Marthinus Maree 92122115

Habit Tracking application

Development Phase

Habit Tracking Application

Table of contents

[Introduction 2](#_Toc104921586)

[Overview 2](#_Toc104921587)

[Definitions 2](#_Toc104921588)

[Project Overview 2](#_Toc104921589)

[User Interaction 2](#_Toc104921590)

[**Command line interface (CLI)** 2](#_Toc104921591)

[**Swagger UI** 4](#_Toc104921592)

[Project Structure 7](#_Toc104921593)

[Technology choices 9](#_Toc104921594)

[Bibliography 9](#_Toc104921595)

# Introduction

Most people want to stop unhealthy habits and create good habits in its place. To achieve this, they turn to technology for assistance. I want to create a habit tracking application to assist them to achieve their goals.

## Overview

I used Python version 3.10.3 with ***FastAPI*** and ***click*** to create a backend for my habit tracking application.

## Definitions

1. ***Habit:*** Regular activity or practice that you want to create e.g. *Go running every day*
2. ***Completed Habit:*** An activity or practice that you have completed at a specific time e.g., Went for a run at 1pm today.
3. ***Tracked Habit:*** A habit that was completed at least once is considered a tracked habit.

# Project Overview

I created a backend with a command line interface and a Swagger UI to simplify user interaction. I have limited data input and some other features to the Swagger UI for simplicity. One could also use tools like Postman to interact with the REST API.

For testing I have included functions to seed (loading data) and delete (purge all data excluding frequencies) testing data.

## User Interaction

All the commands listed below should be run from the command line in the ***habits\_backend*** folder of the cloned project.

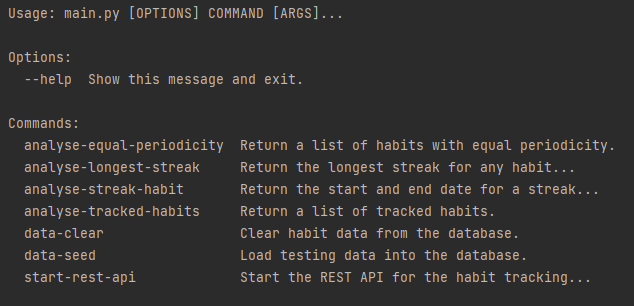
### **Command line interface (CLI)**

#### Help Menu

To load a list of all available commands, run the following command:

python main.py –help

Output:



#### Load Data

To load testing data into the database run the following command:

python main.py data-seed

Output:

Seeded data

#### Clear Data

To clear the database of all habits and completed habits run the following command:

python main.py data-clear

Output:

Database Cleared

#### Get all tracked habits

The following command will return a list of all tracked habits:

python main.py analyse-tracked-habits

Output:

Tracked Habits:

[{'id': 1, 'name': 'Running', 'repeated': 'Daily', 'count': 15}, {'id': 2, 'name': 'Meditation', 'repeated': 'Daily', 'count': 8}, {'id': 3, 'name': 'Family', 'repeated': 'Weekly', 'count': 4}]

#### Get habits with the same period (equal periodicity)

To get habits with the same period (equal periodicity), run the following command:

python analyse-equal-periodicity --frequency daily

Output:

Equal Periodicity:

[{'id': 1, 'name': 'Running', 'repeated': 'Daily', 'count': 15}, {'id': 2, 'name': 'Meditation', 'repeated': 'Daily', 'count': 8}]

#### Habit with the longest run streak

Run the following command to display which habit has the longest run streak:

python main.py analyse-longest-streak

Output:

Longest Streak any habit:

{'start': datetime.datetime(2022, 4, 27, 14, 17, 45), 'end': datetime.datetime(2022, 5, 3, 15, 57, 21), 'cnt': 8, 'habit\_id': 2}

#### Longest streak for a habit

To display the longest streak for a habit:

python main.py analyse-streak-habit --habit\_id 1

Output:

Streak Details:

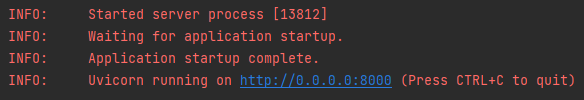
{'start': datetime.datetime(2022, 4, 27, 14, 17, 45), 'end': datetime.datetime(2022, 5, 2, 15, 57, 21), 'cnt': 6}

