Author

Navya Ladi
21f1006997
21f1006997@student.onlinedegree.iitm.ac.in

I have done my graduation, B.Tech in Computer Science stream and I am working on improving my technical skills.

Description

In the Quantified self app, we have to create an application in which users can create and login into their accounts and track their needs using the Add Tracker and Log events for every tracker. Users should be able to Create, Read, Update and Delete trackers and should be able to Create,Read,Update and Delete their logs for every tracker that they have. Users should be able to see the Visualized data of their logs.

Technologies used

- Flask, Flask-SQLALCHEMY for the application
- Flask Restful API's for API
- HTML,JINJA2 statements in HTML & CSS, Bootstrap
- Matplotlib for Data Visualization
- Flask Login

DB Schema Design

I have used three tables.

user_master :	**tracker_master:**	**log_master**:
● user_id - Integer, Primary Key ● user_name - String, Unique, Not Null ● user_email - String, Unique, Not Null ● user_pwd - String, Not Null ● sec_question - String, Not Null ● sec_answer - String, Not Null ● created_date - String modified_date - String ● logout_time - String	● tracker_id - Integer, Primary Key ● name - String, Not Null ● description - String, Not Null ● type - String, Not Null ● settings - String ● chart_type - String ● created_date - String ● modified_date - db.String ● user_id - Integer, Foreign Key from user_master	● log_id - Integer, Primary Key ● log_time - String, Not Null ● value - String, Not Null ● notes - String ● created_date - String ● modified_date - String ● selected_choice - String ● user_id - Integer, ForeignKey from user_master ● tracker_id - Integer, Foreign from tracker_master

I have used one to many relationships between user_master and tracker_master because a user can have multiple trackers & one to many relationships between user_master and log_master because a user can have multiple logs.

API Design

- I have used flask-restful API's to implement api. Used reqparse to read data from the request and marshal_with output fields format in response.
- GET, POST,PUT apis for User
- GET api for Dashboard data
- GET,POST,PUT,DELETE apis for Tracker
- GET,POST,PUT,DELETE apis for Logs

File name in the Project directory : quantified_self.yaml

Architecture

- app.py has the code for initializing and running the application.
- local_setup.sh has the script for setting up the local environment.
- local_run.sh has the script for running the application.
- db_directory folder has quantified_self1.sqlite3 database file.
- The templates folder has the html templates.
- static folder has two folders:
  - img folder has logo.png (logo of the application) and the trendlines will be saved here
  - bootstrap folder
    - css folder has all the styling code for html templates
- application folder has the below:
  - Initialized database in database.py
  - Implemented database models in model.py
  - All the application controllers are implemented in controllers.py
  - Configurations are implemented in config.py
  - Implemented the apis in api.py
  - Implemented custom exceptions in validations.py

Features

- Upon launching the application users can see the home page which has brief information about the application.
- Users can sign up for the application in the signup page and login to from Login Page.
- Once the users login they will be able to see the Dashboard page where they can see an option to Add Tracker or list of trackers if the user already has created.
- Upon clicking on the tracker user can view the tracker page in which they can see the trendline graph and list of logs.
- Users can add a tracker of type Timestamp,Numeric and Multiple Choice. Users can choose the type of chart to be displayed in the Tracker page. Users can log an event for the tracker from the dashboard page or the tracker page.
- Users can edit their Profile data using My Profile link.

  Note: Current State : Users can do all the above, I am working on the CSS part.

Video

I will upload a video in the final submission.