Project Title: Blood Pressure-Tracker

NAME: RUDRANEEL ACHARJEE

REGISTRATION NUMBER: 25BAS10007

DEADLINE: 25TH NOVEMBER,2025

INTRODUCTION:

PROJECT OVERVIEW:

This project is a dedicated Blood Pressure Tracking application designed to help users monitor and manage their cardiovascular health over time. As a first-year student project, it demonstrates the application of fundamental programming, data handling, and basic software design principles.

The core objective is to provide a reliable digital log for users to quickly record their Systolic (SYS), Diastolic (DIA), and Pulse (HR) readings, visualize their personal health trends,

and automatically categorize readings based on standard clinical guidelines.

KEY FEATURES (FUNCTIONAL REQUIREMENTS):

The application includes three major functional modules and fulfills standard CRUD operations:

- **Data Input Module (Create):**

Simple interface for logging new blood pressure readings (SYS, DIA, HR).

Automatic timestamping and input validation to ensure data integrity.

- **Data Viewing & Analysis Module (Read & Processing):**

Displays a comprehensive history log of all recorded readings.

Automatically categorizes readings (e.g., Normal, Elevated, Hypertension Stages) for quick interpretation.

Calculates and displays overall average blood pressure and recent reading dates.

- **Reporting and Visualization Module:**

Generates customizable line charts to visualize blood pressure trends over flexible periods (e.g., 7, 30, or 90 days).

Allows users to filter the history log by specific date ranges.

- **Data Management (Update & Delete):**

Provides functionality to easily edit or delete existing log entries for data correction.

TECHNOLOGY STACK:

Backend/Core Logic

[INSERT YOUR LANGUAGE/FRAMEWORK HERE: e.g., Python, Java, Node.js]

Purpose: Data validation, categorization logic, and mathematical calculations.

Database/Storage

[INSERT YOUR DATABASE HERE: e.g., SQLite, MongoDB, Firebase, or a simple CSV/JSON file]

Purpose: Persistent storage for blood pressure logs.

- **Frontend/UI:**

[INSERT YOUR UI TECH HERE: e.g., HTML/CSS/JavaScript, React, Angular, Console Application]

Purpose: Primary interface for user interaction and data presentation.

- **Visualization:**

[INSERT YOUR CHARTING LIBRARY HERE: e.g., Chart.js, D3.js, or Matplotlib]

Purpose: Generating trend graphs for health analysis.

Version Control:

Git

Repository Link: https://github.com/rudraneel25bas10007-ctrl/Blood-Pressure-Tracker.Git

Non-Functional Requirements (NF-R):

- **Usability**

The project targets ease of use via an intuitive interface that allows for the quick and error-free entry of data.

- **Data Security**

Security features involving user health data should be stored using appropriate security mechanisms.

- **Error Handling**

Provide suitable validation to block impossible or clearly incorrect health readings.

- **Reliability**

All blood pressure readings shall be computed accurately and classified according to stated clinical standard

FUNCTIONAL REQUIREMENTS:

- **Data Input Module (Create):**
- **Data Viewing and Analysis Module:**
- **Reporting and Visualization Module:**
- **Data Management (Update & Delete)**

NON-FUNCTIONAL REQUIREMENTS:

- **Usability**
- **Data Security**
- **Error Handling**
- **Reliability**

PROBLEM STATEMENT:

The modern healthcare paradigm emphasizes proactive, long-term monitoring of vital signs. It is cumbersome, error-prone, and disorganized for patients with or at risk of cardiovascular conditions such as hypertension to record in analog logs their blood pressure readings. This is a problem because there is a lack of available, reliable, and analytic digital mechanisms that can provide tracking of personal health data continuously for timely self-management and physician consultation.

AIM OF THE PROJECT:

The scope of the Blood Pressure Tracker is to develop a single-user application, which handles the complete life cycle of blood pressure data right from secure ingestion to meaningful visualization and management. Inclusions

Recording systolic, diastolic, and pulse readings.

CRUD Persistent data storage and retrieval.

Categorization of readings was performed automatically according to current clinical standards.

Time-series data visualization-graphical representation of trends.

Basic data quality control via input validation and error handling.