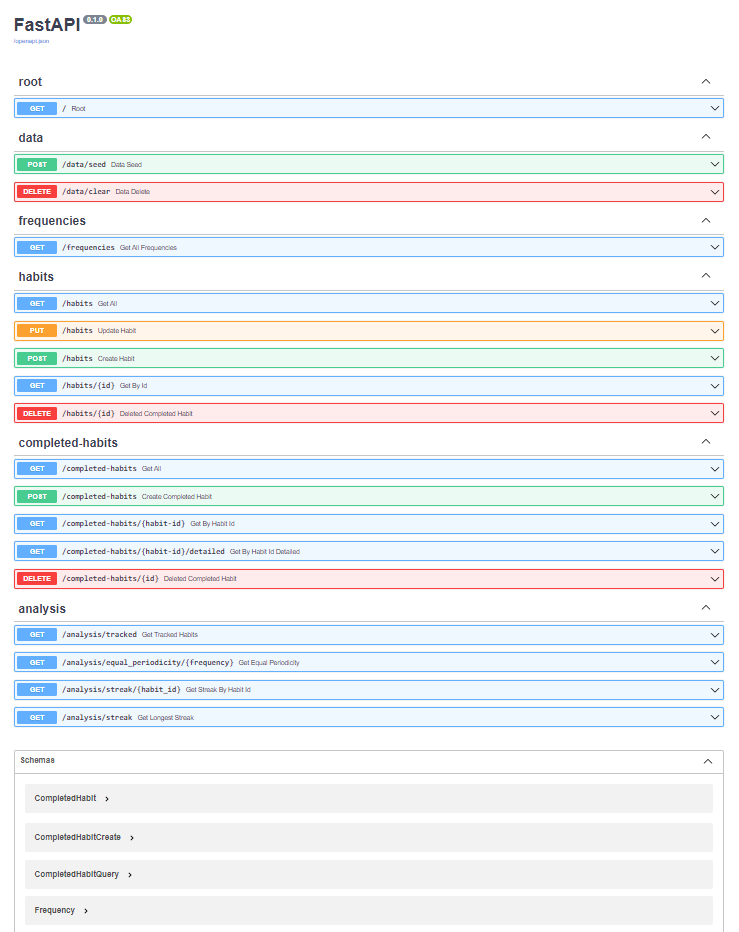
### **Swagger UI**

To start the Swagger UI for a visual and interactive interface run the following command:

python main.py start-rest-api

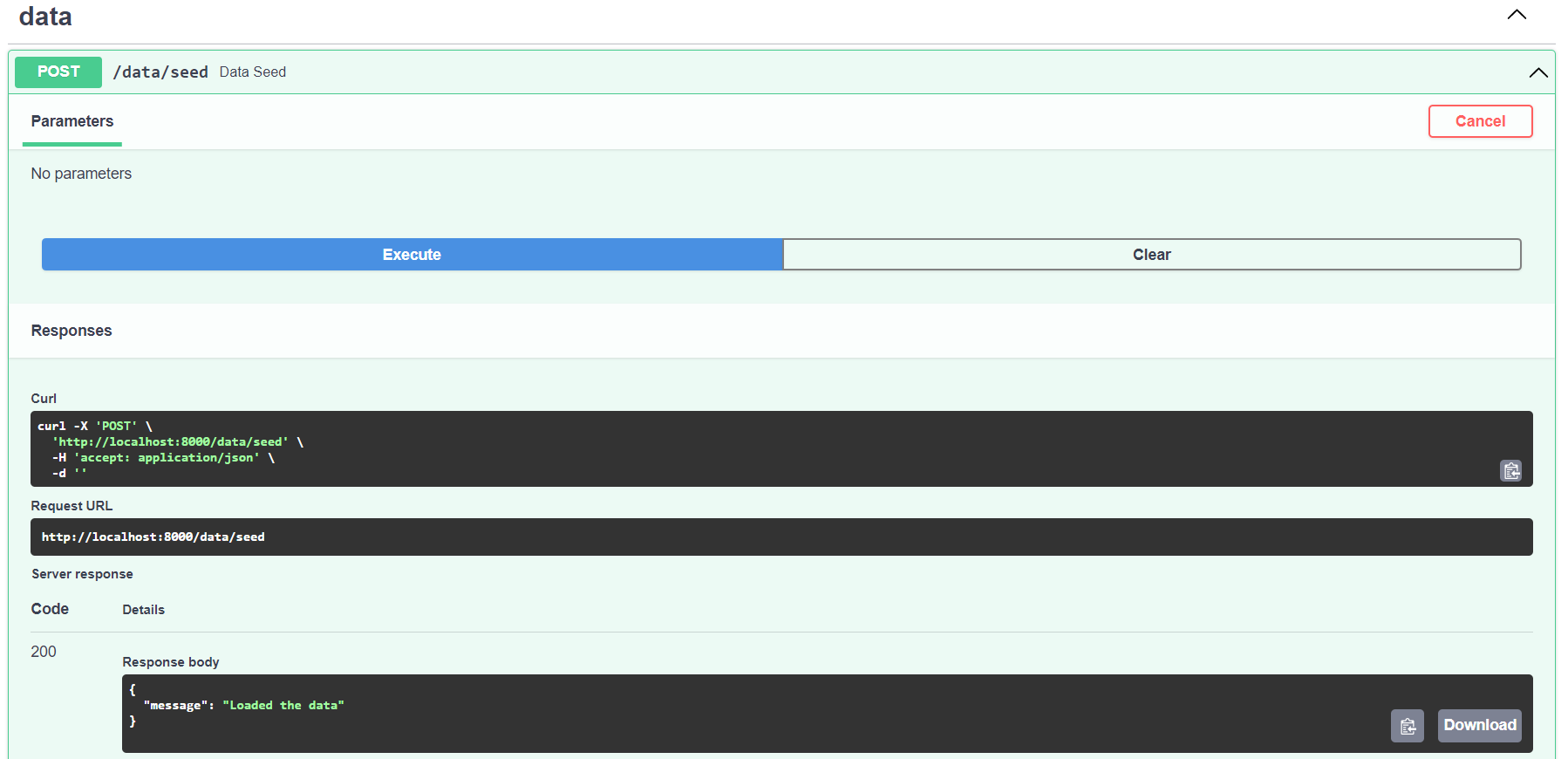
Output:



