HABIT TRACKING BACKEND APPLICATION

Conception Phase

Submitted by:

Atheek Mohamed Rafi

Matriculation: 9212873

Enrolled in:

B.Sc. Applied Artificial Intelligence

Object Oriented and Functional Programming with Python

Written assignment submitted to the

IU International University

In fulfilment of the partial requirements for the course of

DLBDSOOFPP01

Instructor Name: Prof. Max Pumperla
Date: 30/01/2024

Conception Phase - Habit Tracking Application Backend

1. Introduction.

Welcome to the design concept for the development of my Habit Tracking Application! The goal of this application is to assist users in establishing and maintaining habits by providing a tracking system. In this phase I will outline the components, interactions and design choices that serve as the basis for my habit tracking backend.

2. Overview of Solution

My solution comprises several classes, and modules, each serving a specific purpose:

1. db.py (Database module):
   - Responsible for managing the database.
   - Handles storage and retrieval of habit and event data.

2. habit.py (Habit Class):
   - Represents the core entity, a habit.
   - It includes attributes such, as name, description, periodicity, current streak, and creation time.
   - Utilizes the Database module to access and manipulate habit related data.

3. analytics.py (Analytics Module):
   - This module analyses habit data from the database.
   - It provides functionality to retrieve insights such as the longest streaks.

4. main.py (Entry Point):
   - The main.py file initializes the application.
   - It manages user interactions.

5. helper.py (Helper Module):
   - It acts as mediator between the main.py and other modules.
   - It is basically an extension of main.py helping to get user inputs.
   - It streamlines the flow of user interactions enhancing modularity and maintainability.

6. test_habit.py (Unit Test Module for Habit Class):
   - Validates the functions and methods of the Habit class.
   - Uses testing framework pytest

7. test_analytics.py (Unit Test Module for Analytics Module):
   - Validates the functions and methods of the Analytics module.
   - Uses testing framework pytest

8. README.md (User Instructions)

   - Users can easily download and start the application by following the instructions in the README.md file.

3. Relational Database Choice

For this project I chose SQLite as a database due to its simplicity and suitability. When compared to other databases SQLite is self-contained and requires minimal setup. This feature is advantageous as it ensures a user experience without complex database management tasks. The structured nature of a database also allows for querying and retrieval of habit related data.

The database comprises two tables:

- Habit Information Table: Stores details of each habit.
- Event log Table: Records events, such as users checking off habits.

4. User Interaction

Users primarily interact with the application through a Command Line Interface (CLI). Key functionalities include:

- Creating new habits with customizable information.
- Deleting habits, they no longer wish to track.
- Completing a habit with corresponding events logged.
- Viewing and analysing habits
- Customizing habit information, e.g., updating name, description, and periodicity.

UML use case diagram given below describes the interactions between the user and the Habit Tracking App.

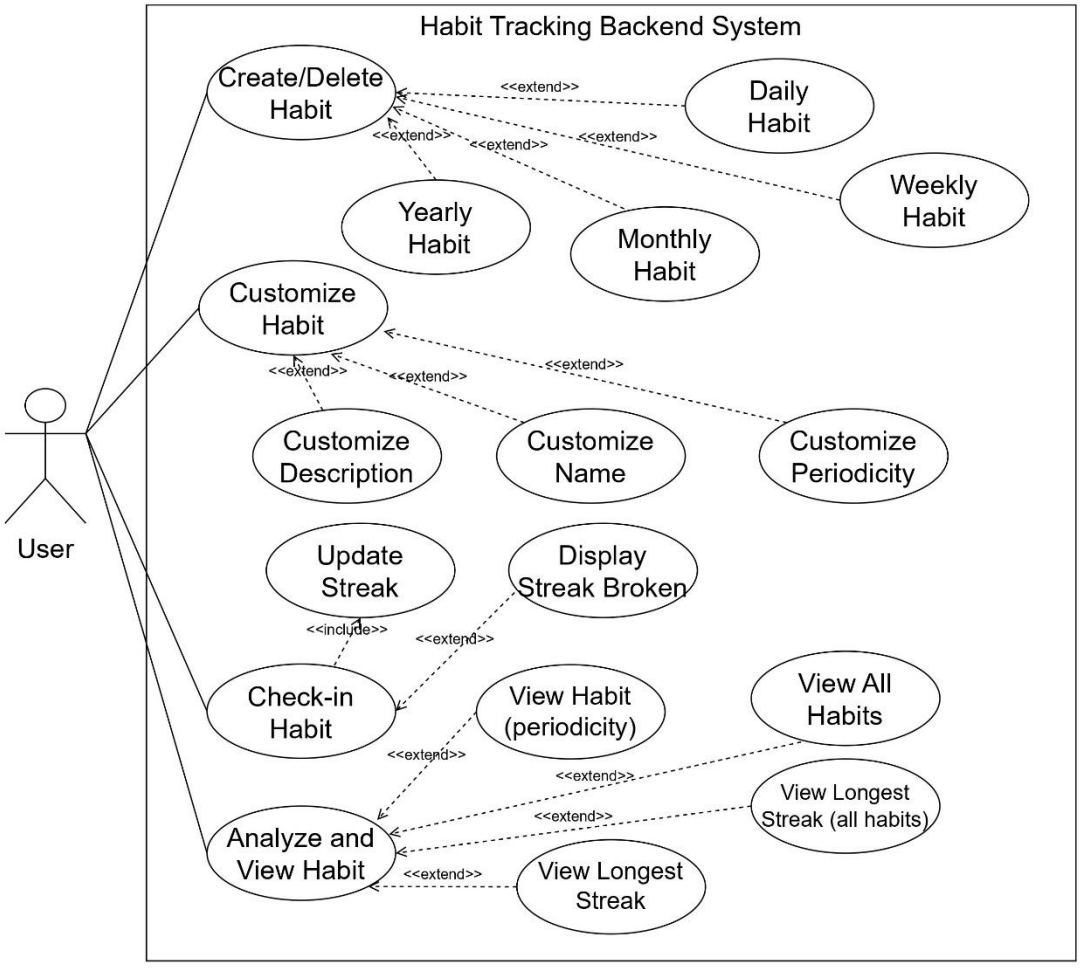

Figure 1: Illustrated by author – UML Use Case diagram of Habit Tracking Backend Application

