Cloud Computing

Assignment 3

Simplified replica of Instagram

Motivation

This web application was developed for the requirement of Master's in Big Data Management and Analytics, Semester 2- Cloud Computing module. The aim of this assignment is to develop a simplified replica of the social media platform Instagram as a PaaS application.

1. Description

'POSTIt' is a PaaS (Platform as a Service) application that is developed using Google App Engine with Python version 2.7. The application serves as a simple version of social media network for multiple users. This application allows users to login and logout, and also create a profile for themselves with desired profile picture and information about them. In the application users can also create posts containing images with a text caption. Users can follow and unfollow each other. There is a timeline that shows the 50 posts of the people the user is following and also his posts. Users can search for each other using their names and see their profile page. The profile page contains all the posts of the user and information about the user such as the profile picture, username, information about him and the number of followers and following. Users can also post comments on the posts. The application is very elegantly designed and resembles the UI of the familiar social networks. This application is taking advantage of Blobstore of Google App Engine which allows the storage of binary large objects [1]. In this application this allows for storage and retrieval of images.

2. File organization and naming conventions

All the image files are included in the 'img' folder and the CSS stylesheet is included in the 'css' folder. These folders with the other yaml, python and html files are zipped together for submission. All files are named according to their actual functionalities making them easier to understand. Similarly, data model names and variable names are also self- explanatory.

3. Images

All images included to enhance the aesthetics of the User Interface are designed using the online platform Canva. Images are thus not violating any copyright laws and are solely created for the purpose of this application.

4. Reference materials used for learning

In order to learn the basics of Google App Engine, the 'Google App Engine with Python' pdf provided as part of the module was utilised. To deeply understand, the concepts of entity, file upload using BlobStore etc., Google App Engine documentation was referenced. All of the references are provided in the end of this document.

5. Code Documentation

5.1. Python classes and methods

1. app.yaml

This file contains all the configuration details including name and version information about the library used (jinja2) and the static directory for 'img' and 'css' folders. Handlers transfers requests for all the URLS by using the variable 'app' which is defined in python file (main.py) which has the routing table for this web application.

2. get() in all Classes

All classes contain the get method for handling HTTP get request and accepts self as argument. This means that this method may only be called on an instance of the class. It will render the result of a GET request to this class. [1] This functionality of get() method is common for all classes. Other functionalities of get() methods specific to classes are documented in appropriate sections.

3. Routing table in main.py

All webapp2 applications are instances of a WSGIApplication which has all of the required code to transfer requests to the appropriate classes. Application routing table is defined as a list in main.py. Tuples are used in the list that states that any requests to the root of the application at '/' calls the respective methods of an instance of the 'MainPage' class. [1]

4. MainPage class

4.1.1. get()

This method creates a URL string as 'login'. The login URL and URL string is rendered as hash map to the index.html which is the Login page. When a user is logged in for the first time, they are redirected to the signup page where the user has to fill up a form containing details. This method renders the email address and id of the logged in user to the signup page. Google App Engine's default login- logout service is used for this assignment.

5. Logout class

5.1.1. get()

This method creates a logout URL with Google App Engine's default login-logout service. The user is redirected to the login page, when the user clicks the Logout button in the navbar from anywhere in the dashboard. The URL and URL string will be rendered to login page (index.html).

6. Timeline class in timeline.py

6.1. get()

The application has a timeline that shows all of the last 50 posts in the reverse chronological order. For this purpose, the get() method renders the user object, all posts and the list of users for performing a search to the Timeline (timeline.html) page. The list of users is rendered using a python hash map, a dictionary.

7. Profile class in timeline.py

7.1. get()

Every user has a profile page associated with them which contains their posts, number of followers and following and the lists of followers and following. The get() method in Profile class fetches all of the posts of the users by invoking other methods written separately from this class- getUserKeyData() that returns user data by accepting user key as argument and getAllPostsUser() which returns all posts of the user. Each profile has a profile picture which can be uploaded using the upload url that BlobStore provides, in this method. This method also invokes the all_users() method to return all users for the search functionality. This method renders the template and details to Profile page(Profile.html)

8. UpdatePic class in timeline.py

This class subclasses the BlobstoreUploadHandler to handle Blobstore and to provide functionality for uploading an image. [1]

8.1. get()

This method provides the necessary functionalities to allow users to update profile pictures. The user object is passed to Profile page (profile.html) for displaying the information related to the user which is necessary for updating the user's profile picture.

8.2. post()

In this method, the new profile picture will be stored in the PicStore model which stores the profile picture as a BlobKeyProperty. The old profile picture is retrieved by querying using the id of the profile picture which is stored as a separate attribute. If there is a picture, it will be updated by obtaining it's key and modified, else a new object is created.