Then if you browse to the <http://localhost:8000/> URL in your browser, you should see the following Swagger UI interface which will allow you to interact with the API’s resources, inspect the paths and see a color-coded representation of the HTTP verbs. This is an interactive interface which allows you to call the endpoints: 

#### Data (Assist with testing)

We have URLs for loading and clearing the data. Normally this would not be part of the REST API but have been added here to assist with testing. This allows for loading test data or clearing the database. It will not delete the default frequencies. See the example below:



#### Frequencies

We have pre-loaded three default frequencies to indicate how often a habit is repeated. For example, a habit must be repeated daily:

Shape

Description automatically generated with low confidenceGraphical user interface, application

Description automatically generated

#### Habits

The habit URLs are used to query, create, and delete habits:

Background pattern

Description automatically generated with low confidence

#### Completed Habits

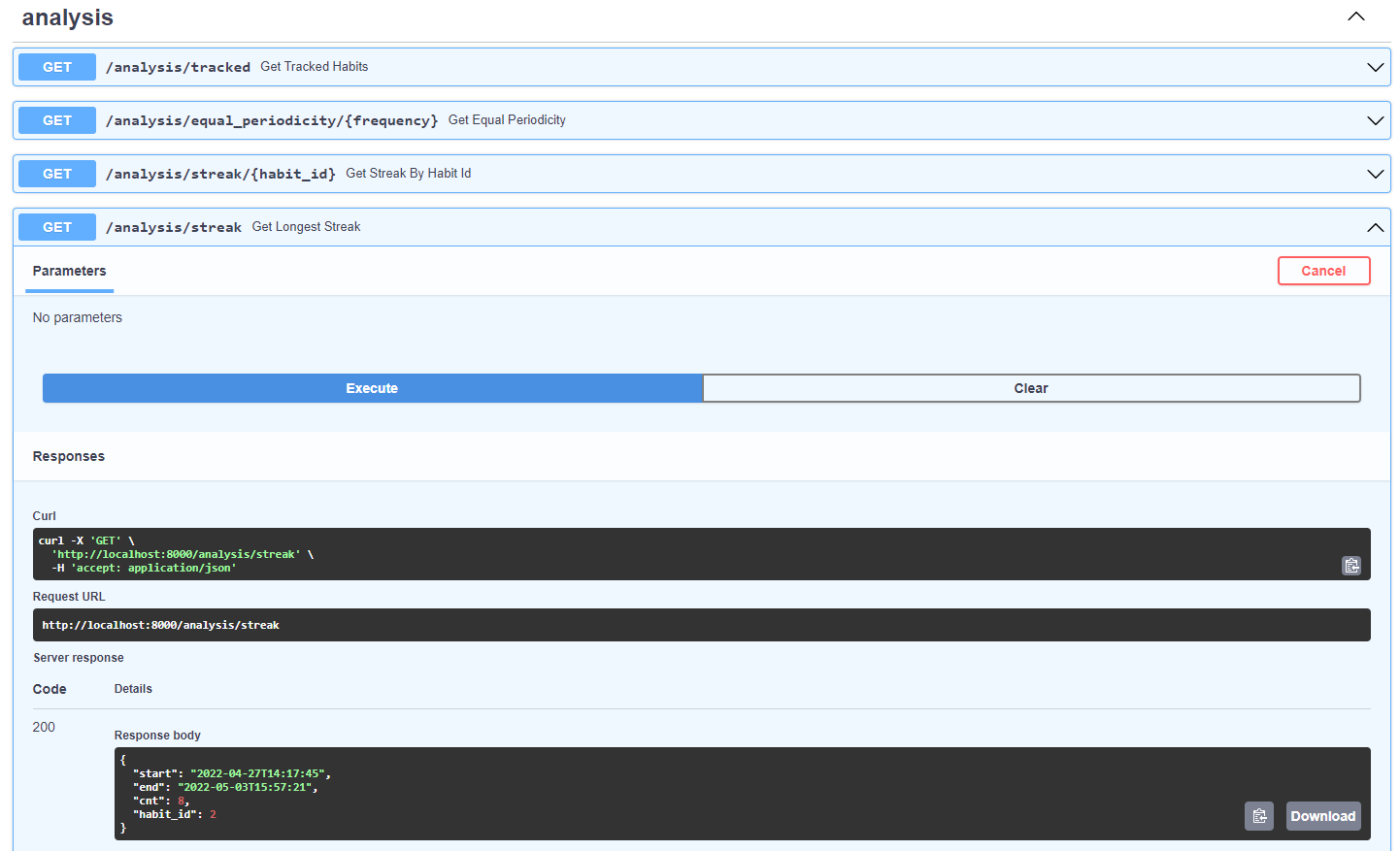
The completed habit URLs are used to query and track when a habit has been completed. The completed habits are used for the analysis and to calculate the running streaks:

Graphical user interface, application

Description automatically generated

#### Analysis

The analysis URLs are used to query information like which habits are tracked, the longest running streak, etc.:



## Project Structure

The main entry point for the application is through main.py module. This module uses the [click](https://click.palletsprojects.com/en/8.1.x/) package to create a command line interface:

Graphical user interface

Description automatically generated with medium confidence

The user will interact with the CLI or the REST API. The interface will call the relevant service which in turn would call the crud functions. This is based on an [Clean Architecture approach](https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html).

The ***pydantic*** schemas are used for data parsing and validation. In turn the ***SQLAlchemy*** models are used for parsing the data for the database. I have added the functional programming module to the modules folder.

I used docstrings and inline comments to document my code:

Text

Description automatically generated

I used ***pytest*** to test the functionality of my backend:

Text

Description automatically generated

## Technology choices

* ***Python version 3.10.3*** – Project requirement to use 3.7 or later.
* ***PyCharm 2022.1.1*** - Popular IDE / source-code editor that runs on Windows, Linux and macOS. I found that it works better than Visual Studio Code for python development.
* ***sqlite3*** – It is a library that provides lightweight disk-based database to persist the data.
* ***pytest*** – Framework for writing tests.
* ***FastAPI*** – Framework for building APIs with python. This will provide an alternative for the CLI.
* ***click*** – Python library for creating command line interfaces.
* ***Pylint*** – Linting tool that checks for coding errors and enforce coding standards.
* ***Swagger UI*** – Interactive exploration to call and test your API from the browser.
* ***SQLAlchemy –*** Accessing data stored in the database.
* ***pydantic -*** Data parsing and validation.

## Bibliography

1. n.a. (n.d). Google Python Style Guide – Naming <https://google.github.io/styleguide/pyguide.html#316-naming>
2. Lutz, M (2013). OOP: The Big Picture, Learning Python 5th Edition. O’Reilly
3. Marcus, S (2021, July 16). Test Driven Development with pytest <https://stackabuse.com/test-driven-development-with-pytest/>
4. n.a. (n.d). [Zalando RESTful API and Event Guidelines](https://opensource.zalando.com/restful-api-guidelines/#_zalando_restful_api_and_event_guidelines)

<https://opensource.zalando.com/restful-api-guidelines/>

1. n.a. (n.d). [FastAPI](https://opensource.zalando.com/restful-api-guidelines/#_zalando_restful_api_and_event_guidelines) - SQL (Relational) Databases

<https://fastapi.tiangolo.com/tutorial/sql-databases/>

1. n.a. (n.d). click

<https://click.palletsprojects.com/en/8.1.x/>

1. Martin, Robert C (2012, August 13) – The Clean Architecture

<https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>