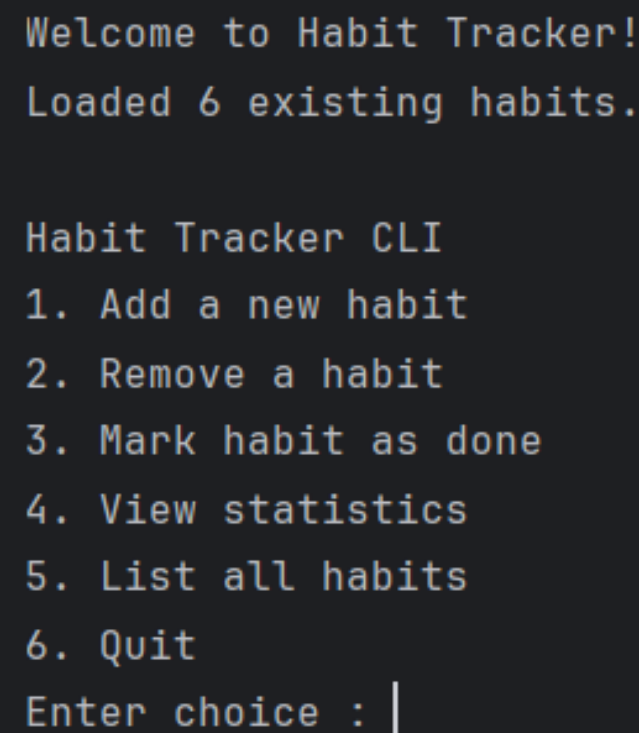


# Habit Tracker Application - Development Report

## Design and Implementation Overview

A screenshot of a terminal window showing the Habit Tracker CLI interface. The text is as follows:

```
Welcome to Habit Tracker!  
Loaded 6 existing habits.  
  
Habit Tracker CLI  
1. Add a new habit  
2. Remove a habit  
3. Mark habit as done  
4. View statistics  
5. List all habits  
6. Quit  
Enter choice : |
```

Michael Jachim

01.03.2025

Figure 1 : Screenshot of CLI interface

# Project Overview

## Purpose :

- A habit tracking application to help users build and maintain habits
- Users can create, manage and analyze habits with daily/weekly/monthly frequencies

