Instagram Application CPA Griffith College Dublin

Assignment 3

Rohin Mehra
Student Id:3082862
Department: Big Data Analysis and Management

5 May 2023

Application Code Functions and Operations Description

Imports we use and their purpose of use will be explained in brief

```
import datetime
import flask
import google.oauth2.id_token
from flask import Flask, render_template, request, session
from flask import abort, make_response
from google.auth.transport import requests
from google.cloud import datastore, storage
from werkzeug.utils import secure_filename
import local_constants
```

The purpose of each imported library is explained below:

1. datetime: This library is used to get the current date and time.
2. flask: This library is used to create the web app.
3. google.oauth2.id_token: This library is used to authenticate users using their Firebase ID token.
4. Flask, render_template, request, session: These libraries are used to create and manage the Flask web application.
5. abort, make_response: These libraries are used to abort requests and create HTTP responses.
6. google.auth.transport.requests: This library is used to authenticate users.
7. google.cloud.datastore, storage: These libraries are used to store data in the datastore and storage bucket.

8. secure_filename: This library is used to secure file names.

In addition, the local_constants library is imported to store constants, for this, we also use the Python file local_conatants.py. The code initializes a Flask web app and creates a datastore client and a request adapter for the Firebase Admin SDK.

def createUserInfo(claims)

This function is used to create a new user in the Datastore. It takes in the claims dictionary which contains user information such as email and name. It creates a new entity with the user's email as the key and updates it with the user's email, name, and empty lists for post IDs, followers' emails, following emails, followers list and the following list of people. Finally, it stores the entity in the Datastore.

retrieveUserInfo(claims)

This function is used to retrieve a user's information from the Datastore. It takes in the claims dictionary which contains user information such as email and uses it to get the user's entity key. It then retrieves the entity from the Datastore using the key and returns it.

blobList(prefix)

List all the blobs in the storage bucket matching a given prefix.

prefix: A string indicating the prefix to search for.

Returns: A list of blobs in the storage bucket matching the given prefix.

blobUpload(image_file, file_path)

Upload an image file to the storage bucket.

image_file: A file object representing the image to be uploaded.

file_path: A string indicating the path to the file to be uploaded.

Returns: None.

blobDownload(filename)

Download an image file from the storage bucket.

filename: A string indicating the name of the file to download.

Returns: A byte string containing the downloaded image file.

Raises: ValueError if the blob with the given filename does not exist.

blobDelete(image_path)

Delete an image file from the storage bucket.

image_path: A string indicating the path to the file to be deleted.

Returns: None.

def blobPublicURL(filename)

This function retrieves the public URL of a file from the storage bucket. The function takes in a filename as a parameter and returns the public URL of the corresponding file if it exists. If the file does not exist in the storage bucket, it raises a ValueError exception.

Parameters:

filename: A string parameter that specifies the filename of the file for which the public URL is to be retrieved.

Returns:

public_url: A string that represents the public URL of the file.

addCommentToPost(content)

This function adds a comment to a post in the data store. It creates a comment object containing the content of the comment and the current timestamp. It then stores this object in the datastore using the datastore_client.put() method. It returns a boolean value True if the comment was added successfully.

deletePost(post_id, claims)

This function deletes a post from the data store. It first verifies that the user is authenticated using Firebase auth. It then checks if the post exists in the datastore and verifies that the authenticated user is the owner of the post. If these conditions are met, the function deletes the post from the datastore and the post image from the storage bucket using the datastore_client.delete() and blobDelete() functions respectively. It returns a boolean value True if the post was deleted successfully.

add_follower(user_info, follower_email)

Adds a follower to the user_info in the datastore. The function takes in the user_info and the email address of the follower to be added as parameters. The function checks if the follower is already in the user's followers list, if not, then it adds the follower to the list and updates the user in the datastore.

add_following(user_info, following_email)

Adds a following to the user_info in the datastore. The function takes in the user_info and the email address of the following to be added as parameters. The function checks if the following is already in the user's following list, if not, then it adds the following to the list and updates the user in the datastore.

remove_follower(user_info, follower_email)

Removes a follower from the user's followers list in the datastore. The function takes in the user_info and the email address of the follower to be removed as parameters. The function checks if the follower is in the user's followers list, if yes, then it removes the follower from the list and updates the user in the datastore.

remove_following(user_info, following_email)

Removes a following from the user's following list in the data store. The function takes in the user_info and the email address of the following to be removed as parameters. The function checks if the following is in the user's following list, if yes, then it removes the following from the list and updates the user in the datastore.

get_followers_list(user_info)

Retrieves the list of followers of a user from the data store. The function takes in the user_info as a parameter. The function creates a query for the UserInfo kind and filters it by the user's email address. It then executes the query and extracts the email addresses from the results, and returns the list of followers.

get_following_list(user_info)

Retrieves the list of following a user from the data store. The function takes in the user_info as a parameter. The function creates a query for the UserInfo kind and filters it by the email addresses of users who are following the user. It then executes the query and extracts the email addresses from the results, and returns the list of the following.

search_users_by_profile_name(search_query)

Searches for users in the datastore whose profile name contains the given search query. Returns a list of User entities that match the query.

get_timeline_posts(user_info)

Parameters:

user_info (dict): A dictionary containing user information.

