Taste of Tradition Management System

Technical Documentation

❖Project Overview

- Project Name: Taste of Tradition Management System
- Team Number: 05
- Team Members Name: Khushi, Anand
- Date: 2024-11-19

❖Project Summary

The **Taste of Tradition Management 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

   - **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: [Python.org](https://python.org)
2. Download Project	Download or clone the project repository.	- Clone: git clone <repository_url> - Extract if downloaded as a zip file.
3. Navigate	Move to the project directory in the terminal or command prompt.	cd <project_directory>
4. Create Virtual Environment (Optional)	Set up an isolated environment for dependencies.	- Create: python -m venv venv - Activate (Linux/macOS): source venv/bin/activate - Activate (Windows): venv\Scripts\activate
5. Install Dependencies	Install required Python libraries. If requirements.txt exists, install dependencies automatically.	pip install -r requirements.txt
6. Manually Install Libraries	If requirements.txt is unavailable, install libraries manually (e.g., uuid, maskpass).	Example: pip install maskpass
7. Set Up Database Files	Ensure all required JSON files (e.g., users_file, menu_file) are in the correct paths or create them if missing.	Initialize files with empty lists ([]) or sample data if necessary.
8. Run the Project	Execute the main script to start the system.	python main.py (Replace main.py with the actual entry point, e.g., Src/Dashboard/manage_dashboard.py)

Section	Description
Imports and Dependencies	
import uuid	Imports uuid module for generating unique user IDs.
import maskpass	Uses maskpass to securely handle password input by masking characters.
from Src.Authentication.file_operation import load_users, save_users	Imports functions for reading (load_users) and writing (save_users) user data to a JSON file.
from Src.Utility.validation	Imports various validation functions, such as validate_name, validate_email, validate_password, validate_phone_number, and validate_date_of_birth.
from Src.Utility.user_input import get_valid_input	Imports a function to manage user input and validation during signup and login.
from Src.Utility.path_manager import users_file	Specifies the path to the JSON file that stores user data (users_file).
from Src.Messages.authentication import AuthHandler	Imports AuthHandler to handle messages shown to the user (e.g., after signup or login).
from Src.Utility.color import Colors	Imports Colors for color-coded console output to improve message readability.

❖AuthSystem Class

- **__init__**
  - Initializes the AuthSystem class:
    - Sets the path to users_file.
    - Sets current_user to None.
    - Creates instances of AuthHandler for messaging and Colors for color-coded console output.
- **signup**

  - Manages the signup process for new users, including validation and uniqueness checks:
    - **Load Users**: Loads the list of users from users_file via load_users.
    - **Name Input**: Prompts for and validates the user's name.
    - **Email Input**: Validates email for uniqueness, ensuring no duplicate entries.
    - **Password Input**: Uses maskpass for secure input; validates password criteria with validate_password.
    - **Phone & Date of Birth (DOB)**: Collects and validates the phone number and date of birth.
    - **Role Assignment**: Automatically assigns the "Owner" role to the first user; subsequent users are assigned the "Staff" role.
    - **User ID Generation**: Generates a unique 8-character user ID using uuid.
    - **User Creation**: Appends the new user to the users list and saves it via save_users.
    - **Feedback**: Displays a signup success message using AuthHandler.
- **login**
  - Manages user login by verifying credentials:
    - **Email & Password Validation**: Checks if the email and password match an existing user.
    - **User Identification**: Sets self.current_user to the matched user's data if authentication is successful.
    - **Feedback**: Displays success or failure messages based on login outcome.
- **is_logged_in**
  - Returns True if a user is currently logged in (self.current_user is not None), otherwise returns False.
- **get_current_user_role**
  - Retrieves the role of the logged-in user from self.current_user['role']. Returns None if no user is logged in.
- **show_all_staff**
  - Displays a list of "Staff" users, if an "Owner" exists in the system:
    - **Owner Check**: Verifies at least one "Owner" is registered.
    - **Staff List Display**: Collects and displays details for each "Staff" user or shows a message if none are found.
- **welcome_system**

- Displays a welcome message by calling welcome_message from AuthHandler.

❖ **Supporting Functions**

- **load_users**
  - Loads and returns user data from the JSON file:
    - Handles missing or unreadable file scenarios using AuthHandler for error messages.
- **save_users**
  - Saves the updated list of users to the JSON file:
    - Includes error handling for potential exceptions during the save process.