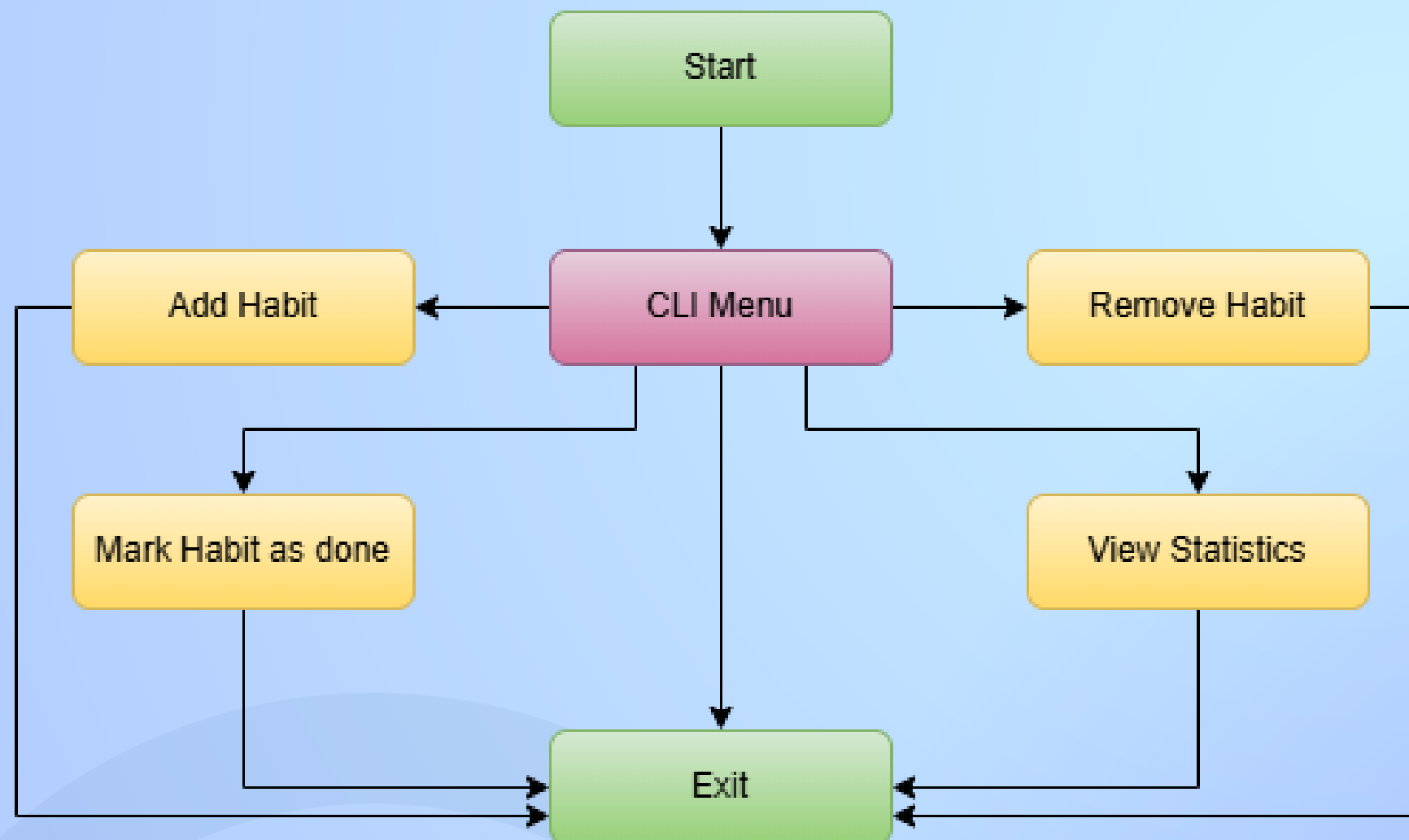


Figure 2 : Flowchart of the User-interaction with the App

## Key elements :

- Add remove and mark habits as completed
- Track streaks (current and longest) for each habit
- Track missed habits and completion history

# Tools and framework

- Programming Language : Python
- Database : SQLite (file-based database for storing habits and completions)
- Libraries : sqlite3 (for database-operations) and datetime (dates and streaks)
- Development Tools : Pycharm and GitHub



# Application Architecture

- CLI (Command Line Interface) : Handles user interaction with the App
- Database : Stores habits and completion dates
- Analytics : Calculates streaks and missed habits
- Habit Class : Represents a habit with attributes like name or frequency
- User Class : Manages a collection of habits for a specific user

