**System Requirements Specification Index**

**For**

**Hydroponic Farm Monitoring System**

**Version 1.0**

**IIHT Pvt. Ltd.**

**fullstack@iiht.com**

**Table of Contents**

1 Project Abstract

2 Business Requirements

3 Constraints

4 Template Code Structure

5 Execution Steps to Follow

**Hydroponic Farm Monitoring System**

**System Requirements Specification**

**1** **Project Abstract**

A small hydroponic farm in the North-East region of India called Karangi Farms needs a system to track plant growth, nutrient levels, and environmental conditions. They require a simplistic interface for logging information for various stages of hydroponic farming. They require separate logs that need to be stored and be accessible in an easy format. Create a python console application that logs information that is needed in simple files utilizing the file handling methodologies commonly used with python.

**2** **Business Requirements:**

| Screen Name | Console input screen |
| --- | --- |
| Problem Statement | 1. Record daily sensor readings in text files  2. Log system activities and alerts  3. Generate reports from historical data  4. Store and retrieve nutrient mixing recipes |

**3** **Constraints**

**3.1** **File Requirements**

1. Directory Structure:

2. `sensor\_readings.txt`: Daily sensor data

3. `system\_log.txt`: Operation logs (append-only)

4. `nutrient\_levels.csv`: Nutrient measurements

5. `recipes.txt`: Nutrient mixing recipes

**3.2** **File Mode Requirements**

1. Read ('r'): For generating reports

2. Write ('w'): For creating new data files

3. Append ('a'): For adding to logs without overwriting

4. Read/Write ('r+'): For updating recipes

**4. Template code Structure:**

**1.** Basic Functions:

o `read\_sensor\_data(file\_path)` - reads sensor history ('r' mode)

o `save\_daily\_readings(file\_path, data)` - records new readings ('w' mode)

o `log\_system\_event(file\_path, message)` - logs events ('a' mode)

**2.** Advanced Functions:

o `update\_recipe(file\_path, recipe\_name, new\_instructions)` - updates recipes ('r+' mode)

o `backup\_data\_files(source\_dir, backup\_dir)` - creates data backups

**3.** Utility Functions:

o `generate\_weekly\_report(data\_file\_path, output\_file\_path)` - creates reports

o `search\_logs(log\_file\_path, search\_term)` - searches logs for specific events

**4.** Main Program Function:

o `main()` - demonstrates all functions and produces formatted output.

# **5. DETAILED FUNCTION IMPLEMENTATION GUIDE**

## **5.1 Basic File Reading Functions**

1. **Write a Python function to read sensor data from text files.** Define: read\_sensor\_data(file\_path="sensor\_readings.txt") The function should:  
   * Accept optional file path parameter with default "sensor\_readings.txt"
   * Use open(file\_path, "r") mode to read the file
   * Handle FileNotFoundError gracefully by returning empty list
   * Parse each line in format: "date,temperature,humidity,ph\_level,light\_level"
   * Split each line by comma and convert numeric values to float
   * Skip empty lines and malformed data lines
   * Return list of dictionaries with keys: "date", "temperature", "humidity", "ph\_level", "light\_level"
   * Handle exceptions and print error messages
   * Use proper file context management with with statement
   * Example: Parse line "2023-06-01,24.5,65.2,6.2,22000" into dictionary
   * Return empty list if file doesn't exist or has errors
2. **Write a Python function to read nutrient level data from CSV files.** Define: read\_nutrient\_levels(file\_path="nutrient\_levels.csv") The function should:  
   * Accept optional file path parameter with default "nutrient\_levels.csv"
   * Use open(file\_path, "r") mode to read the CSV file
   * Skip the header line: "date,nitrogen,phosphorus,potassium,ec\_level"
   * Parse data lines with comma separation
   * Convert numeric values to appropriate types (float for nutrient levels)
   * Handle malformed CSV data gracefully
   * Return list of dictionaries with keys: "date", "nitrogen", "phosphorus", "potassium", "ec\_level"
   * Handle FileNotFoundError by returning empty list
   * Skip empty lines and invalid data rows
   * Use proper exception handling with descriptive error messages
   * Example: Parse "2023-06-01,180,45,210,1.8" into nutrient dictionary

