Habit Tracker

From app concept to open source program

Further training project of

https://www.iu-akademie.de

Project by

Björn Leue, *25.11.1985

iu akademie e-mail: bjoern.leue@iu-academy.org

personal e-mail: webmaster@wildsite.de

Last editing: 07.01.23

Contentlist

Introduction

For the further training of the IU Academy ("Python and SQL programming"), I received the order in the third module to create a habit tracker.
I should orientate myself as I would want to explain to colleagues.

Below you will find all the thoughts of how I made it for version 0.1.

Preview

I looked at the task and analyzed the necessary todo.
I decided to include an extended variant with user interface.

Tasks:

Habits should be able to be created with task specification (with period of execution).
Additional "done" field
At least 2 periods (daily, weekly)
Adjustable daily for 2 weeks
If a habit is broken, it should be pointed out
Habits should be able to be evaluated.
    Longest series
    Current daily habits
    Hardest last month

acceptance criterias:

- Python 3.7
- Installations and execution instructions(ReadMe.md, docstrings,..)
- at least one Objectorientet Class (Habits)
- at least 2 periods (weekly, monthly)
- 5 predefined habits with transparent usabillity
- saving of start/stop of habits (datetime)
- 4 weeks saving data, for *"Test Fixture"*
- frequently entered data should be saved temporarily (sqlite, cookies, json....)
- analysis module for displaying data
- API to create, delete and view
- tests shoud be in

App concept

For the implementation I thought of a Python backend according to the specifications.

This can be controlled by accepting parameters in Key=Value format.

How to call the functions:

Windowskey, then typing „cmd“, press Enter and type following text:

for example „PathToFile“: C:\xampp\htdocs\HabitTracker\Python

```
cd PathToFile
python dateiname.py action=TestEverything automatic_tests=True show_actions=True
python dateiname.py action=SignupUser user_name=YOURNAME user_password=YOURPASS
python dateiname.py action=AddHabit name=Do³some³sport!!! description='' timespan=dayly
python dateiname.py action=ShowAll
```

For this I program a main function that takes the given parameters and inserts them into a settings.json.

All set parameters for further processing are taken from this settings.json.

In the next step, the function to be carried out is selected with the "action" parameter, executed and, if required, data is reproduced (line by line or per json).

These are visually evaluated when using the web version and evaluable data via the HTML front end.

Aufbau

HabitTracker.py	Main program, sets Attributes to json and start an action
actions.py	collection of functions for running the program
analyse.py	collection of functions for analysing the data
sqlite.py	collection of functions for database connection
test_project.py	class with collection of functions for testing the whole project

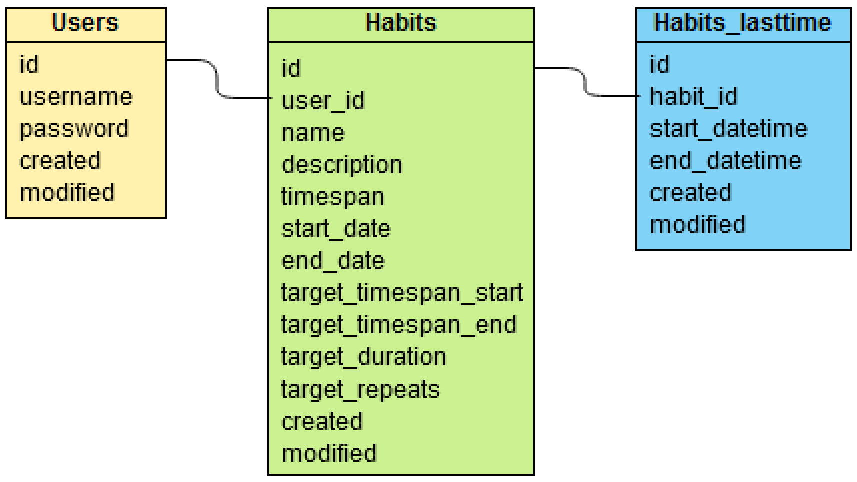
Speicherung der Daten

I use SQLite to store the data, because with local use a local database seems quite logical.

An additional implementation of Mysql would be possible.

I don't plan the basic structure very complex in the first step.
A relational structure like the following should suffice for the functionality:

A user has several habits.
A habit has multiple promotion periods



--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---

For further expansion, a game character extension would be possible.
This usually increases the point to return on apps like this significantly.

A user has several habits, if you stick to them you get points.
From a certain number of points in the areas daily, weekly, .. there are trophies to collect.

users → user_gamepoints → winnings

DBMS-UML

<u>**Users**</u>	
id	ID of users (autoincrement)
name	Username of user
password	Password of user
created	Datetime this row was created
modified	Timestamp of changing

<u>**Habits**</u>	
id	ID of habit
user_id	ID of the user who created this row
name	Name of habit
description	Descriotion or subtext of habit
timespan	timespan (daily, weekly, monthly, yearly)
start_date	Date when the tracking starts
end_date	Date when the tracking ends
target_datetime_start	Planned start time
target_datetime_end	Planned maximum time to exit
target_duration	Planned duration of the habit
target_repeats	Planned repeats of habit
created	Datetime the habit was created
modified	Timestamp of changings

<u>**Habits_lasttime**</u>	
id	ID of the activity
habit_id	ID of related habit
start_datetime	Real time of start
end_datetime	Real time of completition
created	Datetime when the activity was created
modified	Timestamp of changing

Mögliche Spielcharakterliche Erweiterungen (**ab Version 2**)

User_gamepoints	Table for storing points
id	
user_id	
gamepoints_id	
created	

Winnings	Table of points awarded for streaks kept
id	
name	
description points	
image_url	

HTML user-interface (PHP wird benötigt)

The variable names must all be entered correctly for the Python script.
Because I find this a bit uncomfortable, I will create a web frontend where the data can be entered in input fields.

The fields are recorded via Javascript and sent to a PHP script that calls the Python backend script with parameters.

From the HTML user interface (for usability of the transfer to the command line) the comma's are replaced with --comma-- and spaces with the math cubic (³) character.
This change in the given text is reset in the Python backend.

Html / Javascript ↔ PHP Script ↔ Python Programm ↔ DBMS (MariaDB or SQLite3)

HTML Seiten

index.html	Entry point, redirect to login.html
login.html	Login / signup form, data evaluation by Python
start.html	Basic HTML structure for templates, loaded by selecting the function given in the frontend
Templates	HTML snippets with required input fields to control the Python functions