and setting the current user to None.

Methods	Details
signup()	Handles user registration by: - Loading existing users from users.json. - Collecting inputs (name, email, password, phone number, date of birth). - Validating inputs using utility functions (e.g., validate_name, validate_email). - Checking for duplicate emails. - Assigning roles: Owner for the first user, Staff for others. - Generating a unique user_id using uuid.uuid4(). - Saving new user data to users.json. - Displaying a success message.
login()	Authenticates users by: - Loading users from users.json. - Validating email and password against stored data. - For Staff, checking the user's ID against

Methods	Details
	employee.json. - Displaying appropriate messages for success or errors.
is_user_id_valid()	Checks if the given user_id exists in employee.json for Staff users.
is_logged_in()	Returns True if a user is logged in, else False.
get_current_user_role()	Retrieves the role of the currently logged-in user (Owner or Staff). Returns None if no user is logged in.
show_all_staff()	Displays a list of all Staff users if the logged-in user is an Owner. If no Staff users exist, shows a corresponding message.
welcome_system()	Displays a welcome message using AuthHandler.

Utility Functions	Details
users(file)	Loads user data from users.json. Handles file not found and JSON decode errors gracefully.
users(file, users)	Saves updated user data to users.json. Handles write errors gracefully.

User Roles	Details
Owner	Automatically assigned to the first registered user. Can view and manage all Staff.
Staff	Assigned to subsequent users. Requires a valid ID in employee.json to log in.

Validation Functions	Details
validate_name()	Ensures the user's name meets specified requirements.
validate_email()	Ensures valid email format and checks for duplicates.

Validation Functions	Details
validate_password()	Ensures the password meets minimum length and complexity requirements.
validate_phone_number()	Ensures the phone number is 10 digits.
validate_date_of_birth()	Ensures valid date format (YYYY-MM-DD) and meets age restrictions.

File Paths	Details
users_file	Path to users.json. Stores user credentials and information.
employee_file	Path to employee.json. Stores Staff user IDs for validation.

Menu Management

Functionality	Details
Add Item	Adds a new item to the menu with validation for category, item name, price, portion sizes, and ingredients.
Update Item	Updates an existing item by modifying its name, prices, and ingredients. It handles price validations and keeps current data if no new input is provided.
Delete Item	Deletes an item based on the provided item ID.
Show Menu	Displays the entire menu with item IDs, names, prices, and ingredients.
Add Stock Ingredient	Adds an ingredient to an item and associates it with an expiry date.
Check Stock Ingredients	Checks the expiry status of ingredients for a specific item.

Stock Operations

Functionality	Details
Add Stock Ingredient	Adds a stock ingredient to an item and records the expiry date.
Check Stock Ingredients	Checks whether any ingredients in the menu items have expired. If any have expired, it notifies the user.

Item Operations

Functionality	Details
Generate Item ID	Generates a unique item ID for a given category.
Get Yes/No Input	Prompts the user for a yes/no response and validates the input.
Add Item	Handles adding a new item to the menu with category, price, ingredients, and portion sizes validation.
Update Item	Allows users to update an existing menu item with validation of item ID, category, price, and ingredients.
Delete Item	Deletes an item from the menu after validating the category and item ID.
Show Menu	Displays all items in the menu, grouped by category, with item IDs, prices, and ingredients.

ORDER MANAGEMENT MODULE

Function/Method	Description
menu()	Loads the menu from the menu_file and returns the first menu item.
orders()	Loads existing orders from order_file and returns them as a list. If the file is not found, returns an empty list.

Function/Method	Description
orders(orders)	Saves the list of orders to the order_file in JSON format.
generate_order_id()	Generates a unique order ID using the uuid library. If an error occurs, it calls an error message handler.
current_user()	Loads the current logged-in user from users_file and returns the user's data.
take_order()	Displays all orders in the system. If no orders exist, an error message is shown. Handles the process of taking an order from the customer, including choosing items, sizes, and quantities.
find_item_by_id(item_id)	Searches for an item in the menu by its item_id and returns it if found, otherwise returns None.
check_stock(item, quantity)	Checks the stock availability of an item based on its stock_ingredients and the expiration date of the ingredients.
check_order()	Allows the user to check an order by entering the order ID or mobile number.
update_order()	Allows the user to update an order by changing quantities, adding or removing items.
cancel_order()	Cancels an order by order ID or mobile number and removes it from the orders list.
find_order_by_id_or_mobile(identifier)	Searches for an order by either order ID or mobile number. Returns the order if found, otherwise None.
exit_system()	Exits the order management system and shows an exit message.
welcome_system()	Displays a welcome message when starting the system.

Key Concepts:

- **Menu Structure**: Items are categorized, and the system allows the user to specify sizes, quantities, and check stock availability.
- **Order Process**: The user can add, update, and cancel orders, while maintaining order data such as customer name, mobile number, items ordered, total price, etc.