## **5.2 Basic File Writing Functions**

1. **Write a Python function to save daily sensor readings to text files.** Define: save\_daily\_readings(data, file\_path="sensor\_readings.txt") The function should:  
   * Accept list of dictionaries containing sensor readings
   * Accept optional file path parameter with default "sensor\_readings.txt"
   * Use open(file\_path, "w") mode to overwrite existing file
   * Format each reading as comma-separated line: "date,temperature,humidity,ph\_level,light\_level"
   * Write each reading on a separate line with newline character
   * Call log\_system\_event() to record the save operation
   * Return True on successful save, False on error
   * Handle exceptions gracefully and print error messages
   * Validate that data parameter is a list
   * Use proper file context management
   * Example format: "2023-06-01,24.5,65.2,6.2,22000\n"
   * Ensure all numeric values are properly formatted
2. **Write a Python function to log system events using append mode.** Define: log\_system\_event(event\_type, message, file\_path="system\_log.txt") The function should:  
   * Accept event type string and message string parameters
   * Accept optional file path parameter with default "system\_log.txt"
   * Use open(file\_path, "a") mode to append to existing log file
   * Import datetime module to generate timestamps
   * Format log entry: "timestamp,event\_type,message\n"
   * Use timestamp format: "%Y-%m-%d %H:%M:%S"
   * Append log entry without overwriting existing logs
   * Return True on successful logging, False on error
   * Handle exceptions and print error messages
   * Create file if it doesn't exist (append mode behavior)
   * Example entry: "2023-06-01 14:30:00,Data saved,Saved 5 readings to sensor\_readings.txt\n"

## **5.3 Advanced File Operations**

1. **Write a Python function to append nutrient readings to CSV files.** Define: append\_nutrient\_reading(reading, file\_path="nutrient\_levels.csv") The function should:  
   * Accept dictionary containing nutrient reading data
   * Accept optional file path parameter with default "nutrient\_levels.csv"
   * Check if file exists using try/except block
   * Use open(file\_path, "a") mode to append to CSV file
   * Write CSV header if file is new: "date,nitrogen,phosphorus,potassium,ec\_level\n"
   * Format reading as CSV line: "date,nitrogen,phosphorus,potassium,ec\_level\n"
   * Append reading without overwriting existing data
   * Call log\_system\_event() to record the append operation
   * Return True on successful append, False on error
   * Handle missing dictionary keys gracefully
   * Validate reading parameter is a dictionary
   * Example line: "2023-06-01,180,45,210,1.8\n"
2. **Write a Python function to update recipes using read/write mode.** Define: update\_recipe(recipe\_name, new\_instructions, file\_path="recipes.txt") The function should:  
   * Accept recipe name string and new instructions string
   * Accept optional file path parameter with default "recipes.txt"
   * Use open(file\_path, "r") first to read all existing recipes
   * Parse recipe blocks separated by double newlines
   * Find recipe starting with "Recipe: {recipe\_name}"
   * Replace instructions while keeping recipe header
   * Use open(file\_path, "w") to write back all recipes
   * Handle FileNotFoundError by returning False
   * Return True if recipe found and updated, False if not found
   * Call log\_system\_event() to record the update operation
   * Preserve other recipes unchanged
   * Handle exceptions with appropriate error messages
   * Example: Update "Recipe: Leafy Greens" with new nutrient instructions

## **5.4 Report Generation Functions**

1. **Write a Python function to generate weekly reports from sensor data.** Define: generate\_weekly\_report(data\_file\_path, output\_file\_path="weekly\_report.txt") The function should:
   * Accept data file path as required parameter
   * Accept optional output file path with default "weekly\_report.txt"
   * Call read\_sensor\_data(data\_file\_path) to get readings
   * Return False if no sensor data is available
   * Calculate average values for temperature, humidity, pH, and light level
   * Use mathematical operations: sum() and len() for averages
   * Determine date range from first and last readings
   * Use open(output\_file\_path, "w") to create report file
   * Format report with clear sections and headers
   * Include report title: "WEEKLY HYDROPONIC MONITORING REPORT"
   * Write period information and reading count
   * Format averages section with appropriate decimal places
   * Include all daily readings in detailed format
   * Call log\_system\_event() to record report generation
   * Return True on successful generation, False on error
   * Handle division by zero for empty datasets

## **5.5 Search and Utility Functions**

1. **Write a Python function to search log files for specific terms.** Define: search\_logs(search\_term, file\_path="system\_log.txt") The function should:  
   * Accept search term string as required parameter
   * Accept optional file path parameter with default "system\_log.txt"
   * Use open(file\_path, "r") mode to read log file
   * Search for lines containing search term (case-insensitive)
   * Parse matching lines into structured dictionaries
   * Split log lines by comma with limit: split(",", 2)
   * Return list of dictionaries with keys: "timestamp", "event\_type", "message"
   * Handle FileNotFoundError by returning empty list
   * Use .lower() method for case-insensitive search
   * Handle malformed log entries gracefully
   * Return empty list if no matches found
   * Handle exceptions and print error messages
   * Example: Search for "saved" returns all save-related log entries
2. **Write a Python function to backup data files.** Define: backup\_data\_files(source\_path, backup\_path) The function should:  
   * Accept source file path and backup file path as required parameters
   * Use open(source\_path, "r") to read source file content
   * Use open(backup\_path, "w") to write backup file
   * Copy entire file content from source to backup
   * Handle FileNotFoundError for source file by returning False
   * Return True on successful backup creation
   * Call log\_system\_event() to record backup operation
   * Handle exceptions during read or write operations
   * Preserve exact file content including formatting
   * Create backup directory structure if needed
   * Example: Backup "sensor\_readings.txt" to "backup/sensor\_readings\_20230601.txt"

