Habit Tracker Concept

In the spirit of agile project development, we want to start building the simplest possible solution and iterate from there. Since the requirements only describe the basic functionality for the backend of the habit tracking application it can be assumed that more functionality will be added in the future. To make sure the system can be extended easily it is best to avoid overcomplicating the architecture and to build a minimum viable product.

Within the system a habit is modeled as a class with a few simple attributes explained below.

1. Name: A habits name servers as an identifier for a habit within the system. For compatibility reasons we will not allow spaces or special characters here. This can easily be done in the frontend if necessary; however, the systems internal communication should be kept as simple as possible.

2. Period: The habits period is an integer type. It describes the habits periodicity in days. Using an integer here allows for the creation of any desired habit period. Daily, Weekly or Monthly periods can simply be entered into the system as 1, 7 or 30 respectively. With extensibility in mind, we will not set an arbitrary restriction here and enforce only a technical limitation.

3. Creation Date: This is simply the timestamp when the habit was created. It will be used together with the period to calculate the habit intervals.

4. Tasks: The Tasks attribute is a dictionary which stores all tasks belonging to a habit. A task can be completed by the user at any time. When a task is completed the timestamp for the completion will be stored. In the dictionary the keys will hold the tasks name and the value will be a set of timestamps storing the completions.

While the backend is running this data will be held in memory. The system will be capable of storing this information on the disk in the form of a JSON file. This approach keeps the system as simple as possible while also making it easy to attach a document-oriented NoSQL Database like MongoDB in the future.

Since this is only the backend users likely won’t ever interact with the system directly; however, for the sake of simplicity we will treat the exposed CLI as the user interface for now. Once the system is started the user will be able to enter commands to create or delete habits and tasks. Other functionality like checking off tasks or analyzing habit history will also be available here. The CLI is kept open the whole time and the user modifies the data in memory. Once the user is done a command can be issued to create a save file on the disk. This will keep load and store operations on the disk to a minimum, which helps with performance and scalability. In the event the system is extended using a database this approach will make it easy to implement transactions.

All operations manipulating or reading habit attributes will be written as methods of the habit class. All other operations will be written as functions not belonging to any class including, if necessary, wrapper functions.

Since this will be a production system, we need to ensure robust error handling. The philosophy here is to keep the system from crashing at all cost, even if additional checks come with a performance penalty. Custom exceptions will be the only other classes required besides the habit class.