- **Order Management**: The system stores orders in a file and provides methods for adding new orders, updating existing ones, checking status, and canceling orders.

❖ **Modules and Imports**

- **JSON**: Used for reading and writing data to files, such as storing menu data, orders, and user details.
- **UUID**: Used to generate unique order IDs using the uuid4() function.
- **Datetime**: Used for timestamping the order date and time.
- **path_manager**: This imports file paths for menu data (menu_file), order data (order_file), and user data (users_file).
- **OrderOutputHandler**: A custom class (imported from Src.Messages.order) that handles displaying various messages and outputs to the user (e.g., error messages or successful order placement).

❖ **Functions**

1. **load_menu()**:
    - Loads the menu data from the file specified in menu_file. Returns the first item from the list of menu data.
2. **load_orders()**:
    - Tries to open and load the list of orders from order_file. If the file is not found, it returns an empty list.
3. **save_orders()**:
    - Saves the current orders to the order_file in JSON format, with pretty-print indentation.
4. **generate_order_id()**:
    - Generates a unique order ID using uuid.uuid4(). It creates a hex string and slices it to take the first 6 characters as the order ID.
5. **load_current_user()**:
    - Loads user data from users_file and assumes the first user in the list is the currently logged-in user. If the file is not found, it returns an empty dictionary.

❖ **OrderManagementSystem Class**

This class encapsulates all the order management features. Here's an overview of the key methods:

- **__init__()**:
    - Loads the menu and existing orders into the class instance.
    - Initializes the OrderOutputHandler to handle outputs.
    - Loads the current user's email from the user data.

- **take_order()**:
    - Displays details of all orders in the system.
    - The main method for placing a new order. The user is prompted to enter details like customer name, mobile number, and order items.
    - It checks for item availability, item sizes, and stock. After confirming the details, the order is saved, and an order ID is generated.
    - If any issue occurs (invalid item, insufficient stock, etc.), appropriate messages are shown using OrderOutputHandler.
- **find_item_by_id()**:
    - Searches the menu for an item by its ID.
- **check_stock()**:
    - Checks if the stock of ingredients for an item is valid (not expired) before allowing the order.
- **check_order()**:
    - Allows the user to check the details of an existing order using either the order ID or mobile number.
- **update_order()**:
    - Allows updating an existing order by modifying item quantities, adding or removing items, or finishing the update.
- **cancel_order()**:
    - Cancels an order by removing it from the order list.
- **find_order_by_id_or_mobile()**:
    - Searches for an order based on the order ID or customer mobile number.
- **show_menu()**:
    - Displays the menu to the user, including item details (item ID, name, prices, and ingredients).
- **exit_system()**:
    - Outputs an exit message when the system is exiting.
- **welcome_system()**:
    - Outputs a welcome message when the system is started.

❖ **Flow of Order Management**

1. **Placing an Order**:
    - When a user calls take_order(), the system prompts them for customer details and item selection.

- The order details are validated and saved into the system.

2. **Checking and Updating Orders**:
   - Users can check orders (check_order()), view all orders (show_all_orders()), or update them (update_order()).
   - For updates, they can modify quantities, add new items, or remove items.
3. **Cancelling Orders**:
   - can be cancelled using cancel_order() by identifying it through the order ID or mobile number.
4. **Menu Management**:
   - The menu is displayed with all the items available for selection. It includes the item IDs, prices, and ingredients.

❖ **Order Handling and Validation**

- **Validation**: Input validation is carried out using the get_valid_input() function, which ensures that only valid data is entered, such as checking for valid item IDs, phone numbers, and quantities.
- **Stock Checking**: Before adding an item to the order, the system checks if sufficient stock is available, using the check_stock() method, which ensures no expired ingredients are used in the order.

❖ **Error Handling and Messages**

- The OrderOutputHandler class is responsible for displaying error messages (e.g., item not found, insufficient stock) and success messages (e.g., order placed successfully).

❖ **Key Features**

- **Order ID Generation**: Each order is assigned a unique order ID using UUID.
- **Menu Display**: Displays all menu items categorized by their types.
- **Order Tracking**: Allows tracking of orders by ID or mobile number.

Stock Validation: Ensures that orders are only placed if sufficient stock is available.

TABLE BOOKING MODULE

Component	Details
Class Name	TableBookingSystem
Purpose	Manages table reservations for a restaurant or similar setup, handling operations like creating, updating, viewing, canceling reservations, and checking table availability.
Attributes	- **reservations_file**: Path to reservations.json, storing reservation data. - **table_file**: Path to table.json, storing table information. - **tables**: List of table objects loaded from table.json. - **reservations**: List of reservation objects loaded from reservations.json. - **messages**: Instance of BookingHandler for user messages.