The input for uploading images is filtered on client side to jpg or png using 'accept' option in html.

9. DisplayPics class in timeline.py

This class subclasses the BlobstoreDownloadHandler class inorder to implement the functionality for connecting to the blobstore and to fetch the image file. [1]

9.1. get()

Two modes are assigned when page is loaded, i.e 'PROFILE' mode and 'POST' mode. When the mode is 'PROFILE', PicStore model is queried to fetch the profile picture and for 'POST' mode key of picture is obtained from Post without a query, since it is a BlobKeyProperty. send_blob() function provides that file to the user from the blobstore. As arguments, the indexes are passed with the send_blob() function.

10. DoPosts class in timeline.py

This class subclasses the BlobstoreUploadHandler to handle blobstore and to provide functionality for uploading an image. [1]

10.1. post()

This is the most important method which provides the functionality of creating a post. This method invokes the getUserInfo() method to retrieve the details about the user and then getTimeLine() method to fetch the posts in the timeline. For creating a new post, the id of the user, date and time of creation and content is stored in the datastore using this method. For the image functionality, this method fetches what was uploaded and index the received list. Using the BlobKey a BlobInfo object is created which contain filename and size details. [1] The details of the fetched posts and the current post is sent to timeline.html, in order to show a refreshed timeline with the new post.

11. SignUp class in timeline.py

11.1. get()

This method renders the current user status and the user object from InstaUser model to the sign up page(Signup.html) as a template.

11.2. post()

Users have the ability to sign up to the application using their information while they login with an email address for the first time. This method stores the details provided by the user in the signup form. The details such as name and about is obtained from the form and stored in the datastore. If the user selects 'Cancel' option the user will be redirected to login page and url string is rendered to the login page (index.html).

12. FollowUser class in timeline.py

12.1. post()

This method provides the functionality for users to follow and unfollow each other. post() method invokes the getUserInfo() method to get the user details and the action is obtained from the form. This invokes the getUserKeyData() method to fetch the user details from the key of the user object. When the action is 'Follow', the data is stored in the datastore and the list of followers is appended and the count of followers of that user is incremented. When the action is 'Unfollow, the user is removed from the list of followers and the number is decremented.

13. Comment class in timeline.py

13.1. post()

This method is used to allow users to add comments to their own posts and the posts of other users. Comment is limited to 200 characters using 'maxlength' option in html form. Users can comment on posts that appear in profile and timeline page. For this

'page' attribute is set to either 'timeline' or 'profile'. If the comment has to be made on a post from timeline, getTimeLine() method is invoked to fetch the timeline posts for auto refreshing the page after comment is entered to show the comments right away without any delay. If it is 'profile', getAllPostsUser() method is invoked to fetch the user posts. The comment text is retrieved from the webpage, and the date and time of comment creation is also stored using utcnow() method of datetime. The comment is then appended to the list of comments and stored in the datastore. The appropriate timeline or profile page with the new comments will be loaded.

Only the latest 5 comments will be shown directly and to see full comments, users has to click on the expand option. This is achieved using a script that listens on the event, the script is written in JavaScript.

14. SearchUser class in timeline.py

14.1. get()

get() method invokes the getUserInfo() method to get the user detail and getUserIdfromName() method to retrieve the user detail using the name entered in the search field. The list of users in the system will be rendered to Profile.html using all_users() method. getAllPostsUser() method is invoked to fetch the posts of the user that has to be displayed in that user's profile page, if the search is invalid. If the user tries to search on a non-existent name, an appropriate message will be displayed. This step is done to ensure validation and integrity.

15. FollowList class in timeline.py

15.1. post()

This method is used to render the list of followers and following that has to be displayed when clicked on the number of followers and following. post() method invokes the getUserKeyData () method to get the user detail from key of the user object and getUserInfo () method to retrieve the user. Depending upon the mode- 'Following' or 'Followers' appropriate details is fetched by invoking getUserFollowingDetails() and getUserFollowersDetails() method. These lists are rendered to the page where followers and following list will be displayed (FollowList.html) as hash maps.

16. Other methods in timeline.py

16.1. getUserInfo()

This method accepts the user key as argument and returns the user data as a dictionary with user key and value as the user object.

16.2. getUserKeyData()

This method also accepts the user key as argument. However, this method returns the only the user object obtained from the key of the user.

16.3. getUserKeyFollowingData()

This method also accepts the user key as argument and returns the attribute that stores the list of users that the particular user is following.

16.4. getUserIdfromName()

This method accepts a name and returns the id of the user by querying on this name, if the name is not an empty string.

