Author

Hitansh Shah
21f1001178
21f1001178@student.onlinedegree.iitm.ac.in
Hello, I am Hitansh Shah, an avid chess player and a formula 1 enthusiast. I have been fond of puzzles since I was 5, which is when I picked up chess. Programming thus, has been a natural extension of this inclination towards the search of constant brain fodder and is a field I look forward to exploring

Description

In this project we were supposed to create a “Quantified Self” App. The terminology is quite apt. Quantified means to assign some objectivity to an otherwise vague term. One could describe this assignment as being one of subjectivizing oneself. One might want to track themselves in more than one way. Thus it creates the demand for such an app that allows one to track themselves without bias, across multiple spheres of life and in more ways than one. This Quantified Self App would allow the user to create multiple trackers and then log values in those trackers as they deemed fit. The app would then collect those values and display it in a human readable format while also performing basic analysis such as trend lines that adds value to the user.

Technologies used

- Vanilla Flask for the majority of the backend. Flask with its out of the box components was found to be sufficient in most use cases.
- Flask-JWT: A token based out-of-the-box authentication system was sorely missed but Flask-JWT extension more than makes up for it. It is easily integrated into the application and is quite secure
- SQLite3: SQLite3 was chosen as the DB due to its high portability and ease of use while still maintaining similar levels of performance as other heavier databases.
- peewee ORM: peewee was chosen as the ORM for its ease of use, and versatility over SQLAlchemy and a host of other extensions such as the playhouse utils
- VueJS. Front end code was written with VueJS as the UI Framework, with vanilla JS without webpack.
- Vue Flow Form: An extension to Vue3. This was used to design forms in the app.
- Tailwind CSS: All styling was done using either native CSS or Tailwind CSS.
- Chart.js: An open source library that was used to create the charts and graphs.

DB Schema Design

The database is broadly broken down into three tables,

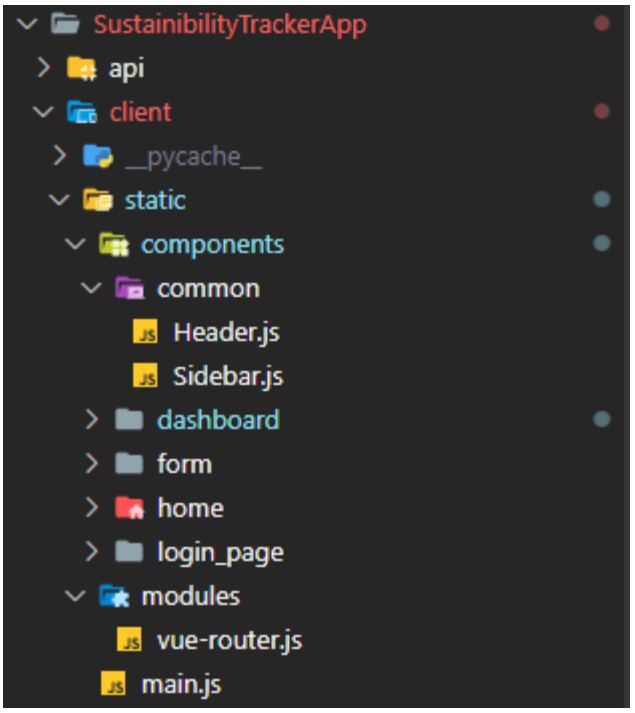
- User
- Tracker
- Logs

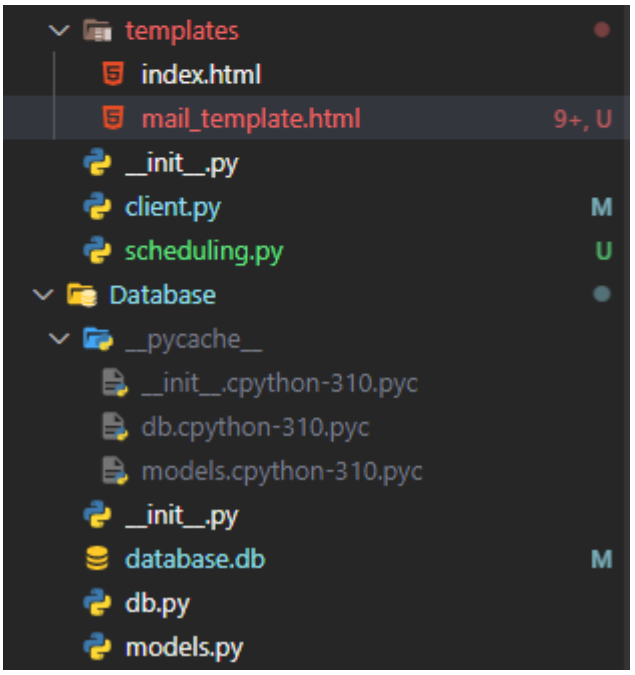
With each of User and Tracker and Tracker and Logs having a 1-N relationship.
This was done in order to reduce. The shape of the information stored in the tracker varied quite a lot depending on the type and data stored in the tracker. This proves quite challenging to model, thus resorting to a JSON based model for the tracker that was stored in the BLOB format along with the primary key and User’s foreign key.

API Design

The API was separated into two major parts, the routing and logic handling, and the database handling. The API is written in Python using the Flask microframework. The API tries to incorporate the principles of RESTfulness and is for the most part a complete REST API barring a few exceptions. All but the login and signup endpoints are protected by the Token based authentication system.

Architecture and Features





The above images show the file structure of this project. The Application was first bifurcated into two parts The Database modules and the Front End Modules. The client.py houses the central controlling logic of the application. The models.py file contains the necessary database models. Other front end files are a part of the static directory. This directory is also the Flask static directory, i.e this is the point Flask refers to when it finds links in the webpage. The static directory houses the component directory where all the vue components are stored. Components are divided into common components such as the navbar and sidebar. Each subdirectory of the component directory is a major screen/ functionality of the application, such as the home page, the dashboards etc.