# Symptom_and_Medication_Journal-PROJECT REPORT

**Submitted by: -** PALAK BHATLA

**Registration no.: -** 25BCY10014

**Course: -** FUNDAMENTAL AI and ML

**Platform: -** VITyarthi - Build Your Own Project

## 1.Introduction: -

Symptom & Medication Journal is a digital health tracking utility created to replace manual, paper-based medical logs. The application, powered with Python for logic and data handling, will enable the recording of medication intake and symptom occurrences by the user with exact timestamps. The project aims to offer a reliable, automated way of tracking personal health history, which better serves communication between a patient and healthcare provider.

## 2. Problem Statement: -

Management of chronic conditions, such as migraines, arthritis, or allergies, depends on human memory, which is subjective and prone to errors. Also, patients often cannot answer questions that are specific and clinically relevant for a doctor, like "How many times did you take this painkiller last month?" or "How much time elapsed from the beginning of the symptom until the medicine took effect?". In the absence of accurate longitudinal data, treatment efficacy is difficult to measure.

## 3. Functional Requirements: -

**Core capabilities provided by the system include:**

- **Item Management:** Users can add new medications or symptoms to their tracking list with category classification of Medication/Symptom.
- **Event Logging:** Users can log a particular occurrence of an item. The current system timestamp is automatically captured by the system.
- **Data Persistence:** The system writes all its logs to a permanent storage file - CSV. This ensures that in case the program closes, data is not lost.
- **Summary Generation:** The user can see a statistical summary that depicts the total count and last recorded timestamp for each item.

- **Input Validation:** The system handles the small caps problems, such as treating "Advil" and "advil" as the same, and numeric input validations.

## 4.Non-Functional Requirements: -

- **Usability:** The CLI was designed with clear prompts and numbered menus in mind, to be accessible for non-technical users.
- **Reliability:** The system avoids crashes on account of invalid user inputs, such as entering letters when the system expects numbers, by using try-except blocks.
- **Data Integrity:** The system generates timestamps automatically, which cannot be falsified or entered incorrectly by a user.
- **Maintainability:** The code is modularized into separate functions, add_item and log_occurrence, which are logical and self-descriptive for future updates.

## 5. System Architecture: -

**The application follows a modular, procedural architecture:**

- **Presentation Layer:** A CLI menu loop that handles user input and prints formatted text.
- **Logic Layer:** Python functions that handle business logic (incrementing counts, fetching timestamps, validating categories).
- **Data Layer:** A list of dictionaries (runtime memory) backed by a CSV file (persistent storage).

## 6.Design Decisions & Rationale: -

- **Language Choice (Python):** Chosen due to readability and solid support for data manipulation libraries (datetime, possibly pandas in the future).
- **Command Line Interface - CLI:** Chosen for the MVP in order to focus on the correctness of the logic and data structure without the overhead of GUI development.
- **Data StorageList/CSV:** A lightweight solution was preferred over a full SQL database to keep the application portable and easy to run without external dependencies.

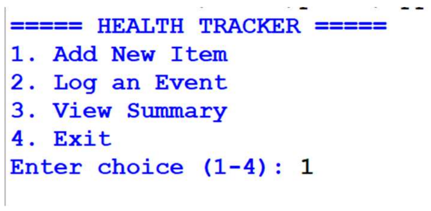
## 7. Implementation Details: -

The core logic relies on the datetime library for temporal tracking.

- **Input Sanitization:** The .strip().title() method chain is used on all string inputs to ensure data consistency.
- **Error Handling:** A while True loop wraps the main menu to ensure the program keeps running. try-except ValueError blocks are implemented in input sections to catch non-numeric entries.

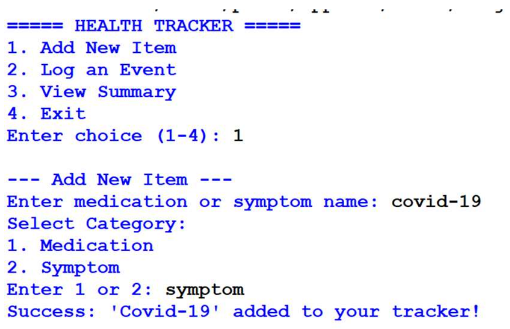
## 8.Screenshots / Results: -

### 1.Menu Drive



```
===== HEALTH TRACKER =====
1. Add New Item
2. Log an Event
3. View Summary
4. Exit
Enter choice (1-4): 1
```