TABLE BOOKING MODULE

Class/Method	Description
TableBookingSystem	**The main class for handling table reservations and managing data.**
__init__	Initializes TableBookingSystem, sets paths to reservations_file and table_file, loads tables and reservations, and creates an instance of BookingHandler for messages.
load_tables	Loads tables from table_file. Parses tables into Table objects on success; displays a message if tables are missing or unreadable.
load_reservations	Loads reservations from reservations_file. Parses reservations into Reservation objects on success; displays a message if reservations are missing or unreadable.
save_reservations	Saves reservations to reservations_file with error handling to display an error message if saving fails.
find_available_table	Finds tables with specified seat capacity and checks availability for a given date_time. Returns a list of available tables.
Core Reservation Methods	

Class/Method	Description
reserve_table	Collects inputs (name, phone, date_time, seats) and finds available tables. Creates a reservation with a unique ID if a table is available, saves, and confirms success. Offers option to make additional reservations.
cancel_reservation	Prompts for booking_id to cancel, finds active reservation by ID, changes status to "Canceled" if found, saves, and provides feedback to confirm cancellation or failure.
update_reservation	Collects booking_id and new date_time, searches for reservation by ID, checks table availability for the new date, updates reservation if available, saves, and confirms outcome.
Utility & Additional Methods	
view_availability	Collects date_time and seats to check availability. Lists available tables for specified time and seat count or displays a message if none are available.
view_all_reservations	Displays all reservations, or shows a message if no reservations are found.
search_reservation_by_id	Collects booking_id, finds and displays matching reservation, or shows a not-found message.
exit_system	Displays an exit message.
welcome_system	Displays a welcome message.
Supporting Classes	
Reservation	**Class representing individual reservations.**
__init__	Sets reservation_id, name, phone, date_time, seats, table_id, status (default: "Reserved").
__repr__	Returns a formatted string summarizing reservation details.
Table	**Class representing individual tables.**
__init__	Sets table_id and seats.
__repr__	Returns a formatted string summarizing table details (table ID and seat count).

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	OwnerDashboard	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	StaffDashboard	Contains methods for handling operations available to Staff, such as Order Management, Invoice Management, Table Booking Management, Stock Management, and accessing their Profile.
Menu Management	MenuManagement	Allows adding, updating, deleting menu items and ingredients, viewing the menu, and checking stock of ingredients. Called within the Owner dashboard.
Order Management	OrderManagementSystem	Manages taking, updating, canceling orders, checking specific orders, and

Component	Class / Function	Description
		showing all orders. Available for both Owner and Staff dashboards.
Invoice Management	BillingSystem	Handles generating and viewing bills, marking as paid, and viewing all bills. Available in both dashboards.
Table Booking Management	TableBookingSystem	Provides table reservation, cancellation, update, and viewing availability or all reservations. Used by both Owner and Staff dashboards.
Stock Management	Report, ExpirationReport, ReservationReport, 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.

INVOICE

Class	Attribute/Method	Description
BillingSystem	**Attributes**	
	orders	List of order objects, loaded from order_file.
	bills	List of bill records, loaded from bill_file.
	bill_handler	Instance of BillingHandler to handle billing-related messages and prompts.
	Methods	
	load_orders()	Loads orders from the order_file JSON and creates Order instances.

Class	Attribute/Method	Description
	save_orders()	Saves the current orders list to order_file.
	load_bills()	Loads bills from the bill_file JSON, handling potential JSON errors.
	save_bills()	Saves the current bills list to bill_file.
	generate_bill()	Generates a bill for an order with status "Processing", updates the order status to "Billed", and saves the order and bills.
	check_bill()	Checks and displays a bill by its billing ID.
	show_all_bills()	Displays all bills if available.
	mark_as_paid()	Marks a bill as "Paid" after validating the payment type and amount. Updates payment details and bill status.
	show_all_paid_bills()	Displays all bills with the status "Paid".
	exit_system()	Displays an exit message.
	welcome_system()	Displays a welcome message.
Bill	**Attributes**	
	billing_id	Unique billing identifier (UUID-based).
	order_id	Associated order's ID.
	customer_name	Name of the customer.
	billing_date	Date and time the bill was generated.
	items	List of items in the order.
	subtotal	Total price of items before tax.
	gst	Calculated GST (18%).
	total	Final total after adding GST.
	status	Initial status set to "Billed".
	payment_type	Payment type (Online/Cash), set when paid.
	payment_date	Payment date and time, set when paid.
	user_email	Email of the user associated with the order (if applicable).