16.5. getAllPostsUser()

This method accepts the id argument and returns the details of all posts of the user in a reverse chronological order. Only 50 posts are fetched at a time. The method appends the id, caption and the created date and time of the post to a list. Along with the posts, the comments are also fetched in reverse chronological order and the final list is returned to the invoking method.

16.6. getUserFollowersDetails()

This method also accepts the user key as argument. If the user has followers, the username and id of those followers are added to a list and the list is returned.

16.7. getUserFollowingDetails()

getUserFollowingDetails() method also accepts the user key as argument. If the user is following others, the username and id of those following are added to a list and the list is returned.

16.8. getTimeLine()

getTimeLine() is a very important method as it is responsible for returning all the posts that should appear in a user's timeline. This method invokes the getUserKeyFollowingData() to get the details of the users the particular user is following and to display the posts accordingly. Only 50 posts are fetched to be shown in the timeline at a time. The method appends the id, caption and the created date and time of the post to a list. Along with the posts, the comments are also fetched in reverse chronological order and the final list is returned to the invoking method

16.9. commentToList()

This method accepts a comment object and returns the comment as a list by appending the username, text and the date and time of the comment creation. The list is returned back to the invoking method.

16.10. postToList()

This method accepts a post object and returns posts as a list by appending the id of the post, text caption of the post and the date and time of the post creation. It calls the getUserKeyData() to get user details to append to the list. The list is returned back to the invoking method.

16.11. all_users()

This method accepts the email address and is used to return the list of users that has to be used for the Search functionality. This is achieved by querying on the InstaUser model using the email address and fetching all the users that match the query.

5.2. JavaScript functions

5.2.1. validatePic () in timeline.html

validatePic () JavaScript function is used to ensure that only JPG and PNG images can be uploaded. This is done in the client side. If the image is not of the required formats, user will be alerted with an error message.

5.2.2. checkPicExist() in timeline.html

checkPicExist() is used to ensure that images are always posted along with text. If the user attempts to post text without image, an alert will be shown. This is achieved by checking if the length of the input is greater than 0.

5.2.3. validateData () in signUp.html

This JavaScript function is used to alert the user to fill the user name and name field of the signup form, and inform the user that it cannot be null. This is used to ensure that incomplete information don't enter the datastore.

6. Data Structures and Models

6.1. Models

6.1.1. InstaUser in instauser.py

The InstaUser class extends the ndb Model.

a. 'emailaddress': Email address contains alphabets, numbers and characters like @. Thus, string is the best choice for it. It also allows for the usage of Google App Engine's default login-logout service**.**
b. 'pro_pic': This attribute stores the file name of the profile picture. Thus, it is defined as StringProperty.
c. 'user_id': This attribute stores the user id of the user, just like social media applications, user can store their short user id which will be displayed in profile. It can contain characters, numbers, alphabets etc. Hence it is also defined as StringProperty.
d. 'user_name': User name contains the proper full name of the user which can have alphabets . Thus, string is the best choice for it and StringProperty is used.
e. 'about': This attribute stores the text description of the user, it contains alphabets, numbers and other characters. Thus, string is the best choice for it.
f. 'followers': This attribute stores the list of the followers of the user. It is a list of string values. List is created by setting repeated as True.
g. 'following': This attribute stores the list of the of people followed by the user. It is a list of string values. List is created by setting repeated as True.

6.1.2. Post in posts.py

The Post class extends the ndb Model.

a. 'user_id: This attribute stores the id of the user who is the creator of the post as a StringProperty with indexed set to True to allow indexing.
b. 'createdDt': This stores the date and time of the creation of the post and hence it is defined as a DateTimeProperty with auto_now_add set to True, to get the time and date at the time of creation.
c. 'content': This attribute stores the text caption of the image of the post, it contains alphabets, numbers and other characters. Thus, string is the best choice for it.
d. 'post_pic'=This attribute stores the image of the post and it is defined as a BlobKeyProperty to allow for storing large images.
e. 'comments'= This attribute stores the list of comments in the post. It is set to be a StructuredProperty to create relationship with the Comments model.

6.1.3. Comments in comment.py

The Comments class extends the ndb Model.

a. 'userId: This attribute stores the id of the user who is posting the comment, as a StringProperty with indexed set to True to allow indexing.
b. 'content': This attribute actual text in the comment. Comments can contain alphanumeric characters. Hence it is defined as StringProperty.
c. 'createdDt': This stores the date and time of the comment creation and hence it is defined as a DateTimeProperty with auto_now_add set to True, to get the time and date at the time of posting the comment.

6.1.4. PicStore in picStore.py

The PicStore class extends the ndb Model.

a. 'picId': This attribute stores the id of the user, who is the owner of this profile picture. It is defined as a StringProperty.
b. 'pics': This attribute stores the profile picture of the user and it is defined as a BlobKeyProperty to allow for storing large images.