Returns:

list: A list of dictionary containing the timeline posts.

The function retrieves the list of posts from the users the current user is following and the current user themselves

in descending order of timestamp. The function first calls the `get_following_list()` function to get the list of users

the current user is following and adds the current user to the list. It then creates a query for the `Post` kind and

filters the query by the following list. The query is then ordered by timestamp in descending order and executed to

retrieve the posts. The function then iterates through the retrieved posts and creates a dictionary for each post.

The dictionary contains the post ID, caption, user email, timestamp, image URL, and image bytes. The function adds the

dictionary to the list of timeline posts and returns the list.

Function: root()

Description: This function is the root page of the Flask app. It retrieves the ID token from the request, retrieves the user's information if they are logged in, retrieves the list of posts, followers, and following from the database, and renders the index.html template with the retrieved data.

Inputs:

None

Outputs:

Returns the rendered index.html template with the retrieved data.

Function: allowed_file(filename)

Description: This function checks if the file extension is allowed for upload. It compares the file extension to a set of allowed image file extensions.

Inputs:

filename: A string representing the name of the file.

Outputs:

Returns True if the file extension is allowed, and False otherwise.

Function: add_comment_handler(post_id)

Description: This function is called when a user submits a comment on a post. It retrieves the ID token from the request, verifies the token, retrieves the comment content from the request, adds the comment to the post in the datastore, retrieves the post from the database, and renders the comment.html template with the retrieved data.

Inputs:

post_id: An integer representing the ID of the post.

Outputs:

Returns the rendered comment.html template with the retrieved data.

Function: add_post_handler()

Description: This function is called when a user submits the add post form. It retrieves the Firebase ID token from the request, verifies the token, retrieves the user information from the database, retrieves the post information from the form, uploads the image file to the storage bucket, adds the post to the datastore, and redirects the user to the post page.

Inputs:

None

Outputs:

Returns the rendered post.html template.

Function: delete_post()

Description: This function is called when a user submits a request to delete a post. It retrieves the Firebase ID token from the request, verifies the token, retrieves the post ID from the request, checks if the authenticated user is the owner of the post, deletes the post from the datastore, deletes the post image from the storage bucket, and redirects the user to the index page.

Inputs:

None

Outputs:

Returns the rendered index.html template.

Function: download_file(filename)

Description: This function is called when a user requests to download a file from the storage bucket. It retrieves the file bytes from the storage bucket, sets the response headers to force a download of the file, and returns the response.

Inputs:

filename: A string representing the name of the file.

Outputs:

Returns the response containing the file bytes and headers.

Function: add_follower_handler()

Description: This function is called when a user submits a request to add a follower. It retrieves the Firebase ID token from the request, verifies the token, retrieves the user information from the database, retrieves the follower information from the form, adds the follower to the database, adds the follower to the user's list of followers, retrieves the updated list of followers from the database, and redirects the user to the index page.

Inputs:

None

Outputs:

Returns the rendered index.html template.

Function: add_following()

Description: This function is called when a user submits a request to follow another user. It retrieves the Firebase ID token from the request, verifies the token, retrieves the user information from the database, retrieves the email of the user to follow from the form, adds the user to follow to the database, and redirects the user to the post page.

Inputs:

None

Outputs:

Returns the rendered index.html template.

Function: search_profiles()

Description: This function is called when the user submits the search form. It retrieves the search query from the form, searches for users in the database by profile name, retrieves their profile images, and renders the search.html template with the list of users.

Inputs:

None

Outputs:

Returns the rendered search.html template with the list of users.

Function: timeline()

Description: This function is called when the user navigates to the timeline page. It retrieves the Firebase ID token from the request, verifies the token, retrieves the user information from the database, retrieves the timeline posts for the user, and renders the timeline.html template with the timeline posts.

Inputs:

None

Outputs:

Returns the rendered timeline.html template with the timeline posts.

Function: remove_handler()

Description: This function is called when the user submits a request to remove a follower or following from the database. It retrieves the Firebase ID token from the request, verifies the token, retrieves the user information from the database, retrieves the email and type of the entity to remove from the form, removes the entity from the database, and renders the post.html template with a success or error message.

Inputs:

None

Outputs:

Returns the rendered post.html template with a success or error message.

Function: retrieve_list_handler()

Description: This function is called when the user clicks on the "followers" or "following" button on the post page. It retrieves the Firebase ID token from the request, verifies the token, retrieves the user information from the database, retrieves the list type from the request arguments, retrieves the list of followers or following from the database, and renders the post.html template with the list of entities or an error message.

Inputs:

None

Outputs:

Returns the rendered post.html template with the list of entities or an error message.