Documentation for Instagram Application

MUSKAN PATEL (3110495) – Cloud Assignment 03

Table of Contents

Overview

Python-based Flask was used to construct the Instagram Clone App. Users can create an account, authenticate, post, browse posts, edit their profiles, and log out. Google Cloud Storage stores user photos, whereas Google Datastore stores user data.

Documentation of Methods

app.py

- register(): This function facilitates the registration of a new user by verifying the user's input and incorporating the new user into the datastore.
- login(): The present technique oversees the login process of a user by verifying the user's input and ascertaining the existence of the user in the datastore.
- logout(): The method responsible for handling user logouts includes the deletion of session data associated with the user.
- index(): This method displays the primary interface of the application alongside all of the posts generated by the user.
- create_post(): This approach facilitates the creation of a new post by managing user input approval and uploading the post image to Google Cloud Storage.
- view_post(): This method displays an individual post along with its corresponding comments.
- edit_post(): This function facilitates user modification of a post by validating user input and updating the post data in the datastore.
- delete_post(): Upon a user's deletion of a post, the present function undertakes the task of removing the post's data from the datastore and its corresponding image from Google Cloud Storage.
- edit_profile(): By confirming the user's input and updating the pertinent data in the datastore, this approach efficiently handles the alteration of a user's profile.
- delete_profile(): By deleting all of the user's posts and photos from Google Cloud Storage, as well as all of the user's data from the datastore, the current system manages the deletion of a user's account.

forms.py

- RegistrationForm(): The present course delineates the form for registering a new user.
- LoginForm(): This class provides a description of the login form utilised by a pre-existing user.
- PostForm(): The present course delineates the form for generating a novel post.
- CommentForm(): The present course delineates the structure of the commenting form utilised for posts.
- EditProfileForm(): The present course delineates the structure of the form utilised for editing user profiles.

models.py

- User(): This class defines the user data model, comprising the user's identification number, electronic mail address, password, and profile details.

- Post(): This class defines the identification number of the post, the URL of the image, the written description accompanying the image, and the details of the person who created the post.
- Comment(): This class defines the ID, text, and author information for every comment.

utils.py

- upload_image_to_gcs(): In the process of uploading pictures to Google Cloud Storage through this method, a bucket will be generated if one does not already exist.

Documentation of Data Structures:

The programme utilises the data models of User, Post, and Comment as defined in the models.py file. Each model possesses distinct attributes and interrelationships with other models.

The model representing a user possesses the subsequent attributes:

- id: The identifier of the user, which is generated automatically by Google Datastore.
- email: The individual requires a unique and specific electronic mail identifier.
- password_hash: The securely protected representation of the user's password.
- first_name: The given name of the individual.
- last_name: The surname of the individual.
- bio: The biographical information of the individual.
- image_url: The location of the user's profile image in Google Cloud Storage.

The model denoted as "Post" possesses the following attributes:

- id: The identifier of the post is automatically generated by Google Datastore.
- title: The title of the post is a mandatory string field that requires completion.
- content: The post's text, which is required.
- created_at: The date and time of the post's inception are automatically established as the present moment.
- user_id: The integer ID of the author of the post is a necessary requirement.

The User model has the qualities listed below:

- id: The identifier assigned to the user, which is automatically generated by Google Datastore.
- username: The username of the user is a mandatory string field that requires completion.
- email: The email address of the user is a mandatory field of type string.
- password_hash: The mandatory string field of the user's hashed password.
- avatar_url: The optional string field for the user's avatar image is represented by a URL.

The Comment model is characterised by the following attributes:

- id: The identifier for the comment, which is automatically generated by Google Datastore.
- content: The data contained in the mandatory text field for a comment.

- created_at: The timestamp of a comment is automatically generated at the moment of its creation, indicating the exact date and time of its inception.
- user_id: It is imperative to include the user's identification number, an essential integer field, in each comment.
- post_id: The integer field that is required for the identification of the post to which the comment is associated.

