*Zewinger, Moritz*

*Matriculation Number: 14141201*

*Object Oriented and Functional Programming with Python (DLBDSOOFPP01)*

*Habits Phase 1 – Conception*

---

## Introduction

The goal of this project is to create a functional backend for a habit tracking app that allows the user to create, modify, and analyze habits in the CLI. The project will include documentation for running, setting up, and interacting with the program as well as the containerized application itself, including the database. The database already comes with predefined data for testing purposes.

## Requirements Analysis

Before we start with the development, it is crucial to create a concept of the application. This ensures that we don't just start by blindly implementing features and having to refactor most of it at the end or if errors occur. A great concept defines the whole outline of the project and ensures that the development phase doesn't fall behind schedule or run into other unforeseen problems such as complexity or missing requirements.

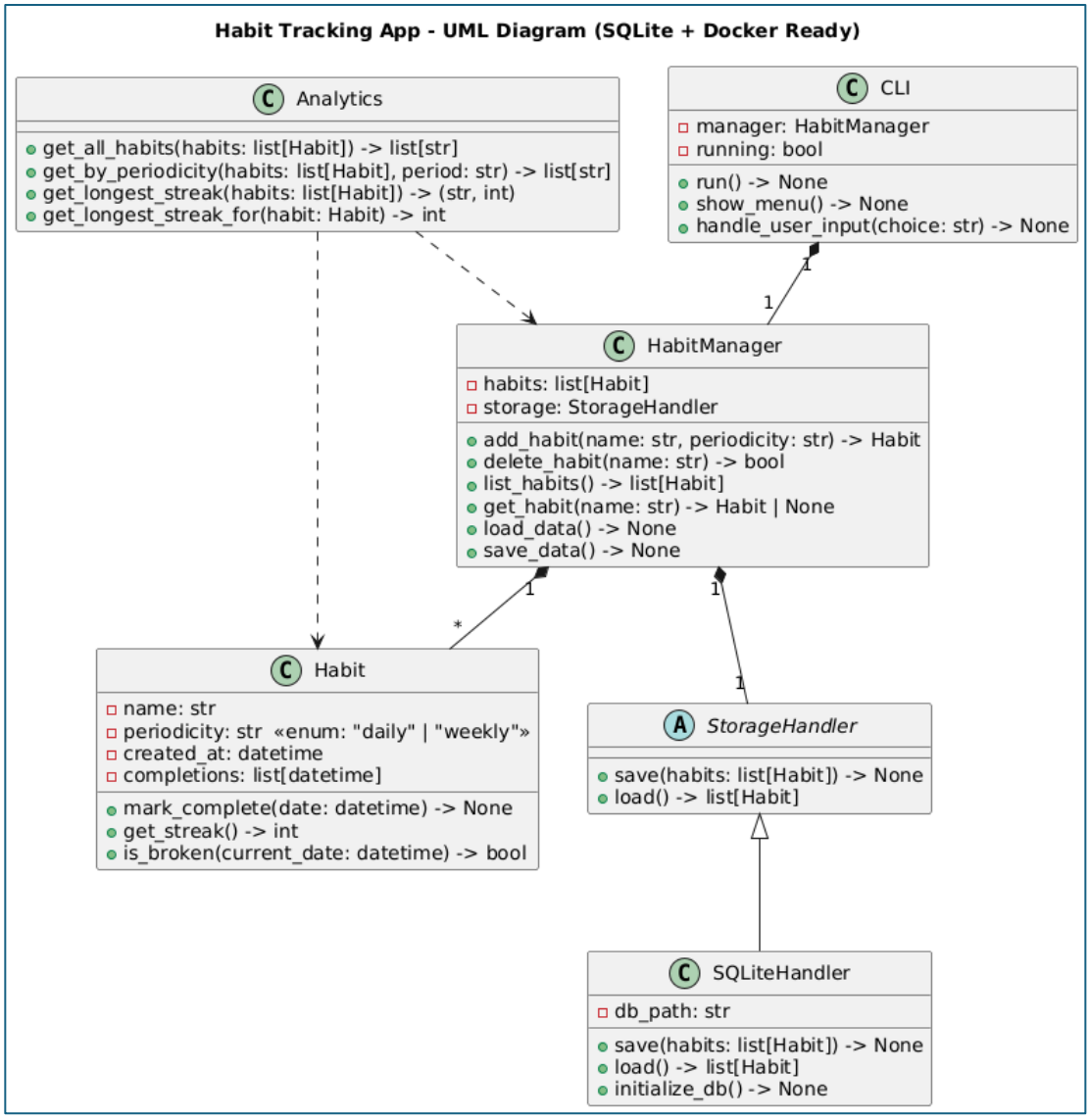
For the Habit tracking program, the following requirements are given:

- Enable users to create and manage multiple daily or weekly habits.
- Track completions and calculate streaks.
- Provide analytics functionality (e.g., longest streaks, habits by period).
- Store data persistently (either in a database or JSON file).
- Offer a command-line interface for user interaction.
- Ensure modularity and reusability for future extensions.

These are the core requirements for the functionality of the program. There are other requirements as well, but it would make no sense to list all of them since they are already given and not visible to the user (such as the use of object-oriented principles and functional programming or unit tests).

## System Design

The system follows a modular, object-oriented design and consists of multiple classes that interact with one another. All classes have built-in functions, which enable a seamless interaction with each object. The first and most important one would be the **Habit** itself. This class represents a single habit entity with the desired parameters, such as name, periodicity, etc. The second class would be the **HabitManager**, which handles the interaction between the user through the CLI and each object of the program. Another class would be the **CLI**. The CLI is the entry point for the users and enables communication with the system since there won't be a UI. The **Analytics** class handles everything that is needed to analyze the habits. At last, we have the abstract class **StorageHandler,** which defines a persistence interface in combination with the **SQLiteHandler** class that implements the interface using SQLite3.



## Data Storage

All habit data will be stored in an SQLite3 database, chosen for its simplicity and lightweight. The database stores tables for habits and completion timestamps. Using a **StorageHandler** abstraction allows switching or implementing other storage mechanisms (e.g., PostgreSQL) without changing the business logic. This is relevant if the application grows, but for now, SQLite3 is perfectly suitable. The data will be loaded into the application automatically or by providing a path for the database file.

## Functional Flow (User Interaction)

Interaction of the application is done through the CLI. They can add a new habit, mark a habit as completed, or analyze their habits. The input fields consist mostly of strings and integers rather than clicking with the mouse. First, a Docker container gets started, and the user is prompted to choose what he would like to do (create, delete, check, or analyze). The CLI then calls the **HabitManager** to perform the requested action. The **HabitManager** interacts with the **SQLiteHandler** to read or update stored data. Afterwards, the result is shown to the user through the CLI.

## Tools, Technologies, and Design Rationale

The chosen technologies in combination with the defined classes create a modular structure that promotes scalability and maintainability. Python 3.13 is used as the programming language for the application, and Click is used as the cli framework. SQLite3 is used for the database since it is lightweight, serverless, and easier to set up in different environments. Docker is used for containerization, which allows consistent and fast deployment in a platform-independent environment. PyTest is used for testing of the application. The use of object-oriented programming allows encapsulation of behavior within classes, while functional programming principles are applied in the analytics module for pure data transformation.

## Summary

The planned architecture establishes a solid foundation for a lightweight yet extensible habit-tracking backend. It combines object-oriented structure, functional data analysis, and containerized deployment to ensure reliability, scalability, and maintainability. The modular design guarantees that future phases—development, testing, and deployment—can be completed efficiently and with minimal refactoring.