Conception Phase of Creating a Habit Tracker App

UML Class Diagram

An UML diagram has been created to represent the structure of a habit tracker application, detailing seven key classes: **Habit, User, Storage, Analytics, Reminders, Completion, and Testing**. Each class is outlined with its respective attributes, including data attribute types such as integers and strings, to capture the essence of the data they hold. The diagram also illustrates the relationships between these classes, such as associations and dependencies, and includes the various methods that define the operations and interactions within the app. This UML diagram serves as a blueprint for understanding the app's functionality and the interconnectivity of its components, providing a clear guide for development and future reference.

Storage and Retrieval of Data

SQLite database file will be created, and structure of the tables defined as followed:

- **Users Table:** Stores user information.
    - Colums: UserID (primary key)
- **Habits Table**: Stores habits.
    - Columns: HabitID (primary key), UserID (foreign key), HabitName, Periodicity
- **Completions Table:** Records each time a habit is completed.
    - Columns: CompletionID (primary key), HabitID (foreign key), CompletionDate

Interaction of SQLite3 with Python:

```python
import sqlite3


2 usages (2 dynamic)
def add_habit(user_id, habit_name, periodicity):
    conn = sqlite3.connect('habit_tracker.db')
    c = conn.cursor()

    c.execute('INSERT INTO Habits (UserID, HabitName, Periodicity) VALUES (?, ?, ?)',
              (user_id, habit_name, periodicity))

    conn.commit()
    conn.close()

```

In addition, data can be:

- retrieved by using “SELECT” or
- updated by using “UPDATE” or
- deleted by using “DELETE”

General User Flow (please also refer to the user flow diagram created)

Users interact with the application primarily through a user-friendly interface that allows them to select, customize, and track predefined habits. They engage with the app by marking habits as completed, viewing their progress through visual indicators, and receiving reminders for upcoming habit completions. The app provides functionalities to adjust habit details, set reminders, and modify user settings for a personalized experience.

The general user flow of the app is as follows:

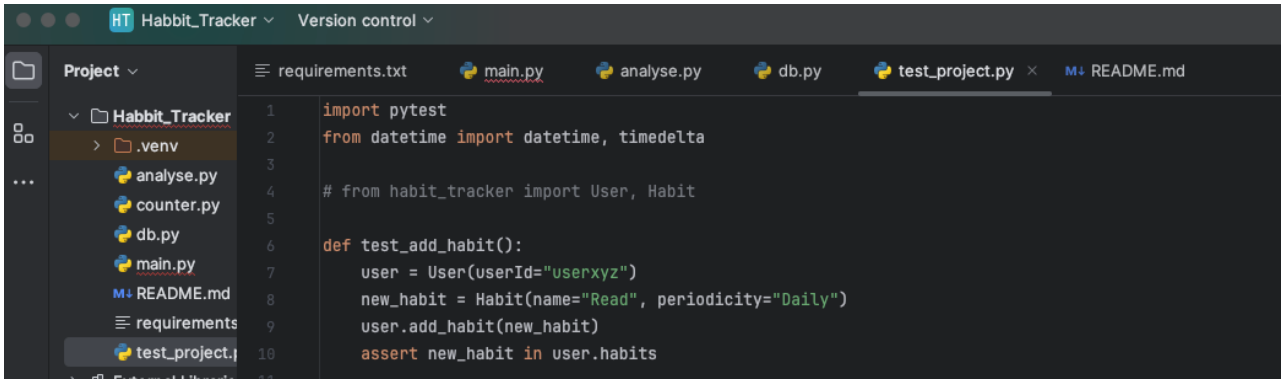
1. **Account Creation:** Users start by creating an account, providing them access to the application's features.
2. **Habit Selection:** Upon logging in, users are introduced to a list of predefined habits. They select habits of interest and customize them according to their goals.
3. **Habit Tracking:** Users regularly interact with the app to log their habit completions, which are then recorded and tracked by the system.
4. **Progress Review:** The application offers users insights into their habit-forming journey through streaks, progress charts, and summaries.
5. **Adjustments and Customization:** Users can modify their habits, set or change reminders, and adjust application settings as needed to fit their evolving goals.
6. **Notifications and Reminders:** The app sends timely reminders to users to encourage habit completion, based on the user-defined schedules.