Methods	Details
__init__()	Initializes the system by loading tables from table.json and reservations from reservations.json. Sets up the BookingHandler instance for messaging.
tables()	Loads tables from table.json. Handles file not found or JSON errors gracefully.
reservations()	Loads reservations from reservations.json. Handles file not found or JSON errors gracefully.
reservations()	Saves updated reservation data to reservations.json. Handles write errors gracefully.
find_available_table()	Finds tables that meet the seat requirement and are available at the given date and time. Returns a list of suitable tables.
suggest_table()	Suggests tables that can accommodate the given number of seats, sorted by seating capacity.
split_table_reservation()	Suggests multiple tables for reservations exceeding a single table’s capacity. Returns a list

Methods	Details
	of tables if successful or None if splitting isn’t possible.
reserve_table()	Creates a new reservation: - Collects and validates user input (name, phone, date, seats). - Finds an available table or suggests alternatives. - Handles reservations requiring multiple tables. - Saves the reservation.
cancel_reservation()	Cancels a reservation by updating its status to "Canceled". Requires a valid reservation ID.
update_reservation()	Updates the date and time of an existing reservation. Finds available tables for the new date and time and modifies the reservation accordingly.
view_availability()	Displays tables available for a given date, time, and seat requirement.
view_all_reservations()	Lists all current reservations. Displays a message if no reservations are found.
search_reservation_by_id()	Searches for a reservation using its ID. Displays reservation details if found, otherwise shows an error message.
exit_system()	Displays an exit message and terminates the system.
welcome_system()	Displays a welcome message using BookingHandler.

Utility Classes	Details
Reservation	Represents a reservation with attributes like reservation_id, name, phone, date_time, seats, table_id, and status (default: "Reserved").
Table	Represents a table with attributes like table_id and seats.

Validation Functions	Details
validate_name()	Ensures the name meets specified requirements.
validate_phone_number()	Ensures the phone number is valid.
validate_booking_date_time()	Ensures the booking date and time are valid.
validate_seats()	Ensures the seat count is valid (non-negative integer within capacity limits).
validate_booking_id()	Ensures the booking ID format is valid.

File Paths	Details
reservations_file	Path to reservations.json, storing all reservation data.
table_file	Path to table.json, storing all table information.

INVOICE

Billing System Methods

Method Name	Purpose	Additional Details
orders	Loads orders from a JSON file and converts them into Order objects.	Displays warning if file not found or invalid.
orders	Saves current Order objects to a JSON file.	Handles exceptions gracefully and notifies users.
bills	Loads bills from a JSON file and returns them as dictionaries.	Displays warning if file not found or invalid.
bills	Saves current bills to a JSON file.	Handles exceptions gracefully and notifies users.
generate_bill	Creates a Bill object for a specified Order and saves it to the bills list.	Updates the order status to "Billed."
check_bill	Retrieves and displays details of a bill by billing_id.	Notifies if no matching bill is found.
show_all_bills	Displays all bills stored in the system.	Notifies if no bills are available.
mark_as_paid	Marks a bill as paid, updates the payment type, and records the payment date.	Validates payment amount and type.
show_all_paid_bills	Displays all bills with the status "Paid."	Notifies if no paid bills are available.
exit_system	Exits the billing system gracefully.	-
welcome_system	Displays a welcome message for the billing system.	-

Bill Class Attributes

Attribute Name	Description	Example Value
billing_id	Unique identifier for the bill, derived using a UUID.	"12ab34"
order_id	ID of the associated order.	"ORD12345"
customer_name	Name of the customer for the bill.	"John Doe"
billing_date	Date and time when the bill was generated.	"20-Nov-2024 03:30 PM"
items	List of ordered items included in the bill.	["Burger", "Fries"]
subtotal	Total price of items before GST.	500.0
gst	GST amount calculated as 18% of the subtotal.	90.0
total	Total price including GST.	590.0
status	Current status of the bill.	"Billed"
payment_type	Method of payment (e.g., Online or Cash).	"Cash"
payment_date	Date and time of payment, if made.	"20-Nov-2024 04:00 PM"
user_email	Email address of the user associated with the bill (optional).	"customer@example.com"

Order Class Attributes

Attribute Name	Description	Example Value
order_id	Unique identifier for the order.	"ORD12345"
customer_name	Name of the customer who placed the order.	"John Doe"
mobile_number	Customer's mobile number.	"9876543210"
order_items	List of items included in the order.	["Burger", "Fries"]
total_order_price	Total price of the order.	500.0
order_date	Date when the order was placed.	"20-Nov-2024"
status	Current status of the order (e.g., Processing, Billed).	"Processing"
user_email	Email address of the user associated with the order (optional).	"customer@example.com"

Example Relationships

Order ID	Customer Name	Billing ID	Status	Total	Payment Type	Payment Date
ORD12345	John Doe	12ab34	Paid	₹590.00	Online	20-Nov-2024 04:00 PM

