NAME: OLUWATOYIN ENIOLA IPADEOLA
MAT NUMBER: 92130758
COURSE CODE: DLBDSOOFPP01
COURSE: OBJECT-ORIENTED AND FUNCTIONAL PROGRAMMING WITH PYTHON

PORTFOLIO: HABIT TRACKING APP

TUTOR: Mirmehdi Seyedebrahim

HABIT TRACKER: CONCEPTION PHASE DOCUMENT

1. Project Overview and Goals

This project outlines the conceptual design for a Habit Tracker application using:

Object-Oriented Programming (OOP) → for modeling habits and completions.

Functional Programming (FP) → for analytics and streak logic.

SQLite → for persistent storage of users, habits, and completions.

The main goal is to create a modular, testable, and extendable system that enables users to:

- Register and log in.
- Create daily or weekly habits.
- Record completions with timestamps, notes, and mood scores.
- Analyze progress using streak calculations.

2. Core Application Components

The application is structured into modules, each with clear responsibility:

- Data Model (habit.py, completion.py)
    I. Defines the core classes of the app.
    II. BaseHabit, DailyHabit, WeeklyHabit (habits) – represent different types of habits.
    III. Completion (check-off records) – represents when a habit is checked off (with timestamp, notes, mood).
- StorageHandler (db.py): handles persistence with DB rules (PK, FK, unique)
    I. Manages SQLite database.
    II. Create tables for Users, Habits, and Completions.
    III. Stores and retrieves data so habits persist across sessions.
- HabitManager (habit_manager.py):
    I. Implement the business logic.
    II. add habits, list habits, update habits, check off completions, delete/deactivate.
    III. Connects the storage layer with the user interface.
- AnalyticsService (analytics_service.py): Provides Functional Programming logic:
    I. Calculate current streak and longest streak, overall longest streak.
    II. Finds inactive habits.
    III. Filter habits by daily/weekly.
- CLI UserInterface (user_interface.py)
    I. Menu-driven interface built with Questionary; test mode
    II. Lets user log in, add habits, check off, view analytics, and exit.
- Testing (pytest)
    I. Ensures the app works correctly.
    II. Unit test for each component.

III. Integration tests for workflows.
IV. Mock tests for CLI interactions

3. Data Model Design (OOP)

The OOP structure makes the code clean and reusable:

- BaseHabit → Attributes: Stores habit name, creation date, and completions.
  I. *name* (string) – the name of the habit.
  II. *_creation_date* (datetime) -when the habit was created.
  III. *_completion_records* (list) – list of all completions for this habit.

  Methods: *check_off(), get_completion_records(), __str__()*

- DailyHabit → Attributes: Inherits all **BaseHabits** attributes.

  Methods: *is_due_on* (date) – to check it is due daily

- WeeklyHabit → Attributes: Inherits all **BaseHabits** attributes.

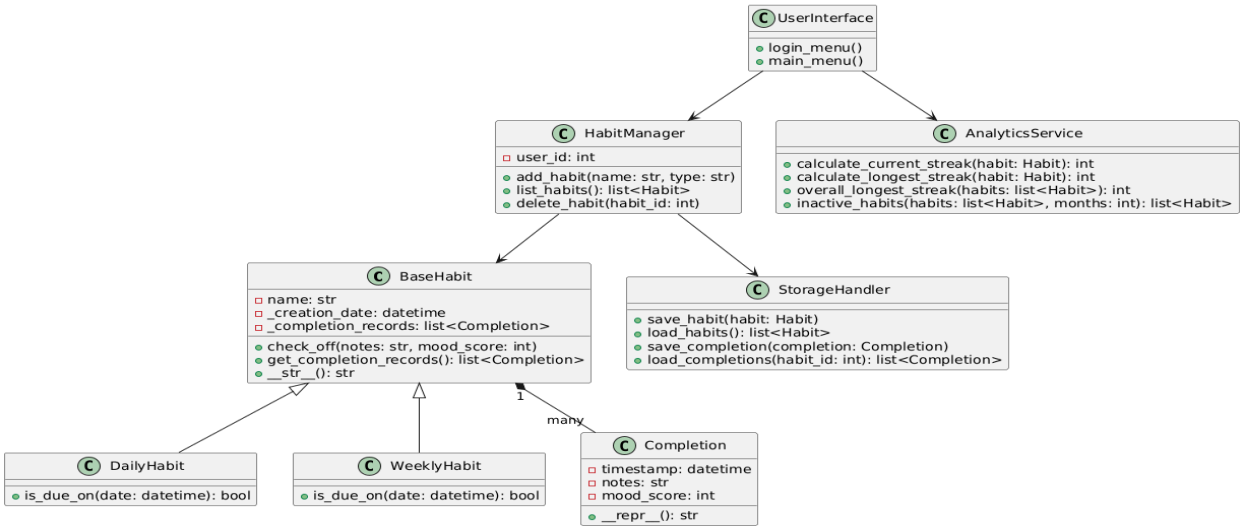
  Methods: *is_due_on* (date) – to check it is due weekly

- Completion → Attributes: Represents a check-off event with timestamp, notes, and optional mood score.
  I. *timestamp* (datetime) – when the habit is completed.
  II. *notes* (str, optional) – description or comments about the completion.
  III. *mood_score* (int, optional) – rating of how the user felt (1-10).