5. Component Interaction

The main entry point, main.py, interacts with helper.py to maintain simplicity. helper.py serves as an intermediary layer, connecting with:

- habit.py for habit-related logic, utilizing the db.py class for database interactions.
- analytics.py for habit data analysis, also interacting with db.py for database queries.

Example 01: when creating a habit, main.py -> helper.py -> habit.py -> db.py facilitates the creation of a new habit entry in the database.

Example 02: when user is requesting to view all habits, main.py -> helper.py -> analytics.py -> db.py facilitates the retrieval of habit data in the database.

UML sequence diagram for the “Example 01” is given below:

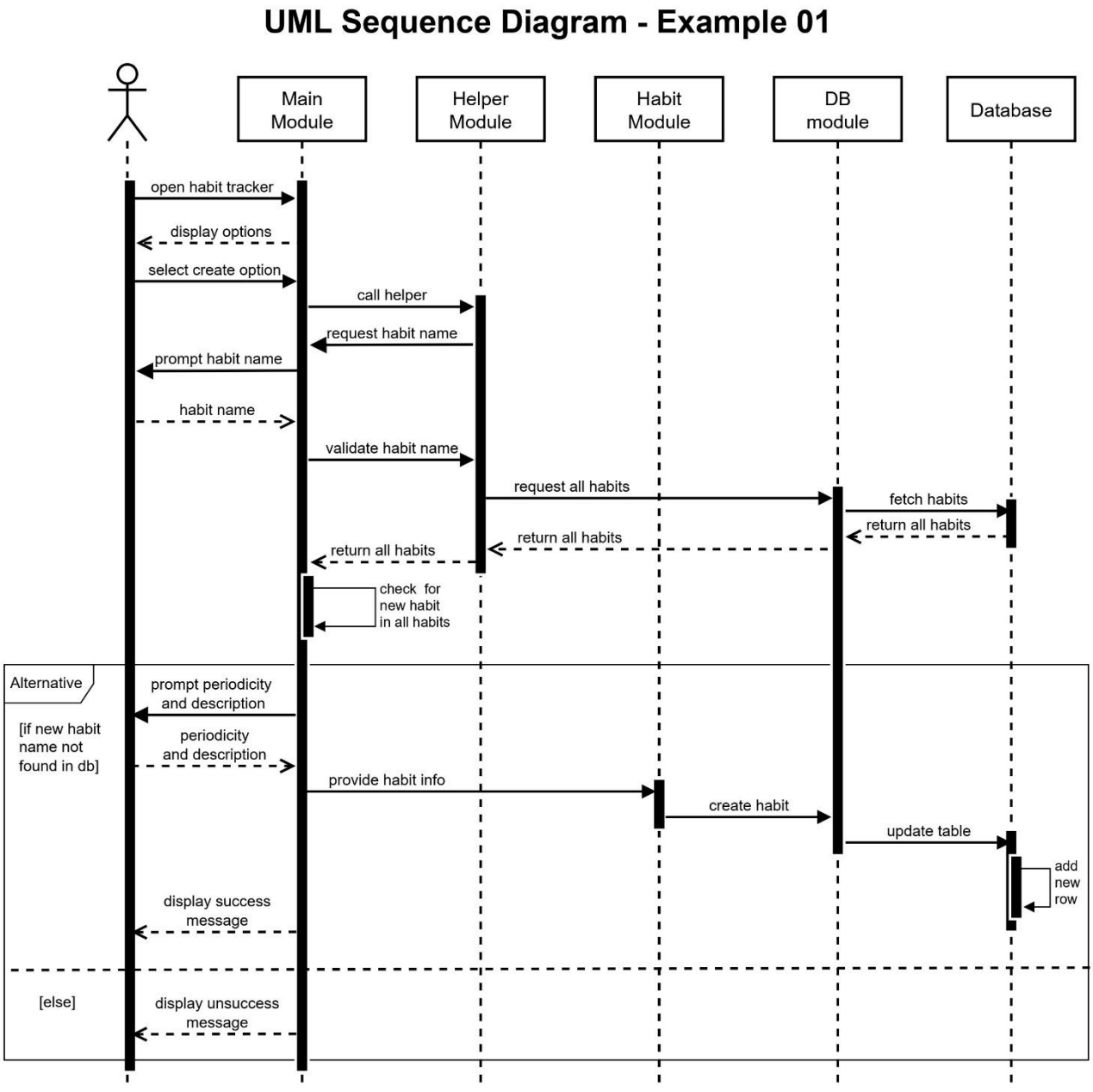

Figure 2: Illustrated by author – UML Sequence diagram for the example 01 given above.

6. Conclusion

As I conclude the phase, it's important to acknowledge the flexibility and scalability of the proposed backend architecture. The design incorporates Object Oriented Programming and functional programming principles to facilitate expansion of functionality in future development stages. The combination of a relational database, CLI (Command Line Interface) and modular component interactions lays a strong foundation not only meeting current requirements but also setting up possibilities for future improvements.

With a focus, on user experience and maintainability, my backend design aims to establish a groundwork aligned with my vision of building a user-friendly Habit Tracking Application.