INVOICE PREVIEW

BILL DETAILS

==========

Billing ID : 15a906

Order ID : 402d80

Customer Name : Amayra

Billing Date : 20-Nov-2024 01:57 PM

Items Purchased:

1. Veg Spring Rolls (Size: Single) x2 = ₹200.00

Subtotal : ₹200.00

GST (18%) : ₹36.00

Total : ₹236.00

STAFF MANAGEMENT

Section	Code/Component	Explanation
Initialization	Salary Management. __init__	Loads employee data into self.employees. Logs exceptions if the file loading fails, and sets self.employees to an empty list.
Method: pay_salary	- Input: Employee ID - Validates days worked and calculates salary - Saves payment record	Handles salary payment calculation. Validates inputs (days worked, bonus), creates payment history, and logs any errors during the process.
Method: show_salary_history	- Input: Employee ID - Displays salary payment history	Retrieves and displays payment history for the specified employee. Logs exceptions if errors occur in fetching or processing data.
Exception Handling	try-except blocks in all methods	Catches and logs exceptions in file operations, data processing, or user input handling using self.logger.exception.
Staff Authentication	Staff Management System. Authenticate_user	Authenticates a user by checking email and password from users.json. Logs errors during authentication if exceptions occur.
Method: add_employee	- Input: Employee details - Creates a new Employee object	Adds a new employee by validating user ID and details. Logs errors if file operations or input processing fails.
Class: Employee	Represents an employee with attributes like ID, name, email, salary, etc.	Defines employee details. Converts them into a dictionary with to_dict() for easy serialization to JSON.
Utility Functions	load_data(file_path) save_data(file_path, data)	Handles JSON file operations (loading and saving). Catches file-related exceptions in the respective methods.
Salary Payment Details	Payment record includes: - date, amount, bonus, reduction_days	Each payment is logged into the employee's salary_payment_history. This helps in tracking payment history accurately.

Key Improvements and Features

Aspect	How It Works/Improves
Centralized Logging	A single logging utility (logger.py) is used across modules for consistency and clarity.
Resilient Error Handling	The try-except blocks ensure the application doesn't crash on errors and logs the root cause.
Extensibility	Easily extendable with additional modules or functionalities due to modular design.
Code Readability	Classes and methods are well-structured, making it easy to understand responsibilities.
Data Integrity	Employee and salary records are consistently validated and saved, preventing data corruption.

DATABASE

USERS.JSON

SL NO.	USERS.JSON	EXPLANATION
1	[{ "id": "4869327A", "name": "Priyanshu", "email": "priyanshu123@gmail.com", "password": "Pr2004", "phone": "7255965594", "role": "Staff", "date_of_birth": "2004-06-20" }]	"id": "4869327A" — A unique identifier for Priyanshu's account.
		"name": "Priyanshu" — The full name of the user.
		"email": "priyanshu123@gmail.com" — Priyanshu's email address for communication and login.
		"password": "Pr2004" — The password associated with this account (ideally stored securely).
		"phone": "7255965594" — Priyanshu's contact phone number.
		"role": "Staff" — Indicates Priyanshu is a staff member in the organization.
		"date_of_birth": "2004-06-20" — Priyanshu's date of birth in YYYY-MM-DD format.

MENU.JSON

SL NO.	MENU.JSON	EXPLAINATION

1	[{
	"item id": "TE03",
	"item name": "Lemon Tea",
	"prices": {
	"single": 35
	},
	"ingredients": [
	"tea leaves",
	"lemon",
	"water"
	],
	"stock_ingredients": {
	"tea leaves": "2024-12-18",
	"lemon": "2024-12-17",
	"water": "2024-12-15"
	}
	}]
		Opening bracket for the item object.
		"item id": "TE03" — A unique identifier for the item, in this case, Lemon Tea.
		"item name": "Lemon Tea" — The name of the item being described.
		"prices": { "single": 35 } — Specifies the price of a single serving of Lemon Tea as ₹35.
		"ingredients": ["tea leaves", "lemon", "water"] — Lists the ingredients used to prepare Lemon Tea.
		"stock_ingredients": { "tea leaves": "2024-12-18", — Expiry date for the ingredient tea leaves.
		"lemon": "2024-12-17" — Expiry date for the ingredient lemon.
		"water": "2024-12-15" — Expiry date for the ingredient water.
		Closing bracket for the item object.

