Author

Name:- Shivansh Jain
Roll. No: - 21f2001305
Student Id:- 21f2001305@ds.study.iitm.ac.in

I am a working professional. I am passionate about my work. I like to complete my assignment in an organized manner

Description

Our assignment is to build a social media platform. Where Users can post, upload images and interact with each other using likes, and comments. We implemented necessary features like post-control, user profile, and Admin user Access.

Technologies used

We are using the following technologies in our project

1. Flask: - Flask is a lightweight web application framework, which we are using in our project.
2. Flask-RESTful: - It is an extension of the flask framework which allows implementing REST ('Representational State Transfer'), we are using it to implement our Post API implementation.
3. Flask-session: - Flask-session is another extension of Flask which helps us to maintain the client state on the server side.
4. Flask-SQLAlchemy: - It is a wrapper over SQLAlchemy that deals with SQLite3 Database.
5. Jinja2: - Jinja is used for user report generation.
6. Markup Safe: - Markup safe implements a text that escapes characters so it is safe to use in the HTML and XML. We are using it to make blog content markup safe.
7. Vue CLI: - Vue CLI is used for User Interface development and batter scalable application development.

DB Schema Design

Please Refer to the image "'blog lite.png"' file for my Database Implementation.
I am storing data in multiple tables to reduce the redundancy as much as possible.
For example

User related schema: -

- **Accounts**: - For maintaining the user_details and user_id referenced by user_id_pwd.
- **following**: - maintaining the user_id and its following candidates.
- **followers**: - maintaining the user_id and its associated followers.

Post Related Schema: -

- **Post_id**: - For maintaining the relationship between user and post this will hold One-Many relations, i.e. one user can have multiple-posts but one post can have only one user.
- **Post_caption:** - To store the actual data of a blog. I.e Post_id, title, caption(blogs), and image_url(to locate and display the image).
- **likes:** - It will store number of likes and dislikes(aliased as Flags in the project) and post_comment_id.
- **comment_table**: - it will store each and every comment of a post, and can be retrieved via a unique Post_Comment_id.

Methodology: -

The main reason behind this approach is to minimize the redundancy and try to eliminate as many Database anomalies as possible.

To reduce the number of joins we have created a few extra schemas like post_interaction & user_post_and_follower_info to reduce number and save compute overhead.

Backbone of the entire schema is user_id_password. Because every other schema is directly or indirectly connected with it.

In case of post management, we have divided it into further smaller components.

Post_id schema is essentially establishing a connection between user_id and post_id. This post_id is further referencing post_details and post_interaction.

In post_interaction we are generating one more **unique post_comment_id** which is referencing a post_comment_table to store comments for the same.

API Design

I have implemented Get, Put, Delete and Post for Users table as Users_api class, Posts table as Posts_api class, Follower table as Followers_api class. Username has been set as primary key in the Users table.

Architecture and Features

In this project I am strictly using Single Responsibility Principle(SRP) which states that single module single responsibility, I have divided the entire application into multiple modules.

Model(Which you can find inside model folder) contains three files **model.py, user_model_controller.py and post_model_controller.py**. Model contains all table and column details with SQLAlchemy operations. User_model_controller contains one class **UserModelManager** is responsible for every user related operation like add, delete and follow/following operation.

Similarly in case on Posts, post_model_controller contains one class **PostModelManger** is responsible for post related operations.

Video link click