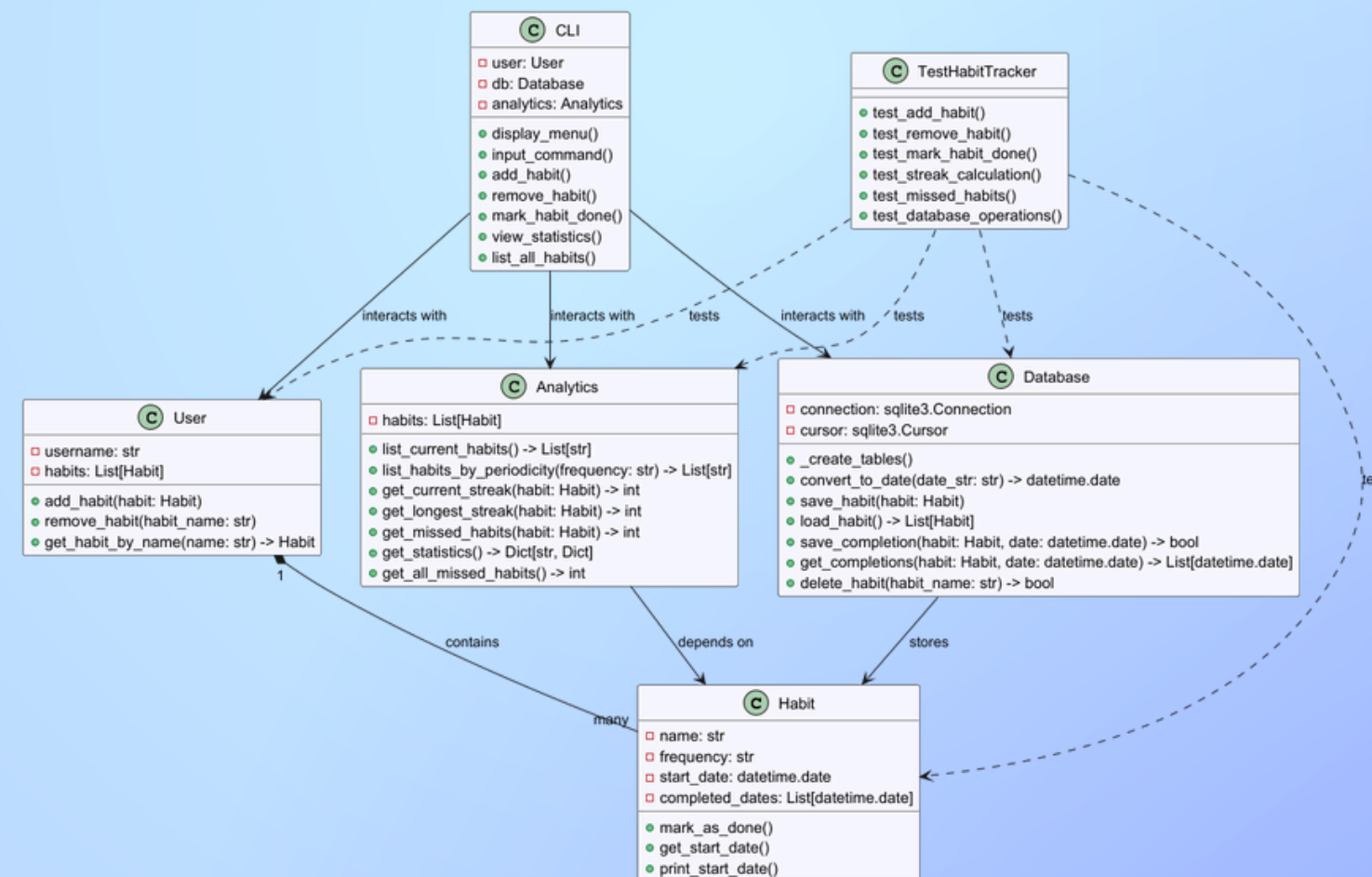


Figure 3 : UML-diagram of the fully build app



# Implementation Details

## Habit Creation :

Users can add habit with a name and frequency

## Habit Tracking :

Users can mark habits as completed, and the app tracks streaks and missed habits.

## Database :

Habits and completions are saved into an SQL database

```
# Class to represent a user, with a username and habits
class User :

    # Constructor to initialize a "User" object, "self" is a reference to the current object
    def __init__(self, username) :
        self.username = username # sets the username
        self.habits = [] # an empty list to store habits or habit objects

    # Method to add a new habit to the user's habit list
    def add_habit(self, habit) : 23 usages (1 dynamic)
        self.habits.append(habit) # adds (or appends) a new habit to the habit list

    # Method to remove a habit from the user's habit list, using the habit name
    def remove_habit(self, habit_name) : 3 usages (1 dynamic)
        # List comprehension, creating a new list filtering out the named habit
        self.habits = [habit for habit in self.habits if habit.name != habit_name]
```

