Contents

[Restaurant Inventory Management System Implementation Documentation 2](#_Toc197713481)

[Overview 2](#_Toc197713482)

[Main Components 2](#_Toc197713483)

[1. Database Initialisation 2](#_Toc197713484)

[2. StockDatabase Class 2](#_Toc197713485)

[3. InventoryController Class 3](#_Toc197713486)

[4. InventoryUI Class 3](#_Toc197713487)

[User Interface Flow 4](#_Toc197713488)

[Error Handling 4](#_Toc197713489)

# Restaurant Inventory Management System Implementation Documentation

# Overview

This document is for the Restaurant Inventory Management System. It's built with **Python**, using **sqlite3** for the database and **tkinter** for the graphical user interface (GUI). The system allows users to add, view, modify, and delete inventory records as well as monitor low stock items. It ist mostly based on the sequence and class diagrams from the design stage

# Main Components

## 1. Database Initialisation

At the top of the code the SQLite database is initialized:

* If the Inventory table does not exist, it is created with columns for item ID, name, quantity, reorder level, and supplier info.
* The database is committed and closed after setup.

## 2. **StockDatabase** Class

Handles all direct interactions with the SQLite database.

#### Key Methods:

* insert\_entry: Adds a new inventory record.
* modify\_entry: Updates an existing record (fields are optional).
* delete\_entry: Removes a record by item ID.
* search\_records: Searches for records by ID and/or name.
* get\_low\_stock\_items: Returns items where quantity is below the reorder level.
* retrieve\_usage\_data: Returns all inventory records.
* close: Closes the database connection.

Error Handling:  
All methods use try-except blocks to catch and raise database errors. Comprehensive error handling (datatypes, empty fields etc) has been done the Inventory UI Class.

## 3. **InventoryController** Class

Acts as a bridge between the UI and the database.

#### Key Methods:

* add\_new\_stock\_entry: Adds a new record via StockDatabase.
* modify\_stock\_entry: Modifies an existing record.
* delete\_stock\_entry: Deletes a record.
* search\_stock\_records: Searches for records.
* monitor\_stock\_records: Gets low stock items.
* generate\_usage\_report: Gets all records.
* close: Closes the database connection.

Error Handling:  
Catches and raises exceptions from the database layer.

## 4. **InventoryUI** Class

Implements the graphical user interface using tkinter. It’s the longest class in terms of code.

#### Initialization:

* Sets up the main window (fixed size, cantered).
* Creates main menu buttons for all actions.

#### Key Methods:

* set\_main\_window: Centres and sizes the main window.
* get\_child\_window\_position: Calculates the position for child windows (to the right of the main menu).
* create\_widgets: Adds buttons for all main actions.
* add\_new\_entry / submit\_new\_entry: Opens a form to add a new item and submits it.
* view\_records: Opens a window to display all records in a formatted, scrollable text area.
* notify\_low\_stock: Opens a window to display low stock items in a formatted text area.
* modify\_entry / submit\_modify\_entry: Opens a form to modify an existing record and submits changes.
* delete\_entry / submit\_delete\_entry: Opens a form to delete a record by ID and submits the deletion.
* close: Closes the controller/database connection.

Error Handling:  
All user actions are wrapped in try-except blocks, with errors shown in message boxes. Note: All possible erros that I could think of has been addressed with error handling.

# User Interface Flow

1. **Main Menu:**
   * Fixed size (400x300), centered on the screen.
   * Buttons: Add, View, Notify Low Stock, Modify, Delete.
2. **Child Windows:**
   * Open to the right of the main menu.
   * Consistent sizing and positioning.
3. **View Records:**
   * Opens a window.
   * Displays all records in a formatted scrollable text area.
4. **Notify Low Stock:**
   * Opens a window.
   * Displays only items below their reorder level, formatted with headers.
5. **Add/Modify/Delete:**
   * Each opens a form window for user input.
   * Validates input and provides feedback.

# Error Handling

* All database and controller operations are wrapped in try-except blocks.
* User input is validated for all possible cases (e.g., required fields, datatypes).
* Errors are displayed to the user via tkinter.messagebox.