Habit Tracking App Concept

Design and implementation overview

Introduction

App Concept

Design and implementation overview

This is a concept for a habit tracking application built using Python. This document outlines the core design principles, technical architecture, and development methodology. The app aims to provide a simple, intuitive, and effective way for users to track their habits and achieve their goals.

Design Overview

The application is structured around several key components:

Habit Class: Represents a single habit, including its name, description, frequency, and completion status.

Tracker Class: Manages the collection of habits, providing methods for adding, deleting, and updating habits.

User Interface: A command-line interface (CLI) or graphical user interface (GUI) for user interaction.

Data Storage: A mechanism for persisting habit data, such as a JSON file or a database.

The rationale behind this design is to create a modular and extensible system. Each component is responsible for a specific task, making it easier to maintain and update the application. The use of Python allows for rapid development and a wide range of available libraries.

Tools & Technologies

The following tools and technologies are considered for the implementation of the habit tracking application:

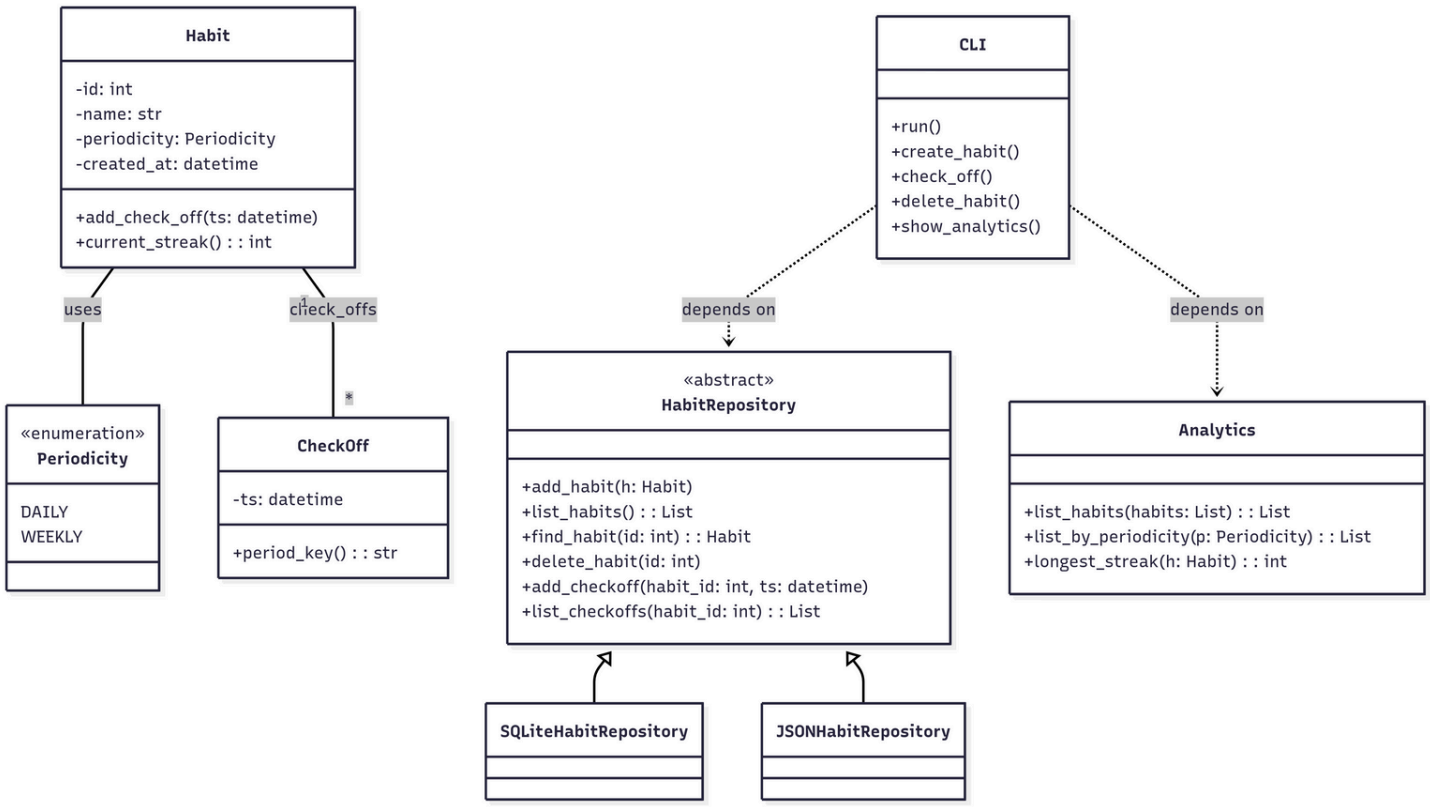
Python: The primary programming language.

SQLite: For local data persistence.

Click: For building the Command Line Interface (CLI).

Tkinter/PyQt: Potential GUI frameworks.

JSON: For data serialization and storage.



Class Relationships:

The class aggregates objects.

The class interacts with the class to manage habits.

The class is used by the class to persist habit data.

The diagram illustrates class relationships: Habit uses Periodicity and CheckOffs; CLI depends on Habit and Repository; Analytics processes Habit lists.

Methodology and Considerations

Approach: Start with model design (Habit class), then persistence (repository pattern for abstraction), analytics (pure functions to avoid side effects), finally CLI for interaction.

Why this structure: Repository pattern allows backend switching without changing core logic. Streaks calculated via date comparisons, handling daily/weekly differently.

Duplicate check-offs prevented by period checks.

User Interaction: Via CLI prompts/menus for habit management and analytics display (tables/panels).

Potential Issues: Timezone handling in dates (assume UTC), large data performance (optimized with indexes in SQLite).

Resources: No external data needed; sample data generator for testing. Git for version control.

This concept ensures modular, testable code aligned with course goals.