System Requirements Specification Index

For

Street Food Vendor Management System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

Street Food Vendor Management System

System Requirements Specification

1 Project Abstract

A street food vendor needs a comprehensive system to track inventory, record sales, store customer feedback, and manage daily operations using various file operations. This system will enable vendors to efficiently manage their business by monitoring stock levels, analyzing sales trends, reviewing customer satisfaction, and generating operational reports.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. Record and update food inventory in text files 2. Log sales transactions in CSV format 3. Save and search customer feedback in text files 4. Generate reports from sales history 5. Create backups of critical data files

3 Constraints

3.1 File Format Requirements

1. Inventory File (`inventory.txt`):
   - Format: `item_name,quantity,price`
   - Example: `samosa,50,2.50`
   - One item per line

- Headers optional but recommended
- Numeric values must be validated

2. Sales Log File (`sales.csv`):
   - Format: `timestamp,item_name,quantity,total_price`
   - Example: `2023-01-01 12:30:45,samosa,5,12.50`
   - Must include header row
   - Timestamp must follow YYYY-MM-DD HH:MM:SS format
   - Quantity must be positive integer
   - Price must be positive number with two decimal places
3. Feedback File (`feedback.txt`):
   - Format: `item_name,quantity,price`
   - Format: Structured text blocks
   - Must include customer name, rating (1-5), and comments
   - Separate feedback entries with delimiters
   - Example:
     - ===== FEEDBACK: 2023-01-01 12:30:45 =====
     - Customer: John Doe
     - Rating: 4/5
     - Comments: Great food and quick service!
4. Daily Report File (`daily_report.txt`):
   - Format: Structured text report
   - Must include date, sales summary, inventory status
   - Must format currency values with $ and two decimal places
   - Must include section headings and separator lines

3.2 Data Validation Requirements

1. Inventory Validation:
   - Item names must be non-empty strings
   - Quantities must be non-negative integers

- Prices must be positive numbers

2. Sales Validation:
   - Items must exist in inventory before sale
   - Quantities must be positive integers
   - Total price must match quantity * unit price
   - Timestamp must be in correct format
3. Feedback Validation:
   - Customer name must be non-empty
   - Rating must be integer between 1 and 5
   - Comments are optional but must be stored if provided

4. TEMPLATE CODE STRUCTURE:

1. Basic Functions:

- `read_inventory(file_path)` - reads inventory from text file
  - - Returns dictionary mapping item names to quantities and prices
  - - Must handle file not found and invalid format errors
  - - Example: `{"samosa": {"quantity": 50, "price": 2.50}}`
- `update_inventory(file_path, item_name, quantity, price)` - updates inventory
  - Updates existing item or adds new item
  - Writes complete inventory back to file
  - Must validate all input parameters
- `log_sale(file_path, item_name, quantity, total_price)` - records sales
  - Appends sale record to CSV file
  - Creates file with header if it doesn't exist
  - Must add timestamp automatically

2. Advanced Functions:

- `save_customer_feedback(file_path, customer_name, rating, comments)` - saves feedback
  - Appends formatted feedback to text file
  - Must validate rating range (1-5)
  - Must include current timestamp
- `read_sales_report(file_path)` - generates sales summary
  - Reads CSV sales data and calculates statistics
  - Returns dictionary with total revenue, items sold, etc.
  - Must handle empty or malformed CSV files
- `generate_daily_report(inventory_file, sales_file, report_file, date)` - creates reports

- Combines inventory and sales data for specified date
- Creates formatted text report
- Must validate date format

3. Search Functions:

- `search_feedback(file_path, search_term)` - searches feedback for keywords
  - Returns list of feedback entries containing search term
  - Must be case-insensitive
  - Must handle empty files and no matches

4. Utility Functions:

- `backup_data_files(source_dir, backup_dir)` - creates backups of all data files
  - Copies all .txt and .csv files to backup directory
  - Adds timestamp to filenames
  - Returns count of files backed up

5. Main Functions:

- `main()` - creates a menu-driven interface for all operations
  - Displays interactive menu
  - Handles user input
  - Calls appropriate functions based on selection
  - Displays results to user

5. EXECUTION STEPS TO FOLLOW:

1. Implement each function using the appropriate file mode:
   - Use 'r' mode for reading existing files
   - Use 'w' mode for creating new files or overwriting
   - Use 'a' mode for appending to logs and feedback

2. Create a hierarchical directory structure:
   - Main data directory for current files
   - Backup directory for archived files
   - Report directory for generated reports

3. Implement proper error handling for all file operations:
   - Check if files exist before reading
   - Create files if they don't exist when writing
   - Validate data before processing

4. Create a user-friendly menu system:
   - Clear options with numbering
   - Input validation
   - Meaningful feedback messages

- Confirmation for critical operations

5. Test each function individually:
    - Test with valid inputs
    - Test with invalid inputs
    - Test with boundary cases
    - Test with empty files