6.2. Data Structures

6.2.1. Dictionaries

Dictionaries offer powerful functionalities like hash maps. Throughout this project, hash map of all the values that is to be rendered in the template, are used. This is particularly useful with jinja2 to render content, message, entity objects, user, posts, followers and following user details and comments etc.

6.2.2. Lists

Lists are used in many classes to store the posts, followers, following and comments, date of post and comment creation, and other details. Lists were chosen because it is easy to

implement and it is easy to append values to list. Application routing table is also provided in the form of a python list.

6.2.3. Tuples

Tuples are given in the application routing table list to ensure that requests to the root of the application invoke the respective methods of an instance of the MainPage class. [1]

7. Design Decisions

The simplified replica of Instagram is named as 'POSTIt, as it meaningfully represents the intent of the application. The name was chosen after understanding that the users are more interested in the kind of posts in their social media accounts. The main focus while building this application was to utilize the beautiful design options that Bootstrap provides to give a very elegant and familiar social media look and feel to the 'POSTIt' application. The design of this web application is very familiar to the timelines and profile page layouts that users might be familiar to in other social media platforms. Bootstrap borders and boxes are used for improving the aesthetics of the application. The layout is designed with one top navigation bar and two columns beneath structure, with the content below changing as different links are being selected by the user, this choice was made to develop a consistent web application. This decision was made to ensure that users has quick access to their profile pages to see their posts and also the timeline to see the post of all the people they follow. Consistent template is the key for giving a professional look to the application. Plain white background seemed to be the best choice for a social media platform where images are posted. Images can be of any colour thus it is necessary to have a background that doesn't affect the image colours or distract from the content. White background brings the user's attention to the image and the caption. The login page is developed with animated images designed using Canva to reduce the mundane look of having a single login button in a plain white background. Appealing images give the users an engaging feel, even though the kind of images users likes can be really subjective [3].

While making design choices for this application, the factors that encourage people to use social media were researched. The most important reason seems to be entertainment and to get away from the hustle and bustle of life for a while. Hence the application is designed to provide users a calming look and feel. Hence use of lot of colours, icons, buttons etc is avoided. Minimalistic web design is used. For easy login-logout facility, logout link is provided in the top navigation bar. Nobody would like to use an application that forces them to navigate to some other pages for basic functionalities like logout. Users don't have to go to a separate page for searching other users as this functionality is also included in the top navigation bar, for the same reason. The ability for users to search all users using both nickname and username is included, to ensure that users can perform search with partial knowledge of other people's accounts also. This is really important in a social media platform where users may not be totally familiar with other users they intend to follow.

Lots of innovative features are included in the application and not just the requirements of the assignment. Users can upload a profile picture by going to their timeline, this feature was

added to ensure that most important features of a social media platform are included. Each post is designed as a separate box to make it easy for users to distinguish between several different posts and the user of the post and comments are also given a different shade for the same reason. The follow and unfollow button are designed to stand out and is given a separate colour to avoid the monotony of white and shade of blue. The ability for users to post comments on a post by simply pressing an Enter button is very useful and increases the ease of using the application. Adding a button to enter a comment would have been very unappealing. Colour choices for the application are finalised by referencing *The Principles of Beautiful Web Design* by Jason Beaird. Dark green and red colours are avoided as much as possible because they are the most common forms of colour blindness. The book also tells that light and shadow can create a visually attractive contrast. This knowledge is utilised while selecting the appropriate box types for posts in Bootstrap [3].

The webpages are responsive because users access the application from different types of devices such as mobile phones, desktops, tablets etc. Another useful feature, is an icon to that takes the user to their own timeline anywhere from the website. This saves unnecessary navigation time searching for the link. The timeline is vertically designed for easy scrolling through the posts. Distinct card-body boxes were selected to display the details of followers, because it looks neat and clean. The book *The Principles of Beautiful Web Design* also talks about the importance of choosing right fonts and avoiding the use of lot of different fonts [3]. Thus, the application will only use one single font. However, a font stack is used in order to cater to a situation of unavailable fonts with different browsers. The fonts that are chosen for the font stack are Tahoma and Verdana. Overall, the design choices reflect the goal to keep the application design innovative and beautiful yet minimalistic.

References

1. Denby, B., 2017. Google App Engine By Example.
2. Google Cloud. 2020. Python On Google App Engine | App Engine Documentation | Google Cloud. [online] Available at: <https://cloud.google.com/appengine/docs/python>
3. George, J., 2014. Principles of Beautiful Web Design, 3Rd Edition.