Exclusions:

- **Integration with external hardware, such as automated blood pressure cuffs.**
- **Direct communication with EHR systems.**
- **Providing complex medical diagnosis or prescriptive clinical advice.**
- **Multi-user accounts, or cloud synchronization beyond the data storage mechanism of the repository.**

TARGET USERS:

The application is intended for users needing routine and long-term blood pressure monitoring. People suffering from Hypertension

The primary audience requiring the tracking of a drug's efficacy or certain changes in lifestyle.

- **Seniors and Caregivers:**

Users who need a straightforward and efficient interface for logging and reviewing health data.

- **Fitness and Health Devotees:**

Proactive individuals who track physiological data as part of a wider wellness routine.

HIGH-LEVEL FEATURES The project involves three main functional modules that the project specifications asked for: Data Input

Provides a dedicated screen for quick, validated entry of new blood pressure readings.

Functionality: Create

Data History and Analysis

Provides a sortable log of all historical readings, complete with automatic categorization and calculation of summary averages.

Functionality:

Read / Processing

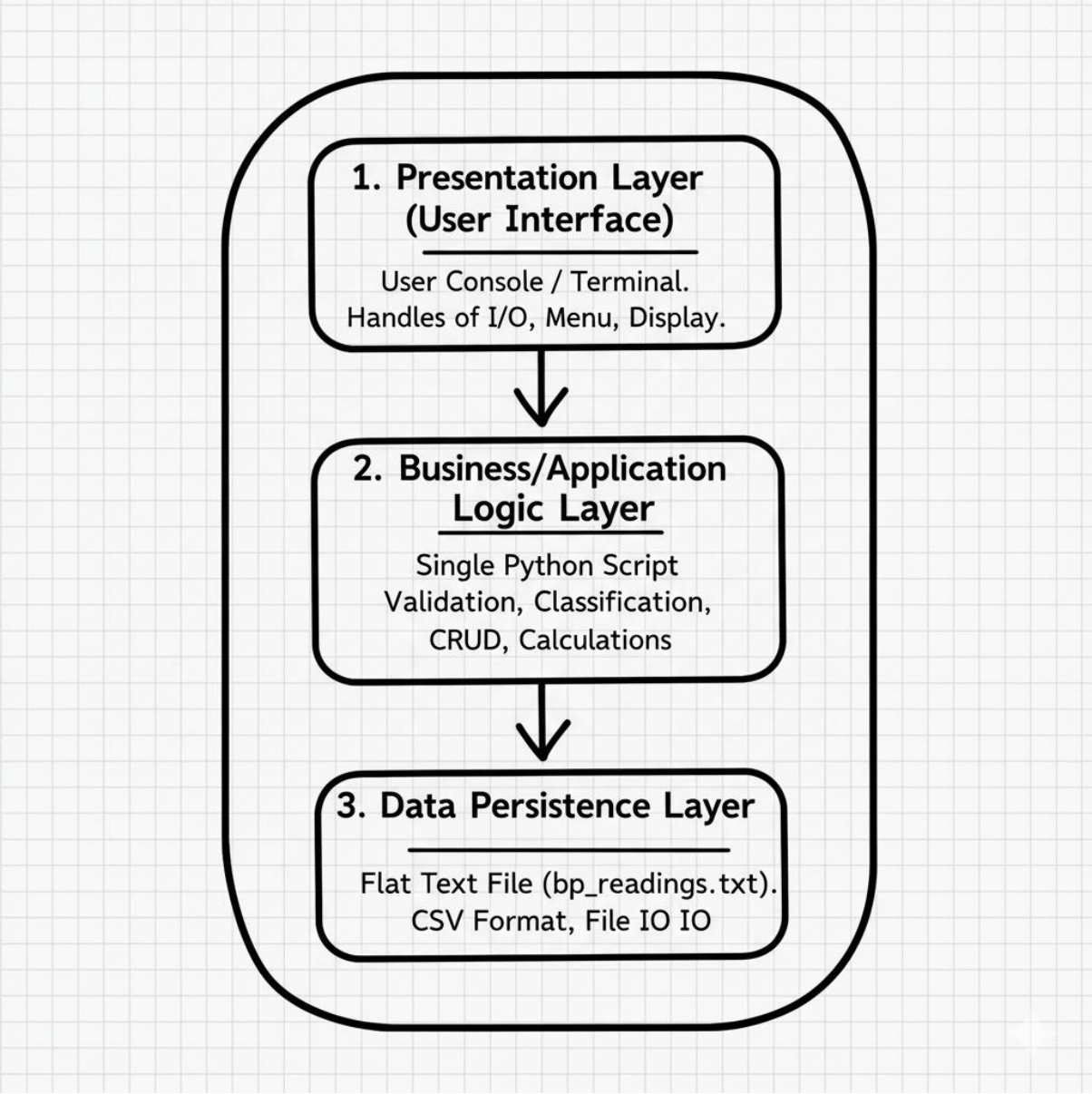
Trend Visualization:

Creates graphical charts to illustrate the fluctuation in blood pressure over time, such as 30/90 days.

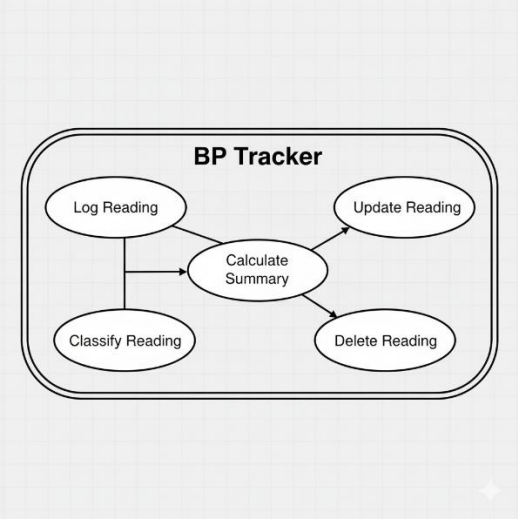
Functionality: Reporting Data Management Allow the user to correct mistakes, or permanently delete certain entries from the history log. Functionality:

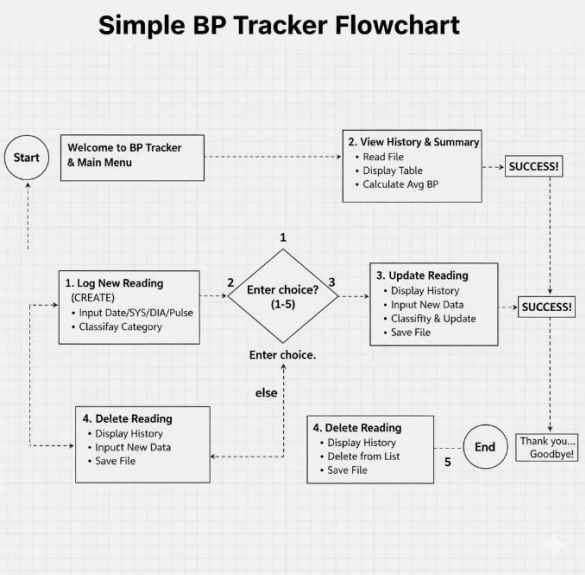
Update / Delete

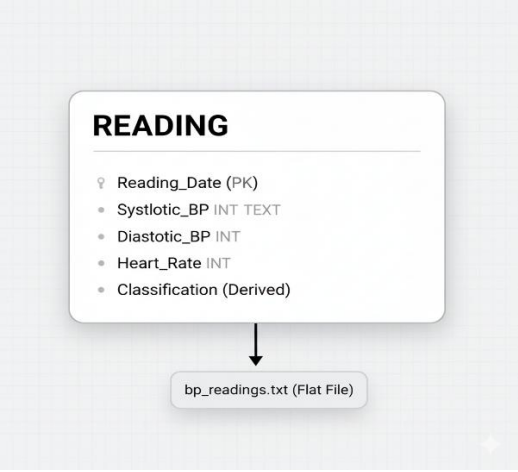
SYSTEM ARCHITECTURE:



DESIGN DIAGRAMS:







Implementation Details:

Implementation does fulfil all the necessary CRUD (Create, Read, Update, Delete) by utilizing in-memory processing and file overwrites, where necessary for the chosen flat-file configuration.

