ASSIGNMENT 03

Name: Gokul Sreeletha Kannan
Student ID: 2985939
Course: MSc Computing
Assignment: Building a simplified replica of Instagram
Module: Cloud Platforms and Applications

INTRODUCTION

This is a documentation for the web application designed using webapp2 framework on the Google app engine. Interfaces are rendered using the Jinja2 template engine and ndb is the database used to store data. For applying style to the application bootstrap library is used. This application is a simplified replica of Instagram. The users can login and create their own profile. They can create posts comment on posts. They can also follow others and view their posts on their timeline. In the profile page of each user, they can view the number of followers and the number of people they follow and also by clicking into it, they can see each individual users they follow or follows them.

DATA MODELS

In this assignment, there are three data models used. Each model is inherited from ndb.model.

User()
The User model has information about individual users. Each user will have the following attributes,

- ***email : StringProperty()***, Holds the email of each user
- ***name : ndb.StringProperty()***, Holds the username of each user
- ***followers : KeyProperty()***, Holds the keys of other users that follows the current user. This is a list.
- ***following : KeyProperty()***, Holds the keys of other users whom the current user follows. This is a list.
- ***posts : KeyProperty()***, Holds the keys of posts created by the current user. This is a list of keys of objects from the Post() model.

Comment()
The Comment model is used to hold the comments of each post. Each comment has the following attributes,

- ***text : StringProperty()***, Holds the comment that user wants to add.
- ***author : KeyProperty()***, Holds the key of the user who added the comment.

Post()

The Post model is used to hold the posts that are created by users. Each post will have the following attributes,

- ***caption : StringProperty()***, Holds the caption text for each post.
- ***image : BlobKeyProperty()***, Holds the key of the image that is stored as a blob.
- ***comments : StructuredProperty()***, Holds the value of comments for the post. This is a list of objects from the Comment model.
- ***date : StringProperty()***, Holds the date and time when the post was created.
- ***user : KeyProperty()***, Holds the key of user who created the post.

MainPage

The MainPage class is the home page of the application. It resides in main.py, which is the main handler of the application. It is rendered using the template main.html. The MainPage class has a get() and a post() method. In the get() method, the user can login, the user object is fetched using the credentials when the user is logged in, if the user is a new one, a new user is created in the database. Then, all the posts are queried in the reverse chronological order and the posts from the current user and his following are stored in the allPosts list. An upload_url is created to allow users to upload image files. It is routed to '/upload' which is the url to UploadHandler class. In the post() method, there is only one functionality, the ability to add comments. When the user clicks the Add Comment button, the post key is passed to the method and the post is retrieved and the currentUser is also fetched. If there is something inthe commentText input, then a new Comment object is created and is inserted to post object's comments attribute. Every new comment is inserted into index 0 for convenience of showing all the comments in reverse chronological order.

UploadHandler

The UploadHandler class handles the upload of the blob. It is inherited from blobstore_handlers.BlobstoreUploadHandler. It only has a post() method. In the post() method, the image file that is uploaded is stored in upload and is saved as blob. A new Post object is created and the key of the post is inserted into the first index of the currentUser.posts. This is for convenience to show the posts in reverse chronological order.

DownloadHandler

The DownlandHandler class handles the download of the blob. It is inherited from blobstore_handlers.BlobstoreDownloadHandler. It only has a get() method. In the get() method, the post object is fetched and stored in post. Then the image in the post is retrieved using the send_blob() method.

Profile

The Profile class handles the request regarding the profile of each user. It is inherited from webapp2.RequestHandler. It has both get() and post().

In the get() method, the current user is fetched and is stored in currentUser. The user that the profile belongs to is saved in selectedUser. The followString is for the follow button in the UI, to know if the text in the button is Follow or Unfollow. This is used in the post() method. All posts that the user is fetched and is stored in allPosts. The followString is set to “Follow” by default. If the selectedUser.key is in currentUser.following list it is set to “Unfollow”. All the above values are passed as template_values. The length of selectedUser.following and selectedUser.followers is passed to UI to show the number of followers and following. The Profile class is rendered using profile.html.

In the post() method, the currentUser and selectedUser are fetched. If the user clicked the Follow button, the key of corresponding users is appended to the respective lists. Similarly, the key is removed if the user clicked Unfollow. If the user click Add Comment, the corresponding post is fetched and a Comment object is created and inserted to the first index of the post.comments.

Search

The Search class handles the search functionality of the app. It is also inherited from webapp2.RequestHandler. The class only has a get() method. Initially currentUser is fetched. Then queryInput stores the string which user wants to search. The text stores the lowercase of queryInput and the searchOutput is queried using text and limit and is passed into the UI as template_values. The class is rendered using search.html.

FollowDetails

The FollowDetails class is used to show the followers and following of each user. It is also inherited from webapp2.RequestHandler. The class only has a get() method. In the get() method, currentUser and selectedUser are fetched. followString shows if the user is looking at followers or following. These values are passed into the UI as template_values. The UI is rendered using followDetails.html.

Design Decisions

The users loading the app for the first time will be asked to login. The user after logged in is redirect to the home page. Every page has a navbar on top that gives a link button for the user to route to the user profile and also a form that allows users to search for other users. The contents of the search form are sent to Search class.

In the home page, the name and email of the current user are shown. Then there is a form that allows the user to create a post. The post has a caption and a file uploader that allows image files. The contents of the form are sent to UploadHandler class. The home page also has a section that shows all the post of the logged-in user and the users that the logged-in user follows in reverse chronological order. Each post has allows the users to add comments. The latest 5 comments are shown with each post with an option to expand all the comments.

In the profile page, the name and email of the selectedUser are shown. Along with there are two link buttons that shows the number of followers and following of selectedUser. The currentUser can click on the buttons to view all the followers and following. If the currentUser is not on his profile, there will be a button that enables the currentUser to follow or unfollow the selectedUser. The profile also has a section for all posts from the selectedUser in reverse chronological order and each post has the comments section similar to the one on the home page.

The followDetails page is rendered when the user clicks the following or followers link buttons. It shows the list of followers or following as link buttons. If the user doesn't have followers or following, it shows no followers or following.

The search page shows the email and name of users that are queried using Search class as link buttons. The user can navigate to each of the searched profile by clicking the link buttons. If the search is empty, it shows no match found.