Overview of Methods

app.py

- home(): This function is responsible for managing inquiries on the home page. The webpage displays all the articles that are stored in the Datastore.
- create_post(): This method addresses post-creation inquiries. If not authenticated, the system redirects to the login page. Authenticated users can store posts in the Datastore and access data from the create post form.
- view_post(post_id): This function manages post views. This function fetches a Datastore post and its comments by ID.
- edit_post(post_id): It handles post-edit requests. The statement confirms user login and post submission. Non-logged-in users are redirected to the homepage. If the user is authenticated and authorised to create a post, the function pulls the relevant post from the Datastore, populates the form with existing data, and updates the post with any new information.
- delete_post(post_id): Removes remarks. The statement validates user login credentials and post creation. Non-logged-in users are redirected to the homepage. If the user is authenticated and the post was written by them, the function will delete the post and any comments from the Datastore.
- register(): This function is responsible for managing the registration requests of new users. The system receives the data submitted through the registration form, conducts authentication procedures, and subsequently generates a novel user in the Datastore based on the verified information.
- login(): When a user logs in, this procedure responds. It takes the data from the login form, checks it for accuracy, and then logs the user in if the data is correct.
- logout(): The logout process is managed by the invocation of the logout_user() function in Flask-Login, which subsequently redirects the user to the home page.
- @app.route('/logout') links the method to /logout.
- The Flask-Login decorator @login_required restricts access to this page to logged-in users.
- Flask-Login's logout_user() function removes the user's session cookie.
- The flash("You have been logged out. ", "success") function adds the message to the flash queue and displays it on the next page.
- Index return redirects to home page.
- profile(): After authentication, the user's profile page appears. @app.route('/profile') links /profile to the function.
- Flask-Login's decorator @login_required restricts this page to logged-in users.
- Post.query(Post.user_id == current_user.get_id()) retrieves all database posts with the current user's ID.The order(-Post.created_at) method sorts posts by creation date. The database retrieves posts via fetch().

- "return render_template("profile.html", posts=posts")" invokes the "render_template" method and passes "posts" data to show posts on a webpage using the profile.html template.
- edit_profile(): This function manages the user's profile form submission. The decorator gives the function /edit_profile and states that it can handle GET and POST requests. @app.route('/edit_profile', methods=['GET', 'POST']) maps the URL function to HTTP GET and POST requests.
- Flask-Login's decorator @login_required restricts this page to logged-in users.
- form = EditProfileForm() creates a new instance of the EditProfileForm class. if form.validate_on_submit(): checks for form submission and field validity.
- current_user.username = form.username.data changes the user's username with the form's username value.
- Finally, db.session.commit() saves the changed user object to the Google Datastore, and a flash message confirms the changes.
- delete_post(post_id): Deletes a post. Post.query.get() obtains the post from the Google Datastore using a post ID. db.session.delete() and db.session.commit() delete posts from the datastore. The user is returned to the index page after a successful deletion flash notification. Error flashes if the post is not found.
- get_post(post_id): This code fetches a Google Datastore post object using Post.query.get(). It returns the post object for a post ID. get_posts()
- Post.query.order_by() finds all Google Datastore posts in this function. The order_by() method sorts posts by date, starting with the most recent. all() lists all posts.
- get_user_posts(user_id): This function uses Post.query.filter_by() and Post.query.order_by() to obtain a user's Google Datastore postings. It filters postings by a given user ID. The most recent posts are then presented first by date. all() lists all posts.
- upload_file(file, folder): Uploads a file to Google Cloud Storage. It requires the file and folder to upload. The code builds a Cloud Storage client using storage.Client() and fetches the bucket to upload the file to using storage_client.get_bucket().

Conclusion

We built an Instagram clone using Flask, Google Cloud Storage, Google Datastore, and Firebase Bootstrap. After setting up the application's framework, we added capabilities including user registration, login, and authentication, posting and editing photos, and viewing and commenting on other users' posts. Flask-WTF and Flask-Session simplified development. We also used Google Cloud Storage and Datastore for user and post data storage and photo uploads. This project shows how to build a web app with Flask and Google Cloud technologies.

References

1. Kim, Y., Jo, H., & Lee, K. (2020). Understanding the factors that influence Instagram users' intention to follow and unfollow brand accounts. Telematics and Informatics, 52, 101412.
2. Shams, F., & Yaghoubi, N. M. (2020). Instagram as a social network site: Its usage patterns, determinants, and effects. Telematics and Informatics, 47, 101330.
3. Gómez-Ortiz, D., Martí-Parreño, J., & Torrent-Sellens, J. (2021). Instagram users' satisfaction with brand-sponsored content: A study of mediating effects. Journal of Business Research, 132, 717-728.