Class	Attribute/Method	Description
	Method	
	to_dict()	Converts the Bill instance into a dictionary format for easy JSON storage.
Order	**Attributes**	
	order_id	Unique identifier for the order.
	customer_name	Customer's name.
	mobile_number	Customer's contact number.
	order_items	List of items in the order.
	total_order_price	Total price of the items in the order.
	order_date	Date the order was placed.
	status	Status of the order (e.g., "Processing" or "Billed").
	user_email	Email associated with the user (if applicable).
	Method	
	to_dict()	Converts the Order instance into a dictionary for easy JSON storage.
Billing Workflow	**Description**	
	Loading Data	Orders and bills are loaded from JSON files when BillingSystem is initialized.
	Bill Generation	A bill is generated for an eligible order (with status "Processing"), calculates total and GST, and updates the order status to "Billed". Then it saves both orders and bills.
	Payment Processing	A bill can be marked as "Paid" after verifying the payment type and amount. Updates the payment details and bill status.
	Viewing Bills	Users can view a bill by billing ID, see all bills, or filter to view only paid bills.
	Exit/Welcome Messages	Displays informative exit and welcome messages using BillingHandler.

BILL PREVIEW

BILL DETAILS

==

Billing ID: B12345

Order ID: O98765

Customer Name: John Doe

Billing Date: 2024-11-18

Items Purchased:

1. Veg Pizza (Size: Medium) x2 = ₹400.00

2. Soft Drink (Size: 500ml) x1 = ₹50.00

3. Ice Cream (Size: Single) x3 = ₹150.00

Subtotal: ₹600.00

GST (18%): ₹108.00

Total: ₹708.00

Status: Paid

Payment Type: Credit Card

Payment Date: 2024-11-18

==

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.

Functionality	Details
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.

MESSAGE MODULE

Class	Functions
AuthHandler	Handles messages related to authentication (e.g., email already taken, login success, staff limit, etc.).
BookingHandler	Manages messages related to table bookings (e.g., reservation success, no available tables, reservation updates, etc.).
BillingHandler	Handles billing and payment-related messages (e.g., bill generation, payment success/failure, invalid payment, etc.).
Menu_Message	Provides feedback for menu item management (e.g., item added, item updated).

❖ **AuthHandler Class:**

Handles authentication-related messages such as login success, signup success, and errors related to users.

- email_already_taken: Alerts when the email is already in use.
- owner_exists: Alerts when there is already an owner.
- signup_successful: Confirms a successful signup.
- login_successful: Confirms a successful login with the user's role.

❖ **BookingHandler Class:**

Handles messages related to reservations and bookings.

- display_reservation_success: Confirms reservation success with a booking ID.
- display_no_available_tables: Alerts if there are no tables available for the requested seats at a specific time.
- display_reservations_on_date: Displays reservations on a specific date.

❖ **BillingHandler Class:**

Handles messages related to billing and payment.

- display_warning_invalid_format: Warns if the billing data is empty or has invalid format.
- display_order_not_found: Alerts if an order ID is not found.
- display_bill_generated: Confirms bill generation for an order.
- display_payment_success: Confirms successful payment.

❖ **Menu_Message Class:**

Handles feedback messages related to menu items.

- print_item_added: Confirms when an item is added to the menu.
- print_item_updated: Confirms when an item is updated.

Code Workflow

❖ **Order Management:**

- When an order is placed, an entry is created with a unique order ID and customer details.
- Ordered items, quantities, and total prices are stored in the system.
- The order status changes as the order progresses (e.g., from "pending" to "completed").

❖ **Inventory Management:**

- The system tracks the stock level of each item.
- When items are used or sold (e.g., when an order is completed), the stock is updated.
- The inventory data may also reflect the unit of measurement (pieces, bottles, etc.).

❖ **Employee Profile:**

- Employee data is stored and linked to roles (e.g., waiter, chef).
- Each employee has a salary, status (active/inactive), and their profile details like contact information and address.
- The system helps in managing payroll, salaries, and the employee status.

❖ **Reservations:**

- The system manages customer reservations, with tables assigned to specific customers for specific dates/times.
- It keeps track of the reservation status and updates it when necessary (e.g., confirming or canceling the reservation).

