System Requirements Specification Index

For

Coffee Processing Plant Management System

Version 1.0

IIHT Pvt. Ltd.

fullstack@iiht.com

TABLE OF CONTENTS

Coffee Processing Plant Management System

System Requirements Specification

1 PROJECT ABSTRACT

Highland Coffee Cooperative needs a simple system to track coffee bean inventory and processing stages. This assignment focuses on implementing basic file handling operations to read, write, and analyze coffee processing data stored in text files.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. System must track coffee bean batches from farmers 2. Tool must record basic processing stages (washing, drying, roasting) 3. System must generate simple reports on inventory and processing status

3 CONSTRAINTS

3.1 FILE STRUCTURE

1. File Structure:
   - Inventory data stored in "bean_inventory.txt"
   - Processing records stored in "processing_records.txt"
   - Operations log in "operations_log.txt"
2. File Formats:
   - Bean inventory records: "batch_id,date,farmer_id,bean_type,weight_kg,status"
     - § Example: "B001,2023-05-15,F042,Arabica,250,received"
   - Processing records: "batch_id,process_type,start_date,end_date,weight_after"
     - § Example: "B001,washing,2023-05-16,2023-05-17,245"

4. TEMPLATE CODE STRUCTURE:

1. File Reading Functions:

- `read_inventory(file_path)` - reads all bean batch records
- `read_processing_records(file_path)` - reads all processing stage records
- `find_batch_by_id(batch_id, file_path)` - locates a specific batch

2. File Writing Functions:

- `add_bean_batch(batch_data, file_path)` - adds a new batch to inventory
- `record_processing_stage(processing_data, file_path)` - adds a processing record
- `update_batch_status(batch_id, new_status, file_path)` - updates a batch status

3. Analysis Functions:
    - `calculate_inventory_summary(file_path)` - summarizes current inventory
    - `calculate_processing_yields(inventory_path, processing_path)` - analyzes processing yields
4. Helper Functions:
    - `log_operation(operation, details, log_file_path)` - logs system operations
    - `create_sample_data()` - creates sample data for testing
5. Main Program Function:
    - `main()` - demonstrates all functionality with a command-line interface

5. Detailed Function Structure:

5.1 File Reading Functions

1. Write a Python function to read all coffee bean batch records from inventory file.

Define: **read_inventory(file_path="bean_inventory.txt")**

This function should:

- Accept optional file path parameter with default "bean_inventory.txt"
- Use try-except block to handle FileNotFoundError
- Open file using `with open(file_path, "r") as f:` statement
- Read file line by line using `for line in f:`
- Skip empty lines using `if line.strip():`
- Split each line by comma: `parts = line.strip().split(",")`
- Validate line has at least 6 parts before processing

- Create dictionary for each valid line with keys: "batch_id", "date", "farmer_id", "bean_type", "weight_kg", "status"
- Convert weight_kg to float: "weight_kg": float(parts[4])
- Return list of batch dictionaries
- Return empty list [] if file not found or on error
- Print appropriate error messages for debugging

2. Write a Python function to read all processing stage records from processing file.

Define: **read_processing_records(file_path="processing_records.txt")**

This function should:

- Accept optional file path parameter with default "processing_records.txt"
- Use try-except block to handle FileNotFoundError
- Open file using with open(file_path, "r") as f: statement
- Read file line by line and skip empty lines
- Split each line by comma and validate it has at least 5 parts
- Create dictionary for each valid line with keys: "batch_id", "process_type", "start_date", "end_date", "weight_after"
- Convert weight_after to float: "weight_after": float(parts[4])
- Return list of processing record dictionaries
- Return empty list [] if file not found or on error
- Handle ValueError for invalid numeric conversions

3. Write a Python function to locate a specific batch record in the inventory.

Define: **find_batch_by_id(batch_id, file_path="bean_inventory.txt")**

This function should:

- Accept batch_id string and optional file path parameter
- Call `read_inventory(file_path)` to get all inventory records
- Iterate through inventory list using `for batch in inventory:`
- Compare each batch's "batch_id" with the target batch_id

- Return the matching batch dictionary if found
- Return `None` if batch not found
- Handle exceptions and return `None` on error
- Use exact string matching for batch_id comparison

5.2 File Writing Functions

4. Write a Python function to add a new bean batch to the inventory file.

Define: **add_bean_batch(batch_data, file_path="bean_inventory.txt")**

This function should:

- Accept batch_data dictionary and optional file path parameter
- Validate required fields are present: ["batch_id", "date", "farmer_id", "bean_type", "weight_kg", "status"]
- Use loop to check each required field: for field in required_fields:
- Return False and print error if any field missing
- Check if batch_id already exists using find_batch_by_id()
- Return False if duplicate batch_id found
- Format data as CSV line: f"{batch_data['batch_id']},{batch_data['date']},{batch_data['farmer_id']},{batch_data['bean_type']},{batch_data['weight_kg']},{batch_data['status']}\n"
- Open file in append mode: with open(file_path, "a") as f:
- Write formatted line to file: f.write(line)
- Call log_operation("add_batch", f"Added batch {batch_data['batch_id']}") to log the action
- Return True on success, False on any error

