Design & Documentation Requirements

Problem Statement

Tracking daily health data such as weight, calorie intake, and exercise is vital for personal health management. However, many individuals find it difficult to consistently record and monitor this data. This project addresses this challenge by developing a simple, interactive Python application that allows users to log and review their health-related metrics easily over time.

Objectives

- Create a user-friendly console-based application for health data logging.
- Ensure persistent storage of user data with timestamping for historical reference.
- Allow retrieval and display of stored data by category for user insight.
- Implement modular, maintainable code adhering to basic software engineering principles.
- Lay the foundation for future enhancements with clear documentation and design.

Functional Requirements

- The app shall allow users to log their weight with the current date.
- The app shall allow users to input daily calorie consumption.
- The app shall allow users to record exercise type and duration.
- Data shall be saved persistently by appending to text files for each category.
- Users shall be able to view all recorded entries for weight, calories, or exercise.
- The app shall handle error cases such as missing data files gracefully.
- Provide a menu interface for users to select operations and exit safely.

Non-functional Requirements

- The application shall be runnable on any system with Python installed.
- Use simple text files for easy portability and minimal dependency.
- The interface shall provide clear prompts and feedback messages.
- Data shall be stored securely without overwriting prior entries.
- Operations should be lightweight to execute quickly via command line.
- Code should be modular for ease of maintenance and scaling.
- Error handling should prevent application crashes and guide users.

System Architecture Diagram (Description)

The architecture consists of:

- User Interaction Layer: Command-line interface presenting a menu and accepting user data inputs.
- Application Logic Layer: Python functions for input validation, data formatting, logging, and retrieval.
- Data Persistence Layer: Plain text files corresponding to weight, calories, and exercise logs where entries are appended.

Each layer communicates sequentially to process user requests.

Process Flow / Workflow Description

1. Start application: Display welcome message and main menu.
2. User selects an action:
    - Record weight/calories/exercise
    - View stored data
    - Exit app
3. On recording input:
    - Prompt user for the relevant data fields.
    - Retrieve current system date.
    - Format data combining date and user input.
    - Append data to respective log file.
    - Confirm success to the user.
4. On viewing data:
    - Ask user which category to view.
    - Open appropriate log file.
    - Display content on screen or notify if empty.

5. Loop back to menu until exit chosen.

UML Design Diagrams (Conceptual)

- Use Case Diagram: User interacts with system to record/view health data and exit.
- Class Diagram (Suggested Refactor):
  - `HealthTrackerApp` class encapsulating methods for data logging and retrieval.
- Sequence Diagram: Illustrates the steps from user choosing "record weight" to confirmation message.

Storage Design

- Data Storage Type: Plain text files for ease of use.
- Files: `weight_log.txt`, `calories_log.txt`, `exercise_log.txt`
- Data Schema per Entry:
  - Date stamp (YYYY-MM-DD) plus value (e.g., `2025-11-24: 68 kg`).
- Sample entry formats:
  - Weight: `2025-11-24: 68 kg`
  - Calories: `2025-11-24: 2000 kcal` (should be fixed from current "kg")
  - Exercise: `2025-11-24: Running 30 minutes` (currently logs exercise and duration but output format can be enhanced)

Code Architecture Features & Design Highlights

- Modularity: Code uses distinct functions for logging (`log_data`), recording different data types, viewing logs, and the main interaction loop.
- Date Handling: Uses Python's `datetime.date.today()` function for date stamping each entry, giving historical context.
- File I/O Best Practices: Files are accessed using context managers (`with open(...)`) to ensure proper resource handling.
- User Input Handling: Inputs are taken via `input()` function with clear prompts guiding the user.

- Exception Handling: Graceful handling of file-not-found errors when viewing data, avoiding crashes.
- Menu-driven Loop: Maintains continuous interaction until the user explicitly chooses to exit.
- User Feedback: Informative print statements confirm actions or alert invalid choices.

Improvement Opportunities

- Input Validation: No current checks for valid numeric input or reasonable ranges; adding validation logic would improve robustness.
- Unit Consistency Fix: Calorie logging mistakenly adds "kg"; should add "kcal" or just calories.
- Enhanced Logging Format: Exercise log can be formatted to clearly separate exercise type and duration (e.g., `"2025-11-24: Running 30 minutes"`).
- Refactoring: Convert procedural code into an Object-Oriented Design with classes encapsulating behaviors.
- Documentation: Add detailed docstrings and inline comments describing each function's parameters and behavior for maintainability.
- Testing Framework: Write unit tests to verify individual function logic and application workflows.
- Project Structure: Organize code, data files, and tests into separate directories for scalability.
- Error Handling: Broaden exception handling for all user inputs to catch invalid formats and provide corrective prompts.

Summary

This Health Tracking App script is a solid foundational implementation for recording and reviewing personal health data with time stamps via text file logging. It meets fundamental program requirements through modular functions, date stamping, file handling, user feedback, and error management. With further enhancements in

validation, formatting, design modularity, and testing, it can evolve into a robust health management solution.

This detailed explanation covers the entire design, functional, non-functional, and architectural aspects required for full project documentation and presentation based on your original code.

If needed, I can help you generate diagrams or refactor this into a complete OO design with validations and tests next.

-