- **Expiring Ingredients:**
  - The system flags ingredients that are close to their expiration dates.
  - This helps in managing the stock effectively and minimizing wastage.
- **Low Stock Ingredients:**
  - tracks inventory items that have low stock levels and alerts the user to restock before running out.
- **Order Management**
  - **Order ID:** 12345
  - **Customer Name:** John Doe
  - **Mobile Number:** 123-456-7890
  - **Order Date:** 2024-11-18
  - **Status:** Completed
  - **Items Ordered:**
- **Item Name:** Veg Pizza
  - **Item ID:** 101
  - **Size:** Medium
  - **Quantity:** 2
  - **Total Price:** ₹500
- **Item Name:** Soft Drink
  - **Item ID:** 102
  - **Size:** Regular
  - **Quantity:** 1
  - **Total Price:** ₹50

- **Inventory Management**

  Item 1:
  - **Item ID:** 101
  - **Item Name:** Veg Pizza
  - **Stock:** 30
  - **Unit:** pcs

Item 2:

- **Item ID:** 102
- **Item Name:** Soft Drink
- **Stock:** 50
- **Unit:** bottles

- **Employee Profile**
    - **Name:** John Doe
    - **Gender:** Male
    - **Date of Birth:** 1990-01-01
    - **Email:** john@example.com
    - **Phone:** 123-456-7890
    - **Role:** Staff
    - **Designation:** Waiter
    - **Joining Date:** 2022-05-10
    - **Salary:** ₹25,000
    - **Status:** Active
- **Address:**
    - **Country:** India
    - **State:** Maharashtra
    - **District:** Mumbai
    - **City/Village:** Mumbai City
    - **Pincode:** 400001

- **Employee List**
    - **Employee 1:**
    - **Name:** John Doe
    - **Role:** Waiter
    - **Salary:** ₹25,000
    - **Status:** Active
    - **Joining Date:** 2022-05-10

- **Employee 2:**
    - **Name:** Jane Smith
    - **Role:** Chef
    - **Salary:** ₹35,000
    - **Status:** Active
    - **Joining Date:** 2021-06-15

Reservations

- **Reservation 1:**
  - **Name:** John Doe
  - **Phone:** 123-456-7890
  - **Booking Date:** 2024-11-18
  - **Time:** 19:00
  - **Table ID:** 5
  - **Status:** Confirmed

- **Reservation 2:**
  - **Name:** Jane Smith
  - **Phone:** 987-654-3210
  - **Booking Date:** 2024-11-18
  - **Time:** 20:00
  - **Table ID:** 3
  - **Status:** Pending

Canceled Reservations

- **Reservation 1:**
  - **Name:** John Doe
  - **Phone:** 123-456-7890
  - **Cancellation Date:** 2024-11-18
  - **Time:** 18:00
  - **Table ID:** 5

Expiring Ingredients

- **Ingredient 1:**
    - **Category:** Dairy
    - **Item Name:** Milk
    - **Expiration Date:** 2024-11-20
- **Ingredient 2:**
    - **Category:** Vegetables
    - **Item Name:** Lettuce
    - **Expiration Date:** 2024-11-22

Low Stock Ingredients

- **Ingredient 1:**
    - **Category:** Dairy
    - **Item Name:** Milk
    - **Quantity:** 5
- **Ingredient 2:**
    - **Category:** Vegetables
    - **Item Name:** Lettuce
    - **Quantity:** 3

ORDER MANAGEMENT MODULE

Function/Method	Description
load_menu()	Loads the menu from the menu_file and returns the first menu item.
load_orders()	Loads existing orders from order_file and returns them as a list. If the file is not found, returns an empty list.
save_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.
load_current_user()	Loads the current logged-in user from users_file and returns the user's data.
take_order()	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.
show_all_orders()	Displays all orders in the system. If no orders exist, an error message is shown.
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.
show_menu()	Displays the entire menu with items grouped by category and their prices and ingredients.
exit_system()	Exits the order management system and shows an exit message.

Function/Method	Description
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()**:
    - 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.
- **show_all_orders()**:
    - Displays details of all orders in the system.
- **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.

REPORTS

1. **BillReport Class**:
   - Loads billing data from a JSON file.
   - Allows the user to filter reports by date (year, month, day, last week).
   - Displays a report based on the filtered date, showing details like billing ID, total, payment type, etc.
2. **ReservationReport Class**:
   - Loads reservation data from a JSON file.
   - Allows the user to filter reservations by date or show canceled reservations.
   - Supports filters like specific dates, months, or "last week".
   - Uses helper methods to filter and display reservations within the given range.
3. **OrderReport Class**:
   - Loads order data from a JSON file.
   - Summarizes the ordered items by their quantity and shows details like the customer name, email, and time of order.
   - Allows the user to filter by date (specific day, month, or year).

You can use these classes to generate and display reports related to bills, reservations, and orders in a structured way, based on user input. Each class leverages helper methods for filtering and formatting the results.