ORDER.JSON

SL No.	ORDER.JSON	Value	Explanation
1	{	"eab708"	Unique identifier for the order.
2	"order_id": "eab708",	"khushi"	Name of the customer who placed the order.
3	"customer_name": "khushi",	"8654321123"	Contact number of the customer.
4	"mobile_number": "8654321123",	List of items included in the order.	Contains details of each item, including ID, name, size, quantity, and price.
5	"order_items": [	"DR01"	Unique identifier for the item (e.g., Mango Lassi).
6	{	"Mango Lassi"	Name of the item ordered.
7	"item_id": "DR01",	"single"	Size of the item (e.g., single portion).
8	"item_name": "Mango Lassi",	2	Number of units of this item ordered.
9	"size": "single",	80	Total price for this specific item in the order.
10	"quantity": 2,	"ST03"	Unique identifier for the next item (e.g., Paneer Tikka).
11	"total_price": 80	"Paneer Tikka"	Name of the next item ordered.
12	},	"single"	Size of the item (e.g., single portion).
13	{	2	Number of units of Paneer Tikka ordered.
14	"item_id": "ST03",	260	Total price for Paneer Tikka in the order.
15	"item_name": "Paneer Tikka",	"RI01"	Unique identifier for the next item (e.g., Steamed Rice).
16	"size": "single",	"Steamed Rice"	Name of the next item ordered.
17	"quantity": 2,	"half"	Size of the item (e.g., half portion).
18	"total_price": 260	1	Number of units of Steamed Rice ordered.
19	},	70	Total price for Steamed Rice in the order.
20	{	410	Sum of all item prices in the order.
21	"item_id": "RI01",	"19-Nov-2024 08:12:PM"	Date and time when the order was placed.
22	"item_name": "Steamed Rice",	"Billed"	Current status of the order (e.g., Billed, Processing, etc.).
23	"size": "half", "quantity": 1, "total_price": 70 }], "total_order_price": 410, "order_date": "19-Nov-2024 08:12:PM", "status": "Billed", "user_email": "anand123@gmail.com" }	"anand123@gmail.com"	Email address associated with the user who placed the order.

Reservation JSON

SL NO	JSON	EXPLANATION

1	[
	{
	"reservation_id": "8765fa65",
	"name": "sachin",
	"phone": "8765456678",
	"date_time": "2024-11-15 9:00 pm",
	"seats": 8,
	"table_id": "T15",
	"status": "Reserved"
	}
	]
		reservation_id: Unique identifier for the reservation. This helps to distinguish it from other reservations.

2		name: The name of the person who made the reservation.
3		phone: Contact phone number of the person who made the reservation.
4		date_time: The date and time of the reservation (in this case, 9:00 PM on November 15, 2024).
5		seats: The number of seats reserved for the party (in this case, 8 seats).
6		table_id: The identifier for the table assigned for the reservation (in this case, Table T15).
7		status: The current status of the reservation. "Reserved" indicates the reservation is confirmed.

BILLING JSON

SL NO	JSON	EXPLANATION

1	[{
	"billing_id": "1aca8d",
	"order_id": "4a1d24",
	"customer_name": "anjali",
	"billing_date": "19-Nov-2024 09:12:PM",
	"items": [
	{
	"item_id": "RI02",
	"item_name": "Jeera Rice",
	"size": "half",
	"quantity": 2,
	"total_price": 110
	},
	{
	"item_id": "DR01",
	"item_name": "Mango Lassi",
	"size": "single",
	"quantity": 2,
	"total_price": 80
	}
	],
	"subtotal": 190,
	"gst": 34.199999999999996,
	"total": 224.2,
	"status": "Billed",
	"payment_type": null,
	"payment_date": null,
	"user_email": null
	}]
		billing_id: Unique identifier for the billing record. This helps to distinguish it from other billing transactions.
		order_id: Identifier for the associated order.
		customer_name: Name of the customer who made the order (in this case, Anjali).
		billing_date: The date and time when the bill was generated (in this case, 19-Nov-2024, 9:12 PM).
		items: List of items in the order with details like item ID, name, size, quantity, and total price.
		subtotal: The total price before applying any taxes (₹190 in this case).
		gst: Goods and Services Tax calculated on the subtotal (₹34.20 in this case).
		total: The final amount to be paid after adding GST to the subtotal (₹224.20 in this case).
		status: The current billing status. "Billed" indicates the bill has been generated.
		payment_type: Payment method used (not specified in this case).
		payment_date: The date and time when the payment was made (not specified in this case).
		user_email: Email address of the customer associated with the billing (not specified in this case).

EMPLOYEE.JSON

SL NO	JSON	EXPLANATION
1	{ "billing_id": "1aca8d", "order_id": "4a1d24", "customer_name": "anjali", "billing_date": "19-Nov-2024 09:12:PM", "items": [{ "item_id": "RI02", "item_name": "Jeera Rice", "size": "half", "quantity": 2, "total_price": 110 }, { "item_id": "DR01", "item_name": "Mango Lassi", "size": "single", "quantity": 2, "total_price": 80 }], "subtotal": 190, "gst": 34.2, "total": 224.2, "status": "Billed", "payment_type": null, "payment_date": null, "user_email": null }	billing_id: Unique identifier for the bill, distinguishing it from other billing entries. order_id: Unique identifier for the order associated with the bill. customer_name: The name of the customer who placed the order. billing_date: The date and time when the bill was generated. items: A list of items ordered by the customer, with details like item_id, item_name, size, quantity, and total_price. subtotal: The total price of the items before taxes. gst: The Goods and Services Tax (GST) applied to the order. total: The total amount after including GST. status: Indicates the current billing status, e.g., "Billed". payment_type: Type of payment used (e.g., Cash, Card, etc.), which is null if not yet paid. payment_date: Date when the payment was made, which is null if not paid yet. user_email: The email of the customer, which is null if not provided.