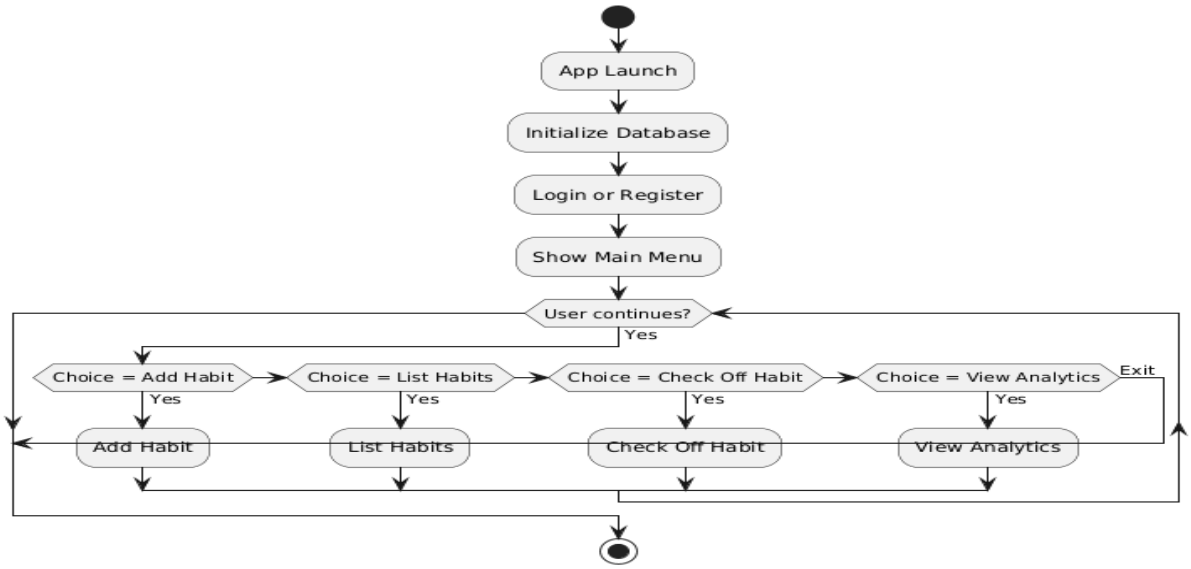
  Methods: *__repr__()*

This design uses inheritance and polymorphism for flexibility.

4. THE UML DIAGRAM



5. THE FLOW CHART



The user interacts through a menu-driven CLI:

- App Launch
- Initialize Database
- Register/Login
- Show main menu
- Add Habits (Daily/Weekly)
- Check Off Habits (with optional notes and mood score)
- List Habits
- View Analytics: Current streak, Longest streak, overall longest streak, Inactive habits, Filter by periodicity
- Delete/Deactivate Habits
- Exit

This flow enables users to manage their habits and view progress instantly and easily.

6. Database Design and Rules (SQLite): The app uses SQLite for reliable, file-based storage.
Schema: Primary Key(PK), Foreign Key(FK), Unique Key

- Users: (*user_id* (int, PK, autoincrement), *username* (string, Unique), *email* (string, Unique), *password_hash* (hashed with SHA-256 or bcrypt, never plaintext), *created_at* (datetime)
- Habits: (*habit_id(PK), user_id; Users.user_id (FK,), name (habit title), type(dailyor weekly), created_at, is_active)*
- Completions: (*completion_id(PK), habit_id; Habits.habit_id (FK), timestamp (not allowed in future), notes, mood_score (range 1-10)*)

Relationships:

- One user → many habits.
- One habit → many completions.

7. Analytics Logic (FP): Analytics are implemented using pure functions:

- *calculate_current_streak* (habit) → Count consecutive completions up to today.
- *calculate_longest_streak* (habit) → Find the maximum streak.
- *overall_longest_streak*(habits) → Find the maximum streak across all habits
- *inactive_habits* (habits, months=6) → Identify habits not updated in a set period.
- *get_habits_by_periodicity*(habits, type) → Filter daily vs weekly habits.

FP ensures predictability, immutability, and testability.

8. Clarifications

- Streak rules
  Daily habits: must be checked every calendar day
  Weekly habits: must be checked once per ISO calendar week
- Input checks
  No future timestamps (reject completions in the future)
  Mood scores must be between 1 – 10
- Non-interactive mode
  For testing: allow commands can run with flags

9. Sample Data

4 weeks, mix of daily and weekly, including month change. Data should include mood scores: e.g.
User: Toyin.
Daily Habit: “Read 10 pages” (logged daily across Jan-Feb, with edge case of month change).
Weekly Habit: “Exercise 3x” (logged once per week, including Feb).
Data includes mood scores (1-10)

10. Testing Strategy

Testing is performed with pytest across all modules:

- Data Model → Habit creation, completion handling, due checks.
- Storage → DB schema, (PK/FK/Unique constraints), inserts, retrieve.
- Manager → Add, list, delete, and check off habits.
- Analytics → Verify streak calculations and filtering.
- CLI → Mocked interactions to test menus and non-interactive commands

This ensures the project is robust, bug-free, and extendable. Constraints will be validated