STAFF MANAGEMENT

Section	Code/Component	Explanation
Logger Setup	logger.py	Sets up a centralized logging utility to log

Section	Code/Component	Explanation
		exceptions and events in all modules. Logs are saved to a file (logs/application.log) and console.
Logger Configuration	get_logger(module_name)	Provides a logger instance for a given module, making it easy to log errors and other information in each class or method.
Initialization	SalaryManagement.__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	StaffManagementSystem.authenticate_user	Authenticates a user by checking email and password from users.json. Logs errors during

Section	Code/Component	Explanation
		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.
Logging Use	self.logger.exception("Error message", e)	Logs the exact point of failure and the error message (stack trace) for debugging purposes.

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.

Aspect	How It Works/Improves
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.

UTILITY

1. Utility Classes and Constants

Component	Code/Definition	Explanation
Color Utility	class Colors:	A class defining ANSI escape codes for text colors to enhance terminal output readability and aesthetics.
File Paths	users_file, reservations_file, etc.	File paths for different JSON databases (users, reservations, menu, orders, etc.). Used across modules.
Booking Constants	OPENING_TIME, CLOSING_TIME, HOLIDAYS	Defines restaurant timings (10 AM to 10 PM) and holiday(s). Used for validating table booking schedules.

2. Dashboard Navigation

Function	Code/Definition	Explanation
ask_for_dashboard()	Displays options to return to the dashboard or log out. Returns True for dashboard and False for logout.	Loop until valid input (1 or 2) is entered. Invalid choices prompt the user again.

3. General Validation Functions

Function	Code/Definition	Explanation
validate_name()	Ensures the name is not blank, has no digits or special characters.	Returns an error string or None if valid.

Function	Code/Definition	Explanation
validate_email()	Validates email format (presence of @, ., and their positions).	Ensures valid email structure, e.g., user@example.com.
validate_phone_number()	Checks that the phone number is exactly 10 digits and contains no letters.	Used for validating user contact information.
validate_date_of_birth()	Validates age (between 18 and 60 years) and ensures the date is not in the future.	Accepts date in YYYY-MM-DD format and calculates age.
validate_password()	Ensures a password is at least 6 characters long.	Basic security measure for user accounts.

4. Menu and Order Validations

Function	Code/Definition	Explanation
validate_category()	Checks if the entered category exists in the menu data.	Prevents users from selecting invalid menu categories.
validate_item_id()	Ensures the item_id exists in a specified category within the menu.	Helps in validating orders.
validate_quantity()	Checks if the quantity is a positive integer.	Used during order placement.
validate_price()	Ensures the price is a non-negative integer.	For validating item prices in the menu.

5. Booking Validations

Function	Code/Definition	Explanation
validate_booking_date_time()	Ensures the reservation time is: future, not on holidays, and within operating hours.	Validates booking date-time in YYYY-MM-DD HH:MM AM/PM format.
validate_seats()	Checks if the number of seats is positive and numeric.	Prevents invalid seating reservations.
validate_booking_id()	Ensures the booking ID contains only alphanumeric characters.	Used to validate or identify bookings uniquely.

6. Input Helpers

Function	Code/Definition	Explanation
get_valid_input()	Loops until valid input is received based on a validation function.	Simplifies repetitive input validation.
get_valid_int_input()	Ensures the user enters a valid integer.	Handles invalid input gracefully.
get_valid_float_input()	Ensures the user enters a valid floating-point number.	Used for prices or other numeric inputs.

7. Ingredient and Portion Validation

Function	Code/Definition	Explanation
validate_ingredient_input()	Ensures the input for ingredients is not blank.	Simplifies ingredient processing by splitting and stripping commas.
validate_has_portion_sizes()	Checks if the input is yes or no.	Used for defining whether menu items have portion sizes.

8. Booking and Billing IDs

Function	Code/Definition	Explanation
validate_billing_id()	Ensures billing ID is alphanumeric and not blank.	Validates identifiers for billing records.
validate_order_id()	Ensures order ID is alphanumeric and not blank.	Validates order identifiers for tracking and processing orders.

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.

Code Component	Explanation
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.

Start
MainDashboard
ExitSystem
UserAuthentication
Exit
Login
Signup
ValidateCredentials
ValidateUser
RedirectToDashboard
AssignRole
StaffDashboard
OwnerDashboard
Reports
TableBooking
BillingSystem
MenuManagement
OrderManagement
StaffManagement
ReservationReport
OrderReport
BillingReport
CancelReservation
ViewAvailability
BookTable
MarkAsPaid
ViewPaidBills
GenerateBill
ManageBills
UpdateItem
CheckStock
ViewItem
DeleteItem
AddItem
CancelOrders
UpdateOrders
ManageStaffProfiles
AddUpdateDeleteStaff
TrackSalaries