Data Validation and Classification:

Input for Systolic, Diastolic, and Pulse readings is validated using chained while loops and the string method .isdigit() so that no non-numeric entry will occur. This approach means that there is no use of try/except error handling but instead uses explicit control flow. The readings, once validated, are then fed into an inline Classification Logic- if/Elif/else chain-that classifies the health status based on clinical thresholds.

CRUD Operations:

The core logic manages data by treating the text file (bp_readings.txt) as a persistent list of records:

log-reading: The input data is formatted into one comma-separated string and written directly to the bp_readings.txt file using the 'a' Python file mode.

Read (View History): The application opens the file in the read mode 'r', and after reading all the existing data into the list of dictionaries in memory, it goes through the list to display the history along with an INDEX and to calculate Summary Statistics: average BP, total readings.

Update and Delete: These operations use a three-step Read-Modify-Rewrite pattern because of the flat-file storage limitation:

All data is loaded from the file into a Python list.

Procedure The user is asked for the index of the record to be deleted/updated. In case of Delete operation, the list item at the provided index is deleted (del readings_data[index]), while in case of Update operation its fields are overwritten with new validated data.

'w' mode overwrites the entire data file with the modified list. This is an important step because it ensures that disk storage mirrors what was changed in memory, so data integrity is preserved.

SCREENSHOTS:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Rudra\OneDrive\Desktop\Blood Pressure-Tracker> python BP-Tracker.py
Welcome to the Blood Pressure Tracker (Single Block Console App - No Imports)

--- Main Menu ---
1. Log New Reading (CREATE)
2. View History & Summary (READ)
3. Update/Edit a Reading (UPDATE)
4. Delete a Reading (DELETE)
5. Exit
Enter your choice (1-5):
```

```
--- Main Menu ---
1. Log New Reading (CREATE)
2. View History & Summary (READ)
3. Update/Edit a Reading (UPDATE)
4. Delete a Reading (DELETE)
5. Exit
Enter your choice (1-5): 1

--- Log New Reading (CREATE) ---
Enter today's date (e.g., YYYY-MM-DD): 2025-11-20
Enter Systolic (SYS) reading: 120
Enter Diastolic (DIA) reading: 80
Enter Pulse (HR) reading: 70

SUCCESS! Reading recorded (120/80) and classified as: Hypertension Stage 1
```

```
--- Main Menu ---
1. Log New Reading (CREATE)
2. View History & Summary (READ)
3. Update/Edit a Reading (UPDATE)
4. Delete a Reading (DELETE)
5. Exit
Enter your choice (1-5): 2

--- Blood Pressure History (READ) ---
INDEX DATE                SYS   DIA   PULSE CATEGORY
------------------------------------------------------------------------
1     2025-11-19          91    90    75    Hypertension Stage 2
2     2025-11-20          120   80    70    Hypertension Stage 1
------------------------------------------------------------------------
Total Readings: 2
Overall Average BP: 106/85
```

```
--- Main Menu ---
1. Log New Reading (CREATE)
2. View History & Summary (READ)
3. Update/Edit a Reading (UPDATE)
4. Delete a Reading (DELETE)
5. Exit
Enter your choice (1-5): 3

--- Select Reading to UPDATE ---
INDEX DATE                SYS   DIA   PULSE CATEGORY
------------------------------------------------------------------------
1     2025-11-19          91    90    75    Hypertension Stage 2
2     2025-11-20          120   80    70    Hypertension Stage 1
------------------------------------------------------------------------
Enter the INDEX number to update (1 to 2), or 0 to cancel: 1

--- Editing Reading #1 ---
Enter NEW date (2025-11-19): 2025-11-20
Enter NEW Systolic (SYS) reading: 91
Enter NEW Diastolic (DIA) reading: 90
Enter NEW Pulse (HR) reading: 70

SUCCESS! Reading #1 updated to 91/90 and classified as: Hypertension Stage 2
```

```
--- Main Menu ---
1. Log New Reading (CREATE)
2. View History & Summary (READ)
3. Update/Edit a Reading (UPDATE)
4. Delete a Reading (DELETE)
5. Exit
Enter your choice (1-5): 4

--- Select Reading to DELETE ---
INDEX DATE                SYS   DIA   PULSE CATEGORY
------------------------------------------------------------------------
1     2025-11-20          91    90    70    Hypertension Stage 2
2     2025-11-20          120   80    70    Hypertension Stage 1
------------------------------------------------------------------------
Enter the INDEX number to delete (1 to 2), or 0 to cancel: 1

SUCCESS! Reading #1 has been deleted.