TABLE JSON

SL NO	JSON	EXPLANATION
1	table_id: "T1", "seats": 2	table_id: A unique identifier for each table. For example, "T1" refers to Table 1. seats: The number of seats available at the table. For Table T1, it has 2 seats.
2	table_id: "T2", "seats": 2	Similarly, Table T2 has 2 seats.
3	table_id: "T3", "seats": 2	Table T3 also has 2 seats.
4	table_id: "T4", "seats": 2	Table T4 is another 2-seater table.
5	table_id: "T5", "seats": 4	Table T5 has 4 seats.
6	table_id: "T6", "seats": 4	Table T6 has 4 seats.
7	table_id: "T7", "seats": 4	Table T7 is a 4-seater.
8	table_id: "T8", "seats": 4	Table T8 is also a 4-seater.
9	table_id: "T9", "seats": 4	Table T9 is another 4-seater table.
10	table_id: "T10", "seats": 6	Table T10 has 6 seats.
11	table_id: "T11", "seats": 6	Table T11 is a 6-seater.
12	table_id: "T12", "seats": 6	Table T12 has 6 seats.

SL NO	JSON	EXPLANATION
13	table_id: "T13", "seats": 6	Table T13 is another 6-seater.
14	table_id: "T14", "seats": 8	Table T14 has 8 seats.
15	table_id: "T15", "seats": 8	Table T15 is an 8-seater.

General Explanation:

- **tables**: This is an array of objects where each object represents a table in the restaurant.
- **table_id**: A unique identifier for each table, such as "T1", "T2", etc.
- **seats**: The number of seats available at the corresponding table. It indicates how many people the table can accommodate.

MESSAGES

Message Function	Description
welcome_message	Shows a welcome message for any specific system (e.g., Order Management, Staff Management).
exit_message	Displays a goodbye message when exiting a specific system.
show_order_placed	Displays a success message for order placement, showing order ID and total price.
item_not_found	Displays a message indicating the requested menu or inventory item was not found.
invalid_size	Shows a message for invalid size selection (e.g., half/full for rice).

Message Function	Description
insufficient_stock	Indicates insufficient stock availability for a requested item.
show_order_details	Displays the details of a specific order, such as items, prices, and total.
show_menu_item	Formats and displays details of a menu item, including ID, name, price, and ingredients.
order_update_success	Indicates that an order was successfully updated.
order_cancel_success	Indicates that an order was successfully canceled.
order_not_found	Displays a message when no order is found with the given ID.
order_status	Displays the current status of a specific order (e.g., pending, completed).
invalid_input	Shows a message for invalid or missing input in a specific field.
order_save_error	Indicates an error occurred while saving the order, with details of the exception.
no_orders_found	Displays a message when no orders are found.
display_inventory	Lists the details of inventory items, including stock levels and units.
inventory_update_success	Indicates a successful inventory update for a specific item.
invalid_quantity	Displays a message for invalid quantity input in inventory management.
generate_message	Displays an error message for unique order ID or reservation ID generation.

Message Function	Description
no_items_in_order	Shows a message indicating no items were added to the order.
profile_updated	Confirms that a user profile or employee profile was updated successfully.
invalid_credentials	Indicates invalid login credentials were entered during authentication.
employee_added	Confirms a successful addition of a new employee.
employee_not_found	Indicates the specified employee could not be found in the system.
employee_deleted	Confirms successful deletion of an employee from the system.
no_employees_found	Indicates no employees are available in the system.
display_profile	Displays detailed profile information for an employee, including salary details.
display_all_profiles	Displays all employee profiles with sequential numbering.
show_reservations_header	Displays a header message for reservations on a specified date.
show_reservation_details	Displays details of a specific reservation, such as time, table, and customer name.
no_reservations_found	Indicates no reservations were found for a specific date.
show_canceled_reservations_header	Displays a header for canceled reservations.
show_canceled_reservation_details	Displays details of a specific canceled reservation.
no_canceled_reservations_found	Indicates no canceled reservations were found.