Testing

Testing class (test_habits.py) will be created to ensure accuracy of the code. Below is how the testing will be setup using add_habit as an example. This test will verify that a habit can be successfully added to a user's list of habits. Data of 4 weeks will build the basis of the tests.

test_add_habit():

- **Setup:** Create an instance of the User class and an instance of the Habit class.
- **Action:** Use the add_habit method to add the habit to the user.
- **Assertion:** Check if the habit is now in the user's list of habits.



Analytics

Focus will be on the following analytics:

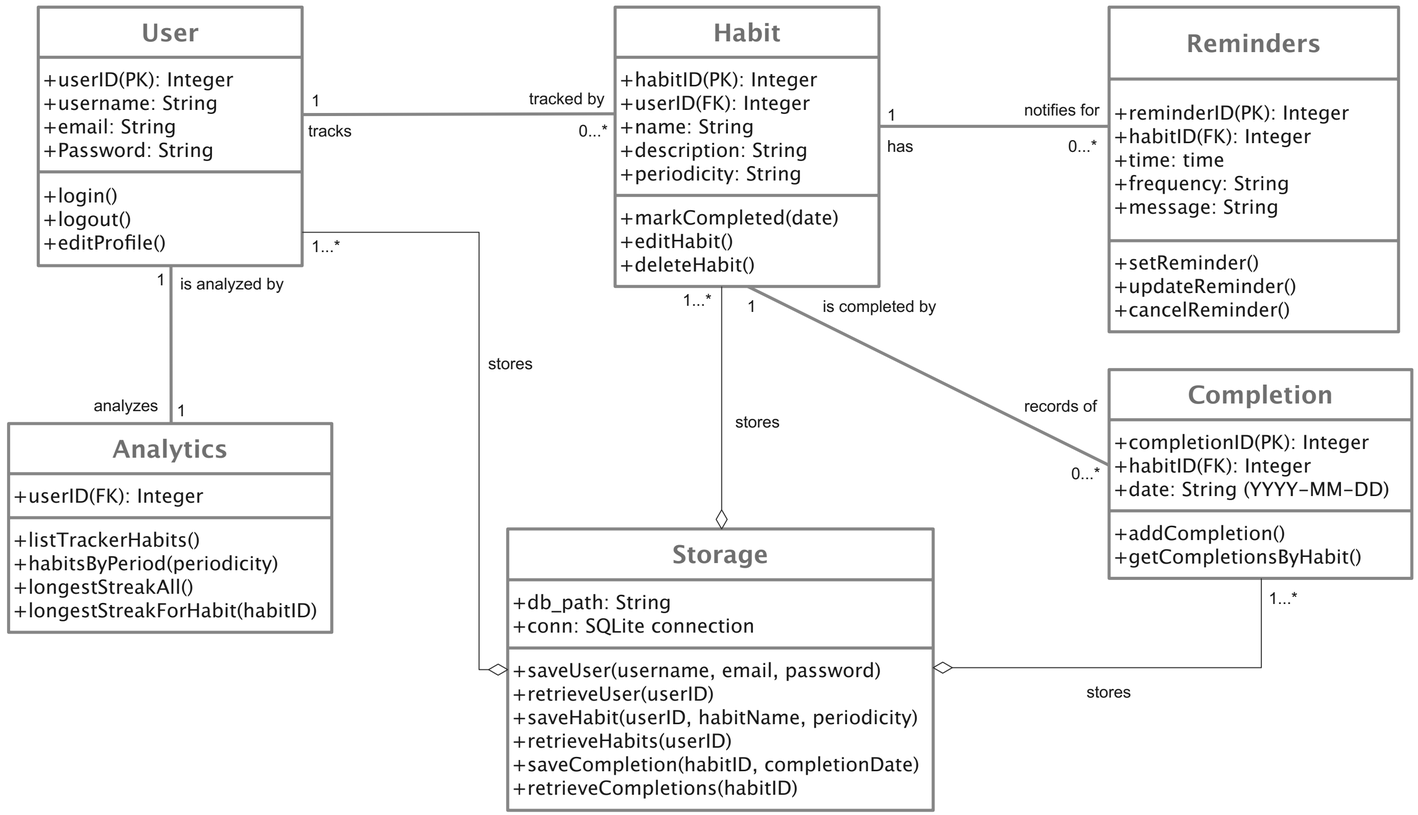
1. **Return a List of All Currently Tracked Habits**
    - Use the map function to extract the name of each habit from the list of habit dictionaries.
2. **Return a List of All Habits with the Same Periodicity**
    - Utilize the filter function to select habits that match the specified periodicity.
3. **Return the Longest Run Streak of All Defined Habits**
    - Implement a function to calculate the run streak for a single habit.
    - Use the map function to apply this calculation across all habits.
    - Use the max function to find the habit with the longest streak.
4. **Return the Longest Run Streak for a Given Habit**
    - Use the previously defined function to calculate the run streak for the specified habit.