Figure 4 : Commented code snippet from the User Class

```
# Database Class, to manage and store data related to the habits
class Database : 14 usages
    # Constructor to initialize a database object, database name chosen is optional
    def __init__(self, db_name = "habit_tracker.db") :
        self.connection = sqlite3.connect(db_name) # Connect to the database (or create one)
        self.cursor = self.connection.cursor() # Creates a "cursor" object, to be able to execute SQL commands
        self._create_tables() # Method to create necessary tables

    # Method to create the required tables for storing habit data
    def _create_tables(self) : 1usage
        # SQL statement, creates a "habits" table to store habit information
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS habits (
                                id INTEGER PRIMARY KEY,
                                name TEXT,
                                frequency TEXT,
                                start_date TEXT)''')

        # SQL statement, creates a "completion" table to record completions
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS completions (
                                habit_id INTEGER,
                                completion_date TEXT,
                                FOREIGN KEY (habit_id) REFERENCES habits (id))''')

        # Commit changes to the database to save the table creation
        self.connection.commit()
```

Figure 5 : Commented code snippet from the Database Class

# User Interaction

- Users interact with the App via a CLI menu
- Options include adding or removing habits, marking habits as done or viewing statistics

## Example workflow :

1.User adds a habit (e.g. “swimming”, weekly)

```
Habit Tracker CLI
1. Add a new habit
2. Remove a habit
3. Mark habit as done
4. View statistics
5. List all habits
6. Quit
Enter choice : 1
Predefined habits :
1. Studying
2. Exercise
3. Meditating
4. Chores
5. Take-A-Break
You can choose a habit from the list (type in number),
or enter your own habit : Swimming
Enter frequency (daily/weekly/monthly) : weekly
Habit 'Swimming' added successfully !
```

Figure 6 : CLI interface, adding habits

2. User marks the habit as done

```
Enter choice : 3

Current habits :
1. Studying
2. Meditating
3. Dancing
4. Chores
5. Take-A-Break
6. Swimming
Enter habit number or name : 6
Completion for habit 'Swimming' saved on 2025-03-08
Habit 'Swimming' marked as done !
```

Figure 7 : CLI interface, marking habit as done

3.User views statistics

```
Habit : Swimming
Frequency : Weekly
Start date : 2025-03-08
Current Streak : 1 weeks
Longest Streak : 1 weeks
Missed Weeks : 0
Recent completions :
- 2025-03-08
-----
```

Figure 8 : CLI interface, view statistics

# Challenges and solutions

Obviously any project comes with its challenges, and creating an app as an inexperienced programmer was no easy task.

The main challenges started with the structuring of the program, but even after deciding on a structure (several separate classes instead of one, easier to find, edit or add methods), the amount of methods needed grew over time, and with that the amount of potential problems and headaches.

A couple of the more specific problems I encountered and a potential solution :

- The biggest issue was handling dates and date calculations for streaks across different classes. One example of a solution in the Analytics Class, was importing the “datetime”-library, using datetime and timedelta for comparisons, and write a method which is looping through the completed dates (starting with the newest entry) to find out whether the last completed date falls into this day/week/month, the last one, or whether the streak has been broken.
- Implementing the CLI as a whole, not just implementing different classes into the CLI, but also using a variety of if/else-cases or exceptions to handle different or wrong choices here, or in the database (like e.g. wanting to mark a habit, which hasn't been added to the list, as completed).



# Potential future improvements and conclusion

Like with any app or product, there are a variety of improvements that could be made in the future.

- A Graphical User Interface (GUI) would improve the user experience and be more pleasant for the eyes. With that, you could also include animations, congratulating for reaching certain streaks or changing with more progress.
- The introduction of a gameplay-element might encourage people to be more motivated finishing the tasks, it could be a minigame that is being unlocked, a character e.g. telling a story or interacting with you.
- Including notifications or reminders for the user.
- Adding a variety of different statistics, using a GUI, graphs could also be included.
- Being able to export your statistics.

However I do believe, that this app lays a solid foundation, and is doing what is most important -

- Helping you to manage habits you want to keep track off, or remove them if you don't need them anymore
- Allowing you to mark them as completed
- Showing you a list of all the habits in your list
- Making it possible for the user to see whether they've been able to finish the habits on a regular basis, with the option of seeing the longest successful streak.