Message Function	Description
show_order_summary_header	Displays a header for an order summary for a specified date.
show_item_summary	Displays a summary of items ordered, including quantity and name.
show_expiring_ingredients_header	Displays a header for ingredients nearing expiration.
show_expiring_ingredient_entry	Displays details of an expiring ingredient, including category and expiration date.
no_expiring_ingredients	Indicates no ingredients are nearing expiration in the specified timeframe.
show_low_stock_header	Displays a header for low-stock ingredients.
show_low_stock_entry	Displays details of a low-stock ingredient.
no_low_stock_ingredients	Indicates no ingredients are low in stock.
show_summary_header	Displays a header for a summary of expiring and low-stock ingredients.
show_summary_category	Displays the category of ingredients in the summary.
show_summary_item	Displays details of a specific item in the ingredient summary.
no_summary	Indicates no data is available for expiring or low-stock ingredients.
reservation_success	Displays a message confirming successful table reservation with reservation ID.
reservation_error	Displays an error message when a reservation fails, with an exception reason.
invalid_date_time_format	Shows a message for invalid date-time input during table booking.

Message Function	Description
table_unavailable	Indicates that the requested table is not available for booking.
reservation_update_success	Confirms that a reservation was successfully updated.
reservation_cancel_success	Indicates a reservation was successfully canceled.
salary_update_success	Displays a message confirming successful salary update for an employee.
menu_update_success	Indicates that a menu item was successfully updated.
menu_add_success	Confirms successful addition of a new menu item.
menu_delete_success	Confirms successful deletion of a menu item.
ingredient_added_success	Displays a success message when a new ingredient is added to the inventory.
ingredient_deleted_success	Confirms successful deletion of an ingredient from the inventory.

UTILITY

Authentication System Files:

File Name	Description
auth.py	Handles the main authentication logic, including signup, login, and validation.
user.py	Manages user data, validation, and role-based access control.
users.json	Stores user data such as ID, name, username, password, role, etc.

Dashboard Files:

File Name	Description
owner_dashboard.py	Contains the owner dashboard with options for managing menu, orders, staff, and more.
staff_dashboard.py	Contains the staff dashboard with options like order management and table booking.
main.py	Entry point for the dashboard, calls required modules and navigates based on roles.

Order Management Files:

File Name	Description
order_management.py	Handles creating, updating, canceling, and searching orders.
order.py	Contains the structure and data related to an individual order.
orders.json	Stores order information, including items, quantities, status, etc.

Menu Management Files:

File Name	Description
menu_management.py	Manages the menu operations, such as adding, updating, deleting items.
menu.py	Contains the structure for menu items, including their details (ID, name, price).
menu.json	Stores the menu items and their details like name, price, and ingredients.

Table Booking Files:

File Name	Description
table_booking.py	Handles table reservation, cancellation, updates, and availability checks.
reservation.py	Contains the structure and data related to individual table reservations.
reservations.json	Stores reservation information, such as table numbers, dates, and customer details.

Inventory Management Files:

File Name	Description
inventory_management.py	Manages the inventory, including adding and updating ingredients.
ingredient.py	Contains the structure for ingredients, their stock, and expiration data.
inventory.json	Stores inventory data, including ingredient names, quantities, and expiration dates.

Billing System Files:

File Name	Description
billing.py	Handles generating, displaying, and managing bills for orders.
invoice.py	Contains the structure for invoices, including details like itemized costs.
bills.json	Stores bill details and payment statuses.

Salary Management Files:

File Name	Description
salary_management.py	Manages employee salary, payments, and records.
employee.py	Contains the structure for employee details like name, position, and salary.
salary.json	Stores salary records and employee payment histories.

Reports Files:

File Name	Description
reports.py	Generates various reports, such as order and bill reports.
report.json	Stores the generated reports in JSON format.

Utility Files:

File Name	Description
merge.py	Merges or integrates different features, like combining menu and order functionalities.
utils.py	Contains helper functions and utilities for various system tasks.

Data Files:

File Name	Description
menu.json	Stores all menu items, prices, and ingredients.
users.json	Stores user data (Owner, Staff, User), including roles, IDs, and passwords.
orders.json	Stores order data, including order details and statuses.

File Name	Description
reservations.json	Stores table reservation details.
inventory.json	Stores ingredient data, stock quantities, and expiration dates.
bills.json	Stores bill data, including generated bills and payment status.
salary.json	Stores salary data for employees.

This table format organizes the various modules and files within your system, detailing their role and functionality.

MAIN

Code Component	Explanation
from Src.Dashboard.manage_dashboard import main	Imports the main function from the manage_dashboard module located in the Src/Dashboard directory.
if __name__ == "__main__":	Ensures the script runs only when executed directly, not when imported as a module in another script.
main()	Calls the main function, which likely initializes and controls the primary flow of the application.