5. Write a Python function to add a new processing record to the processing records file.

Define:**record_processing_stage(processing_data,file_path="processing_records.txt")**

This function should:

- Accept processing_data dictionary and optional file path parameter
- Validate required fields: ["batch_id", "process_type", "start_date", "end_date", "weight_after"]
- Return `False` and print error message if any field missing
- Format data as CSV line with all fields
- Open file in append mode and write the formatted line
- After successful write, call `update_batch_status(processing_data['batch_id'], processing_data['process_type'])`
- Call `log_operation("record_processing", details)` to log the action
- Return `True` on success, `False` on error
- Handle exceptions with try-except block

6. Write a Python function to update the status of a batch in the inventory file.

Define:**update_batch_status(batch_id,new_status,file_path="bean_inventory.txt")**

This function should:

- Accept batch_id string, new_status string, and optional file path
- Call `read_inventory(file_path)` to get all current records
- Iterate through inventory to find matching batch_id
- Update the status field: `batch["status"] = new_status`
- Set batch_found flag to True when batch is located
- Return `False` if batch not found
- Rewrite entire file with updated data using `with open(file_path, "w") as f:`
- Format each batch as CSV line and write to file
- Call `log_operation("update_status", details)` to log the action
- Return `True` on success, `False` on error

5.3 Analysis Functions

7. Write a Python function to generate a summary of the current inventory.

Define: **calculate_inventory_summary(file_path="bean_inventory.txt")**

This function should:

- Accept optional file path parameter
- Call `read_inventory(file_path)` to get all inventory records
- Return `None` if inventory is empty
- Calculate total_batches using `len(inventory)`
- Calculate total_weight using `sum(batch["weight_kg"] for batch in inventory)`
- Create bean_types dictionary to group by bean type
- Iterate through inventory and accumulate weight by bean_type
- Create stages dictionary to group by processing status
- Iterate through inventory and accumulate weight by status
- Return dictionary with keys: "total_batches", "total_weight", "bean_types", "stages"
- Handle exceptions and return `None` on error
- Example return: `{"total_batches": 5, "total_weight": 1200.0, "bean_types": {"Arabica": 800.0, "Robusta": 400.0}, "stages": {"received": 300.0, "washing": 500.0, "drying": 400.0}}`

8. Write a Python function to calculate yield percentages through different processing stages.

Define: **calculate_processing_yields(inventory_path="bean_inventory.txt", processing_path="processing_records.txt")**

This function should:

- Accept optional inventory and processing file paths
- Call `read_processing_records(processing_path)` to get all processing records
- Group records by process_type using dictionary
- For each process type, calculate yield statistics
- For each processing record, find original weight using `find_batch_by_id()`

- Calculate yield percentage: `(weight_after / original_weight) * 100`
- Calculate average yield percentage for each process type
- Count number of batches processed for each type
- Return dictionary with process types as keys
- Each process type should have: "average_yield_percentage" and "count"
- Handle division by zero and missing batch errors
- Return empty dictionary `{}` on error

5.4 Logging and Utility Functions

9. Write a Python function to log operations to the log file with timestamp.

Define: **log_operation(operation, details, log_file_path="operations_log.txt")**

This function should:

- Accept operation string, details string, and optional log file path
- Import datetime: `from datetime import datetime`
- Get current timestamp: `timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")`
- Format log entry: `f"{timestamp},{operation},{details}\n"`
- Open log file in append mode: `with open(log_file_path, "a") as f:`
- Write log entry to file: `f.write(log_entry)`
- Return `True` on success, `False` on error
- Handle exceptions with try-except block
- Example log entry: "2023-05-20 14:30:25,add_batch,Added batch B001"

10. Write a Python function to retrieve the most recent log entries.

Define: **read_recent_logs(count=5, log_file_path="operations_log.txt")**

This function should:

- Accept count parameter (default 5) and optional log file path
- Use try-except to handle FileNotFoundError
- Open file and read all lines: `lines = f.readlines()`
- Get last 'count' lines: `lines[-count:]`

- Parse each line by splitting on comma with limit: `parts = line.strip().split(",", 2)`
- Create dictionary for each log entry with keys: "timestamp", "operation", "details"
- Reverse the list to get newest entries first: `recent_logs.reverse()`
- Return list of log dictionaries
- Return empty list `[]` if file not found or on error
- Handle malformed log entries gracefully

11. Write a Python function to create sample data files for demonstration.

Define: **create_sample_data()**

This function should:

- Create "bean_inventory.txt" with at least 3 sample batch records
- Include different bean types (Arabica, Robusta) and statuses
- Sample format: "B001,2023-05-15,F042,Arabica,250,received"
- Create "processing_records.txt" with corresponding processing records
- Include different process types (washing, drying)
- Sample format: "B001,washing,2023-05-16,2023-05-17,245"
- Use try-except block to handle file creation errors
- Use `with open(filename, "w") as f:` for each file
- Write multiple sample lines to each file
- Return `True` if both files created successfully
- Return `False` if any file creation fails
- Print appropriate success/error messages

5.5 Main Program Function

12. Write a Python function to display the inventory summary in a readable format.

Define: **display_inventory_summary(summary)**

This function should:

- Accept summary dictionary parameter
- Check if summary is None or empty and print appropriate message
- Print formatted header: "=== INVENTORY SUMMARY ==="
- Display total batches and total weight with proper formatting
- Print "By Bean Type:" section with weight and percentage for each type
- Calculate percentage: `(weight / summary['total_weight']) * 100`
- Print "By Processing Stage:" section with weight and percentage for each stage
- Use appropriate number formatting (e.g., 1 decimal place)
- Handle division by zero when calculating percentages
- Example output format: "Arabica: 800.0 kg (66.7%)"

13. Write a Python function to demonstrate the complete coffee processing system.

Define: **main()**

This function should:

- Print system header: "===== COFFEE PROCESSING SYSTEM ====="
- Prompt user to create sample data with input prompt
- Implement while loop for menu-driven interface
- Display menu options:
  - View inventory summary
  - Add new batch
  - Record processing stage
  - View processing yields
  - View recent logs
  - Exit
- Handle each menu choice with appropriate function calls
  For option 1:`Call calculate_inventory_summary()` and `display_inventory_summary()`
- For option 2: Prompt for batch details and call `add_bean_batch()`
- For option 3: Prompt for processing details and call `record_processing_stage()`
- For option 4: Call `calculate_processing_yields()` and display formatted results
- For option 5: Call `read_recent_logs()` and display formatted log entries
- Include input validation for numeric inputs (weight values)
- Handle invalid menu choices gracefully

- Exit gracefully when user chooses option 0

6. EXECUTION STEPS TO FOLLOW:

1. Implement the required file handling functions according to specifications

2. Create sample coffee batch and processing records for testing

3. Test each function with basic error handling

4. Create a simple command-line interface to demonstrate the functionality

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through Command Terminal.

- To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal

- This editor Auto Saves the code

- If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page)

- These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

- To launch application: **python3 filename.py**

- To run Test cases: **python3 -m unittest**

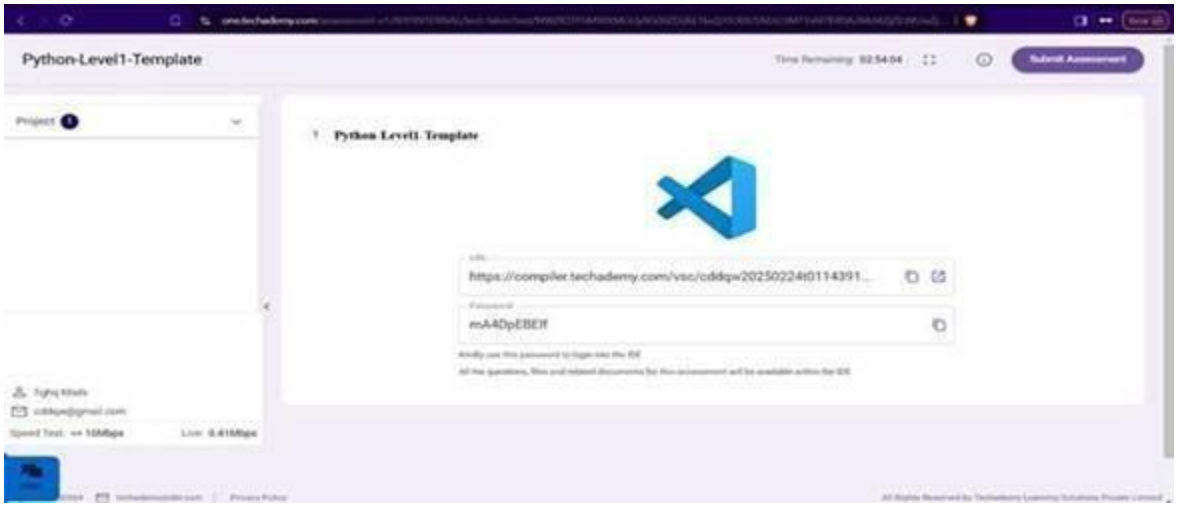
Screen shot to run the program

To run the application

python3 filename.py

To run the testcase

python3 -m unittest



- Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.