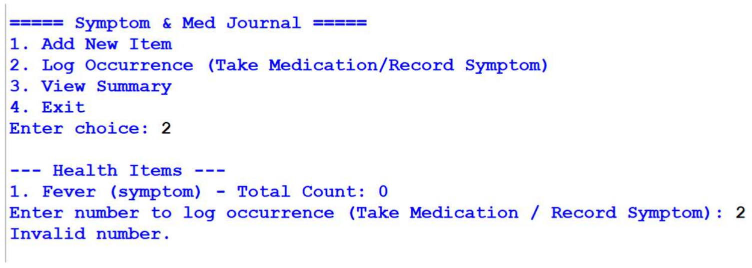
### 2.Entering choice as 1



```
===== HEALTH TRACKER =====
1. Add New Item
2. Log an Event
3. View Summary
4. Exit
Enter choice (1-4): 1

--- Add New Item ---
Enter medication or symptom name: covid-19
Select Category:
1. Medication
2. Symptom
Enter 1 or 2: symptom
Success: 'Covid-19' added to your tracker!
```

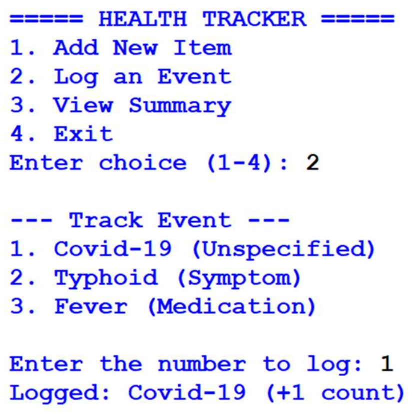
### 3.Entering choice as 2 and checking for error by entering wrong input



```
===== Symptom & Med Journal =====
1. Add New Item
2. Log Occurrence (Take Medication/Record Symptom)
3. View Summary
4. Exit
Enter choice: 2

--- Health Items ---
1. Fever (symptom) - Total Count: 0
Enter number to log occurrence (Take Medication / Record Symptom): 2
Invalid number.
```

### 4.Entering choice as 2 and checking result by entering correct input



```
===== HEALTH TRACKER =====
1. Add New Item
2. Log an Event
3. View Summary
4. Exit
Enter choice (1-4): 2

--- Track Event ---
1. Covid-19 (Unspecified)
2. Typhoid (Symptom)
3. Fever (Medication)

Enter the number to log: 1
Logged: Covid-19 (+1 count)
```

### 5.Entering choice as 3 and checking for the summary

```
===== HEALTH TRACKER =====
1. Add New Item
2. Log an Event
3. View Summary
4. Exit
Enter choice (1-4): 3

=========================
 JOURNAL SUMMARY
=========================
Name: Covid-19
Type: Unspecified
Total Events: 2
Last Event:   2025-11-24 22:48
---------------
Name: Typhoid
Type: Symptom
Total Events: 2
Last Event:   2025-11-24 22:48
---------------
Name: Fever
Type: Medication
Total Events: 2
Last Event:   2025-11-24 22:48
---------------
```

## 9. Testing Approach

- **Unit Testing:** Verified that adding a duplicate item name does not crash the system**.**
- **Boundary Testing:** Tested selecting item #0 or a number larger than the list size (System correctly reports "Invalid number").
- **Format Testing:** Verified that timestamps are readable and accurate to the minute.

## 10. Challenges Faced: -

- **Data Volatility:** Initially, data was lost every time the program closed. This was solved by implementing file I/O (Input/Output) to write to a CSV file.
- **User Input Errors:** Users entering text when asked for a menu number caused crashes. This was resolved by implementing Exception Handling.

## 11. Learnings & Key Takeaways: -

- Mastered the use of Python dictionaries to store structured data.
- Learned how to implement input validation to make software robust against user error.
- Understood the importance of timestamps in creating meaningful health data.

## 12. Future Enhancements: -

- **Visualizations:** Integrate matplotlib to generate graphs showing symptom frequency over time.
- **GUI:** Build a graphical interface using Tkinter or Streamlit.
- **AI Analysis:** Implement correlation algorithms to suggest potential triggers for symptoms based on medication history.

## 13. References: -

1. Python 3 Documentation: docs.python.org
2. "Automate the Boring Stuff with Python" by Al Sweigart.
3. VITyarthi Project Guidelines