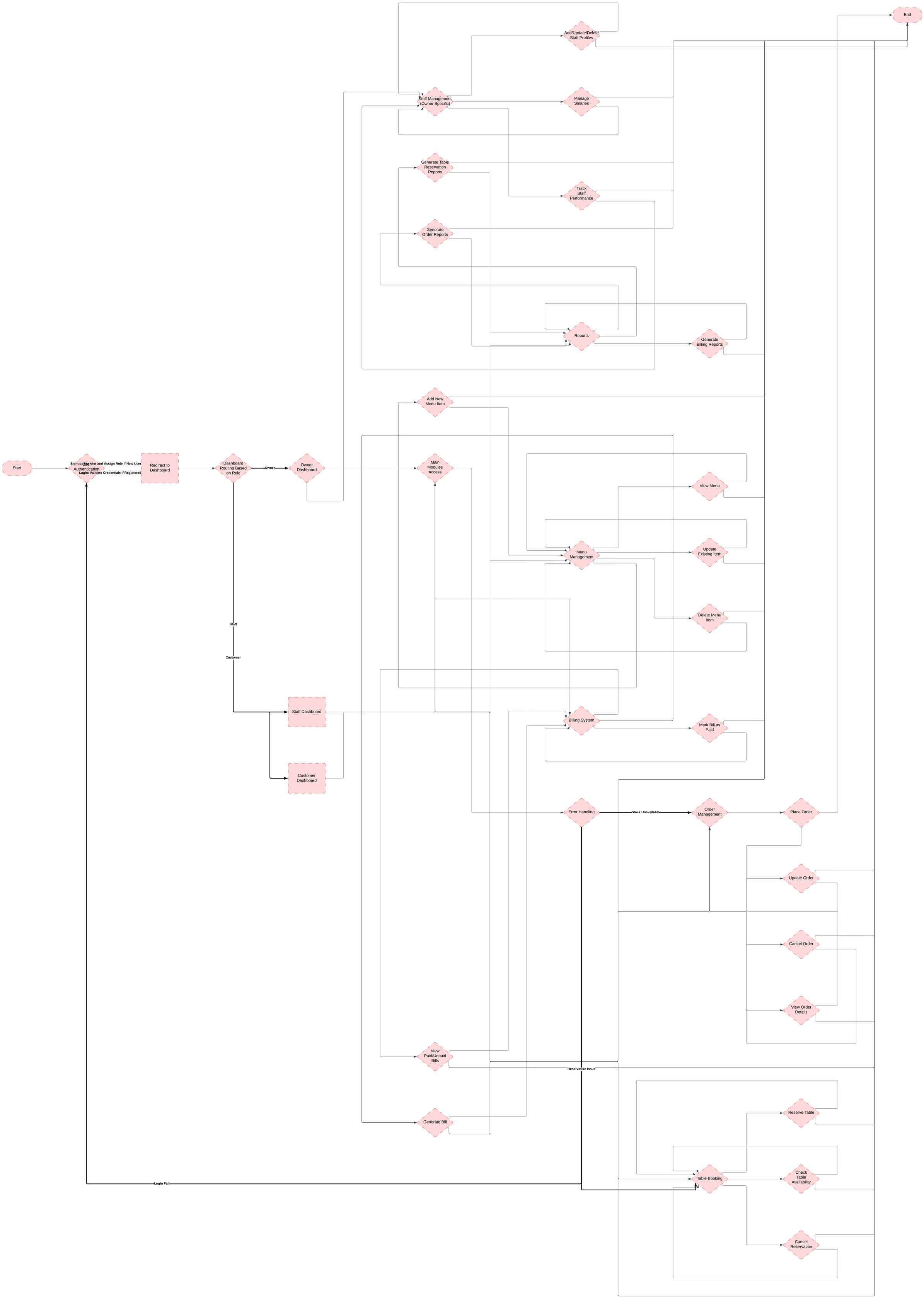
❖ Purpose

This script acts as the starting point for your restaurant management system. It allows the program to execute the main function, which probably:

1. Initializes the dashboard.
2. Routes to different functionalities like user login, menu management, order placement, billing, or reporting.

❖ Behaviour

- When executed directly, the program will start with the logic defined in main().
- If imported elsewhere, it won't execute the main() function, allowing modular use of its components.


End
Add/Update/Delete Staff Profiles
Staff Management (Owner Specific)
Manage Salaries
Generate Table Reservation Reports
Track Staff Performance
Generate Order Reports
Reports
Generate Billing Reports
Add New Menu Item
Start
Signup: Register and Assign Role if New User
Authentication
Login: Validate Credentials if Registered
Redirect to Dashboard
Dashboard Routing Based on Role
Owner
Owner Dashboard
Main Modules Access
View Menu
Menu Management
Update Existing Item
Delete Menu Item
Staff
Customer
Staff Dashboard
Customer Dashboard
Billing System
Mark Bill as Paid
Error Handling
Stock Unavailable
Order Management
Place Order
Update Order
Cancel Order
View Order Details
View Paid/Unpaid Bills
Reservation Issue
Generate Bill
Reserve Table
Table Booking
Check Table Availability
Cancel Reservation
Login Fail