Project Report

Name: Arvind Raghavendran
Roll Number: 21f1005301
Email ID: 21f1005301@ds.study.iitm.ac.in

About Me:

I am a Data Science Master's student from CMI Chennai, having done his Bachelor's in Mathematics from Indian Statistical Institute, Bangalore. I have had a good working experience with programming in Python, and had successfully created the app in MAD-1. This is my second experience in creating a full-fledged app.

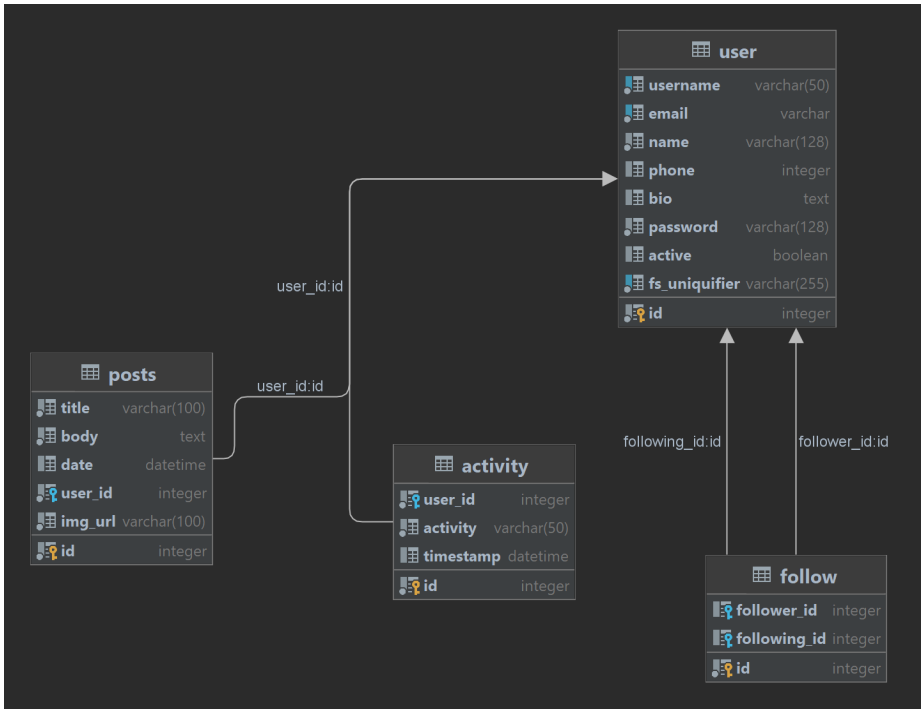
Description

The aim of the project was to build a blogging app that works very similar to that of common sites found today like Blogger. We were assigned the task of building both the frontend and the backend of the app; the former using VueJS styled with CSS, and the latter using Python and its various functionalities. The user should be able to view any/all of his posts and edit/delete them at their will. They should also be able to follow/unfollow any other blogger. The user should be notified periodically if they haven't logged/posted in the day and also a monthly report of their progress on the app. They should be able to export their blogs as CSV and view some basic stats about their account like number of follows, unfollows, logins, etc.

Technologies used

A variety of Python extensions and modules were used to build the backend of the app. Flask, Flask-RESTful, Flask-CORS, Flask-Security, SQLAlchemy, Matplotlib, Celery and Flask-Cache are a few to name. A complete list of the packages and their versions can be found in the requirements.txt file in the Project directory. Authentication was token based. The frontend was built using Vue-CLI, and is served on another port. JS was used at the frontend to fetch data from the backend. The entire app was built locally on PyCharm, and DB Browser for SQLite3 was used to build the database. Styling and aesthetics were done using CSS from Bootstrap for each of the components. MailHog was used to set up a fake SMTP server to send emails. Redis was used as both the message broker for Celery and to cache the APIs. The app is served on a browser free interface using electron.js

DB Schema Design



There are 4 main tables in the database: User, Follow, Posts and Activity.

User keeps track of all the users who have logged in to the app, with details including id, username, fullname, email, password and bio. Additionally, it also has a column called active that is set to True when they log in and a fs_uniquifier that is set using their email. The latter is used to generate the auth token. Email, username and fullname are VARCHAR and bio is TEXT with the UNIQUE constraint enforced on email and username. The password column is also VARCHAR, and it stores an SHA-256 encoding of the password using bcrypt. Id is a non-nullable, auto increment column which is the primary key of this table.

Posts keeps track of all the posts logged in by different users. It has columns id (INTEGER NOT NULL), title, img_url (all VARCHAR), body (TEXT) and date (DATETIME). There is a foreign key

link to the User, where it references the user_id. This helps us track which user posted which post. The primary key of this table is id.

Activity keeps track of different activities logged in the app. It has a foreign key to User table via user_id, activity (VARCHAR) and timestamp (DATETIME). The activity can be either of 'logged_in', 'visited_post', 'visited_profile', 'followed' and 'unfollowed'. These would be used later to send monthly reports to the users.

Follow keeps track of who follows whom in the app. It has columns id (INTEGER NOT NULL), follower_id and following_id (both INTEGER). There are 2 foreign key references, both to user.id. The convention is that the user with follower_id follows the user with following_id.

All the database schemas are stored in models.py file. It was a very intuitive, simple design for the database. The foreign key constraints had to be imposed to ensure that multiple trackers were still identified by their users, and similarly for the logs by their trackers. In addition to these, we also set up decorator classes for the APIs using Marshmallow to serialize SQLAlchemy objects.

Architecture and Features

The file hierarchy of the project is very similar to what was explained by Thejesh Sir. There is a config.py file for the configuration settings, an init .py file for the initialization of the app, apis, cache, marshmallow and login features. There is an api.py file that takes care of all the APIs. A separate file called database.py was made to initialize the database via SQLAlchemy, and as mentioned earlier, models.py has all the data related to the database schema. To handle the celery tasks, there is a workers.py that defines the worker class, and tasks.py along with the definitions of the various tasks. These 7 files along with the database itself are stored in a folder called backend. The vue components are in a folder called components. The templates folder contains the various HTML templates needed for the body of the emails and the pdfs while the assets folder contains the pdfs, csvs, post images and graphs in their respective subdirectories. Along with main.js, App.vue, routers.js and store.js, all the above are present in the folder src within the folder frontend. Along with these folders, there is an API.yaml, README.md, requirements.txt file, Welcome.pdf and main.py in the root directory. There are also 4 shell scripts, information about which are mentioned in README.md.

The main features of the application as mentioned before are adding/deleting blogs, and following/unfollowing other users. The user will receive a welcome email after signing up. The dashboard displays the list of all the user's blogs, and the homepage displays all the blogs in the app in decreasing order of timestamp. There is a feature to search for the usernames and navigate to profiles, where they can follow/unfollow a user. There is an export button in the user's profile where they will get their blog details in an email. They can view their activities in the activity tab. Monthly reports would be sent comparing the last 2 months' activities of the user. They would receive an email every morning at 10 if they haven't logged in/posted reminding them to do so. The user can also import blog details as a csv to post in bulk. The posts in the homepage and profile page are cached one minute.

Demonstration Video

https://drive.google.com/file/d/1Q6ulDezIhRisKDytuf3atljzxNHHsu5q/view?usp=sharing