## **5.6 Sample Data Creation Functions**

1. **Write a Python function to create sample data files for testing.** Define: create\_sample\_data() The function should:
   * Create "sensor\_readings.txt" with sample sensor data
   * Write multiple lines of realistic sensor readings
   * Include varied temperature (20-30°C), humidity (60-80%), pH (5.5-7.0), light (20000-25000 lux)
   * Create "nutrient\_levels.csv" with CSV header and sample data
   * Include realistic nutrient values: nitrogen (150-200 ppm), phosphorus (40-60 ppm), potassium (180-220 ppm), EC (1.5-2.5)
   * Create "recipes.txt" with formatted recipe entries
   * Include at least two recipes: "Leafy Greens" and "Tomatoes"
   * Format recipes with proper headers and nutrient specifications
   * Use open(file, "w") mode for each file creation
   * Call log\_system\_event() to record sample data creation
   * Return True on successful creation, False on error
   * Handle exceptions during file creation
   * Create realistic date sequences (e.g., 2023-06-01, 2023-06-02, 2023-06-03)

## **5.7 Main Program Function**

1. **Write a Python function to demonstrate the hydroponic monitoring system.** Define: main() The function should:
   * Print system title: "===== HYDROPONIC FARM MONITORING SYSTEM ====="
   * Implement menu-driven interface with numbered options
   * Option 1: View sensor readings - call read\_sensor\_data() and display formatted output
   * Option 2: Add new sensor reading - prompt for input and update sensor file
   * Option 3: View nutrient levels - call read\_nutrient\_levels() and display data
   * Option 4: Add new nutrient reading - prompt for input and append to CSV
   * Option 5: Update recipe - prompt for recipe name and new instructions
   * Option 6: Generate weekly report - prompt for file paths and create report
   * Option 7: Search system logs - prompt for search term and display results
   * Option 8: Create backup - prompt for source and backup paths
   * Option 0: Exit program
   * Use input() for menu selection and user data entry
   * Implement input validation for numeric values using try/except
   * Display success/failure messages for each operation
   * Use continuous loop until user chooses to exit
   * Handle invalid menu selections gracefully
   * Format output clearly with section headers and proper spacing

## **5.8 Error Handling and Validation Requirements**

All functions must implement comprehensive error handling:

**File Operation Errors:**

* Handle FileNotFoundError for reading operations
* Handle PermissionError for write operations
* Handle general Exception for unexpected errors
* Print descriptive error messages to console

**Data Validation:**

* Validate input parameter types before processing
* Check for None values in required parameters
* Validate dictionary structure for reading data
* Handle malformed data lines gracefully

**Return Value Consistency:**

* Reading functions: return empty list on error
* Writing functions: return boolean success/failure status
* Search functions: return empty list if no results
* Maintain consistent return types across all functions

**File Format Validation:**

* Ensure CSV files have proper headers
* Validate sensor data line format (5 comma-separated values)
* Handle incomplete or corrupted data files
* Skip invalid lines and continue processing

**User Input Validation:**

* Validate numeric inputs using try/except with ValueError
* Check for empty string inputs where appropriate
* Provide user-friendly error messages for invalid input
* Allow users to retry input on validation failures

**Resource Management:**

* Always use with statement for file operations
* Ensure files are properly closed even if exceptions occur
* Handle cases where files are locked by other processes
* Implement proper cleanup for temporary files

**6. Execution Steps to follow:**

1. Implement each function using the appropriate file mode

2. Create realistic sample data for testing

3. Demonstrate error handling for common file issues

4. Develop a simple menu-driven interface

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) then you need to use CTRL+Shift+B -command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.

● 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

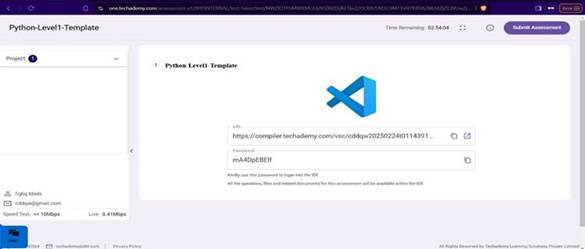
● Before Final Submission also, you need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

Screen shot to run the program

To run the application

Python3 filename.py

To run the testcase python -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.