System Requirements Specification Index

For

Flower Shop Inventory Management System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

1 PROJECT ABSTRACT

"Bloom & Blossom" is a rapidly expanding flower shop chain with 15 locations across the region. As the business grows, their manual inventory system has become inefficient and error-prone, leading to issues like overordering perishable stock, unexpected shortages during peak seasons, and financial losses from discarded expired flowers. Management needs a robust inventory system that handles the unique challenges of their perishable products while maintaining reliable operation even when faced with unexpected inputs, network issues, or user errors. The Flower Shop Inventory Management System will address these challenges with comprehensive error and exception handling throughout all critical business operations.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. Handle inventory management for various flower types 2. Process customer orders with proper validation 3. Track daily sales and generate reports 4. Implement robust error handling using all four exception handling blocks 5. Maintain data integrity during exceptional conditions

3 CONSTRAINTS

3.1 CLASS AND METHOD REQUIREMENTS

1. `FlowerShopException` Class:

   - Base exception for the entire system
   - Methods
     - `__init__(message, error_code=None)`: Initialize with message and optional error code
     - `__str__()`: Return formatted error message with error code if available
2. Required Exception Subclasses:
   - `InvalidFlowerDataError`: For flower data validation errors
   - `InvalidOrderError(reason)`: For order-related errors with constructor taking reason parameter
   - `OutOfStockError(flower_name, requested, available)`: For insufficient inventory
   - `ExpiredFlowerError(flower_name, expiry_date)`: For expired flowers
3. `Flower` Class
   - Attributes:
     - `name`: String name of the flower
     - `price`: Float price per stem
     - `quantity`: Integer number of stems
     - `freshness_date`: Datetime when flower expires
   - Methods:
     - `__init__(name, price, quantity, freshness_days=7)`: Initialize flower with try-except-else pattern
     - `validate_name(name)`: Validate name format and raise appropriate exception
     - `validate_price(price)`: Validate price is positive number and raise appropriate exception
     - `validate_quantity(quantity)`: Validate quantity is non-negative integer and raise appropriate exception
     - `is_fresh()`: Return True if flower is still fresh, False otherwise
     - `__str__()`: Return string representation of flower

4. `Inventory` Class
   - Attributes:
     - `flowers`: Dictionary mapping flower names to Flower objects
     - `transaction_log`: List of dictionaries containing transaction records

- Methods:
  - `__init__()`: Initialize empty inventory
  - `add_flower(flower)`: Add flower to inventory with full try-except-else-finally pattern
  - `remove_flower(flower_name, quantity)`: Remove flowers from inventory with full try-except-else-finally pattern
  - `check_stock(flower_name)`: Check stock level with try-except-else pattern
  - `get_all_flowers()`: Return list of all flowers in inventory
  - `get_transaction_log()`: Return the transaction log

5. `Order` Class
   - Attributes:
     - `customer_name`: Name of the customer
     - `inventory`: Reference to Inventory object
     - `items`: Dictionary mapping flower names to quantities
     - `status`: String status of order ("new", "processed", or "failed")
     - `total_price`: Float total price of the order
   - Methods:
     - `__init__()`: Initialize empty inventory
     - `__init__(customer_name, inventory)`: Initialize with try-except-else pattern
     - `add_item(flower_name, quantity)`: Add item with full try-except-else-finally pattern
     - `process()`: Process order with full try-except-else-finally pattern
     - `_rollback_inventory(processed_items)`: Roll back inventory changes with try-except-finally pattern
     - `__str__()`: Return string representation of order

3.2 Function Constraints

1. `generate_daily_report(inventory)`:
   - Generate sales and inventory report with full try-except-else-finally pattern
   - Return dictionary with report data
   - Must use all four exception handling components
2. `main()`:

- Demonstrate all exception handling patterns
- Must include examples of all error types
- Must show proper usage of try-except-else-finally blocks

3.3 Error Handling Constraints

1. `try` Block Usage:
   - All operations that could raise exceptions must be in try blocks
   - Code should be structured to isolate potential error sources
   - Required in all methods that modify state or access external resources
2. `except` Block Usage:
   - Must catch specific exceptions, not generic Exception where possible
   - Handle exceptions appropriately with meaningful error messages
   - Re-raise exceptions when appropriate
   - Must include proper error logging
3. `else` Block Usage:
   - Must be used for code that should run only when no exceptions occur
   - Must implement logical separation between error-prone and safe code
   - Use for transaction completion and state updates
   - Required in all methods with try-except blocks
4. `finally` Block Usage:
   - Must be used for cleanup operations that always need to happen
   - Must implement resource management and logging
   - Handle rollback operations when required
   - Required in all methods with state changes or resource use

3.4 Data Constraints

1. Transaction Log Structure:
   - Each entry must be a dictionary with:
     - `type`: String type of transaction (e.g., "add", "remove")
     - `flower_name`: String name of the flower
     - `quantity`: Integer quantity affected

        - `status`: String status ("pending", "completed", "failed")
        - `error`: String error message if failed (optional)

2. Order Structure:
    - Must track customer information
    - Must track all items and quantities
    - Must track order status
    - Must track total price
3. Report Structure:
    - Must include date
    - Must include inventory statistics
    - Must include transaction summary
    - Must include stock alerts for low inventory

3.5 Constructor/Destructor Constraints

1. No bare except blocks allowed
2. All transaction operations must use the complete try-except-else-finally pattern
3. Logical flow control must not rely on exceptions (exceptions for exceptional cases only)
4. All resource cleanup must be in finally blocks
5. All transactions must have rollback capability in case of errors

4. Template code Structure:

1. Exception Classes:

- Base FlowerShopException
- Specialized exception subclasses

2. Flower Class:

- Attributes for flower properties
- Validation methods
- Error handling with try-except-else

3. Inventory Classes:

- Flower storage
- Transaction log

- Operations with full exception handling

4. Order Class:

- Customer information
- Order items
- Processing with transaction integrity

5. Utility Functions:

- Report generation
- Helper functions

4. Main Program:

- Demonstration of all exception handling patterns

5. Execution Steps to follow:

1. Create exception hierarchy with all required exception classes
2. Implement Flower class with validation using try-except-else
3. Develop Inventory class with full try-except-else-finally blocks
4. Build Order class with transaction handling and rollback
5. Implement utility functions with complete error handling
6. Create main function demonstrating all exception handling patterns