Tools & Modules that will be used for this project

- Pycharm: PyCharm is an advanced IDE tailored for Python development. It integrates smart coding assistance, debugging, and testing features.
- SQLite3: a built-in module for SQLite database operations, allowing for more structured data storage and complex queries without needing an external database server.
- Pytest: for writing and running tests. It's a powerful testing framework that makes it easy to write tests.
- Django: web framework for developing web applications in Python. Django offers a lot of features out of the box, making it suitable for more complex applications. It also has its own ORM (Object-Relational Mapping).

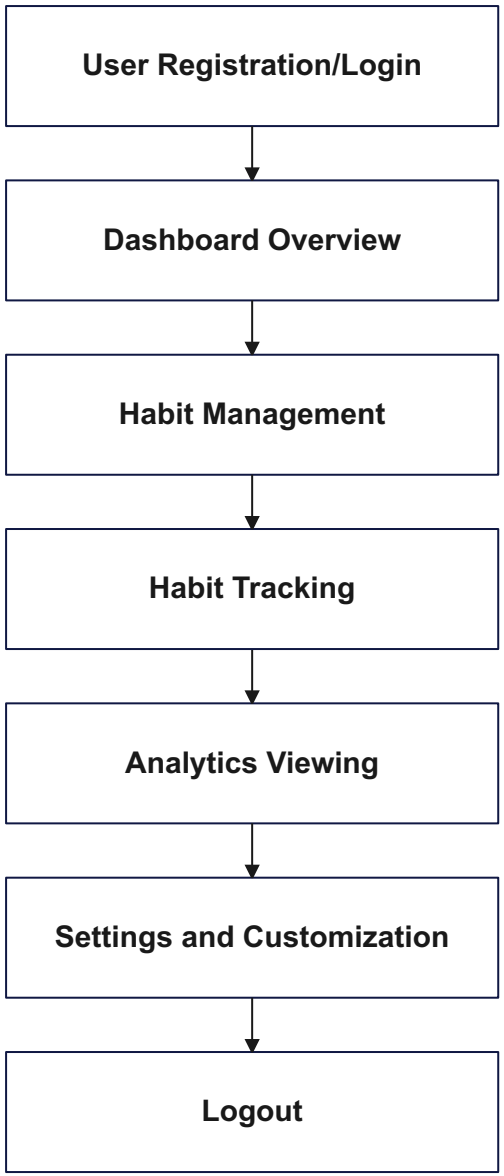
- Datetime: The datetime module in Python is crucial for handling dates and times, an essential part of a habit tracker app that tracks when habits are created, updated, or completed. It's a built-in Python module.
- Faker: used for creating “fake” data that build the foundation for testing; tool not necessarily needed for this project (as it could be quickly created manually), but good to gain some experience for more complex projects.

Habit Tracker App – Class Diagram



Testing
none (operates through methods)
+testUserCreation() +testHabitManagement() +testCompletionLogging() +testAnalyticsCalculation()

Habit Tracker App - User Flow Diagram



Description: Users start by either registering for a new account or logging into an existing account. This is the entry point into the habit tracker app.
Functionalities: Account creation, Login.

Description: After logging in, users land on the dashboard, which provides an overview of their tracked habits, daily progress, and quick access to analytics.
Functionalities: View current habits, daily completion status.

Description: Users can manage their habits (select from 5 pre-defined habits, daily or weekly). This includes adding new habits, editing existing ones, or deleting them.
Functionalities: Add, edit, delete habits; set habit periodicity and reminders.

Description: For daily engagement, users mark habits as completed. This action is crucial for tracking progress and calculating analytics.
Functionalities: Mark habits as completed, view past completions.

Description: Users access the analytics section to gain insights into their habit formation progress, such as streaks and habit adherence.
Functionalities: • View all currently tracked habits • Filter habits by periodicity • View the longest run streak of all habits • View the longest run streak for a specific habit.

Description: Users can customize their experience, adjust settings, and configure notifications to enhance their habit tracking experience.
Functionalities: Manage account settings, configure notifications, customize app behavior.

Description: Finally, users can securely log out of the app or exit, ensuring their data is saved and protected.
Functionalities: Secure logout, data persistence.