--- UPDATED Blood Pressure History ---
INDEX DATE                SYS   DIA   PULSE CATEGORY
------------------------------------------------------------------------
1     2025-11-20          120   80    70    Hypertension Stage 1
------------------------------------------------------------------------
```

Testing Approach:

Since no automated unit testing framework was used in the project, it relies on a Manual Feature Testing approach for core functionality checks. Testing has been focused on data integrity and classification logic mainly.

Testing Scenarios:

The following was successfully tested against the final implementation:

Input Validation Check - Verified that non-numeric input, such as "ABC", as well as out-of-range numerical input like "0" or "500" are correctly rejected by the inline while loop validation to prevent corrupted data entry.

Data Categorization Check:

The test inputs map correctly to their established clinical categories. For example, 135/85 should be categorized as Hypertension Stage 1.

CRUD Integrity Check:

Ensured that logging, viewing, and deletion do not corrupt existing data; this means the process of reading and rewriting a file is sound.

Challenges Faced:

The greatest challenges experienced in this project revolved directly around the self-imposed constraints and the architectural choice:

File I/O Overhead for CRUD: To implement Update and Delete on a flat text file, the entire file had to be brought into memory, changed, and entirely rewritten for each operation. While this would work, it is highly inefficient and causes considerable processing overhead compared to using an appropriate database solution. Herein lies one important scalability issue.

The "No Imports" Constraint: The mandate to use no external modules created several challenges:

Timestamping: The automatic generation of precise, reliable timestamps was replaced by simple manual date entry by the user, compromising data reliability.

Error Handling: Due to the prohibition on the try/except structure-which is the standard for error handling in Python-the code had to become verbose to handle simple input validation, using multiple chained if/else and while loops; this complicated the code structure.

File Existence Check: Without os module, it will be a non-trivial task to check if bp_readings.txt exists. If the user tries to 'Read' before the first 'Create' operation, this could result in a runtime FileNotFoundError.

Single Block Architecture: It required duplicating every step of input validation, file handling, and classification inline, multiple times. This severely reduced the code maintainability and readability compared to a clean, modular design using functions.

Learnings & Key Takeaways:

This project was a comprehensive introduction into the basics of programming:

Core Control Flow: Ability to use the structures of sequence, decision (if/else), and repetition (while loops) appropriately.

Data Structures: Practical use of lists to hold readings and dictionaries to manage the reading attributes in Python.

File Handling: Gained experience with basic file input/output, using the 'a', 'r', and 'w' modes of a file to achieve persistence of data.

Algorithm Implementation: Core classification algorithm-checking blood pressure against threshold ranges-was implemented with great success to transform raw data into clinically meaningful output.

Future Improvements:

Further development would focus on the enhancement of usability and the robustness of the application:

Graphical Visualization: Replace the console outputs with real Trend Visualization charts by incorporating an external module such as Matplotlib; this will fully complete Section 2.3.

Modular design: re-factor the code base to use functions and classes where appropriate, hence improving readability, maintainability, and testability.

Timestamping: Reintroduce the use of the datetime module to provide accurate, automatic timestamps, instead of manual user input.

Robust Data Storage: Migrate from a flat text file to a simple, embedded database (such as SQLite) to manage the data efficiently and avoid rewriting the entire file at each update/delete operation.

REFERENCES:

- **PYHTON OFFICIAL DOCUMENTATION: docs.python.org**
- **GITHUB HELP DOCUMENTATION: docs.github.com**
- **VS CODE DOCUMENTATION: code.visualstudio.com/docs**
- **COLLEGE COURSE MATERIALS AND TEXTBOOKS**
- **ONLINE PROGRAMING RESOURCE(VITYARTHI)**