MAD-1 Project Report: Blog Lite Application

Author

Name: Sanjana Mohan
Roll no.: 22f1001400
Student Email: 22f1001400@student.onlinedegree.iitm.ac.in
About me: I am a 20-year-old living in Mumbai and I've just completed my third term of the IITM degree in programming and data science. I have a passion for coding and math, but am also a finance student doing a degree at NMIMS University in Mumbai.

Features

Blog Lite is an application for uploading and sharing images, as well as browsing content created by other users. The app has been coded using the flask framework in python. The following are the key features of the app:

- Login and register system to authenticate users
- Create and edit blogs with images that show up on a personal user feed
- Follow other users and keep up with them. Home feed shows you all the recent posts by the people you follow
- Like and comment on other users' posts to show your appreciation and start a conversation
- The app is fully responsive and works on devices with any aspect ratio, including mobiles!

Technologies used

Backend: The application was coded in python, with the primary framework used being Flask, which is a framework mainly for web applications. Flask also uses Jinja to render HTML templates. Apart from this, Flask login, Flask Bcrypt, Flask RESTful, Flask SQLAlchemy, Flask Uploaded (Reuploaded), Flask WT Forms were used. For more details on these extensions, refer to the requirements.txt file.

Frontend: HTML, CSS and JavaScript were used in developing the frontend. The templating engine used was Jinja. Bootstrap 5 was also used as a frontend framework.

App Structure

The file structure of the app is as follows:

1. **Application folder:** Contains all the python files. They are described briefly below:
    a. **Api:** Contains the API objects coded using Flask Restful for all the API routes described in the Open API YAML.
    b. **Config:** Contains app configuration objects
    c. **Controllers:** Contains all the main routes for the website
    d. **Database:** Establishes connection to the database and creates the "db" object
    e. **Forms:** Contains the form objects created using WT-Forms.
    f. **Models:** Contains the flask-sqlalchemy models that enable easy queries to the database.
    g. **Validation:** Contains custom errors defined for the API
2. **Static Folder:** Contains all the static content such as images, CSS files, and also contains bootstrap.
3. **Templates:** Contains HTML templates to be served to the user by the server
4. **BlogLite_API.yaml:** Documentation in the form of YAML for the API.
5. **Main.py:** Python file that needs to be run to initialize and run the app.
6. **Requirements.txt:** Text documents containing the python dependencies required for the app.
7. **README.txt:** Contains details on how to run the app.

Models

The database management is done using Flask's SQL-Alchemy which maps the SQL tables to python objects. The following models describe how the data of the app is being stored, and forms the base of the application:

1. **User:** This is the model that stores the details of each user. It has the following fields:
    a. **Id:** A unique number that identifies the user and forms the primary key of the table
    b. **Username:** The name given to the account by the user. It has to be unique.
    c. **Password:** Used to authenticate the user, and is stored in a hashed format in the database.
    d. **Email**
    e. **About:** Short description of the user that is shown on their profile
    f. **Blogs:** List of blogs created by the user. It is derived from a relationship with the "Blogs" model.
    g. **Followers:** List of followers of the user. It is derived from a relationship with the "User" model itself
    h. **Likes:** List of posts liked by the user. It is derived from a relationship with the "Blogs' model.
2. **Blogs:** This is the model that stores the information about the posts or blogs created by the users. It has the following attributes:
    a. **Id:** Unique identifier of the blog entity
    b. **Title:** User entered title of the blog
    c. **Caption:** Description of the blog entered by the user
    d. **Image URL:** Stores the filename of the image as it is stored in the server. The images are stored in the /static/images folder, from where they can be accessed using the filename stored in the database.
    e. **Time Stamp:** Date and time when the blog was first created
    f. **User id:** Foreign key that refers to the "Id" attribute of the "User" object. It relates each blog object with one user object, forming a one-to-many relationship between blogs and users.
    g. **Liked_by:** List of users that have liked the blog. It forms a relationship with Users
    h. **Comments:** List of comments on the blog. It is derived from a relationship with the "Comments" model.
3. **Comments:** This model stores the comments made by users on particular blogs. It is a many-to-many relationship between users and blogs, that has an additional field that stores the comment made by the user. Following are the fields:
    a. **Id:** Unique identifier of the Comment entity
    b. **Comment:** Comment made by the user on the blog
    c. **User id:** Id of the user making the comment
    d. **Blog id:** Blog on which the comment was made

The following are the relationships that exist between the above models:

1. **User and Blogs:** There is a one-to-many relationship between users and blogs.
2. **Likes:** The "likes" table describes a many-to-many relationship between users and blogs.
3. **Follows:** The "follows" table relates the User entity to itself in a many-to-many relationship.
4. **Comments:** The "Comments" table relates users to blogs on which they have commented in a many-to-many relationship while also storing the comment made by the user.

API

The API of this app provides basic CRUD functionality for the User and Blog entities. The API was implemented using Flask Restful package. The routes defined are as follows:

1. **/api/user/{user_id}:** Supports GET, PUT and DELETE operations to read a user's information, update their information and delete the user respectively, with user id being equal to the path variable, {user_id}
2. **/api/user:** Supports the POST operation to create a new user object.
3. **/api/blog/{blog_id}:** Supports the GET, PUT and DELETE operations to read a blog's information, update the blog's information and delete the blog's information with blog id being equal to the path variable, {blog_id}. When a blog is read using the GET operation, the api will return an Image URL attribute, that contains the filename of the Image as stored on the server. This file name can then be used with the route **/api/image/{ImageURL}** to fetch the image from the server.
4. **/api/blog:** Supports the POST operation to create a new blog.

For more detailed documentation on the requests and responses of these routes, refer to the YAML file.

Video Link

https://drive.google.com/file/d/1SxQ5fOEEDn2i6f6BgGTb9E9LuYb4oACg/view?usp=sharing