**Riddhish Atul Bharadva**

**MSC-Computing (Cloud Computing)**

**Assignment 2 (Task Manager)**

**Student Id: 3009299**

**Assignment Description:**

Aim of this assignment is to make an Instagram clone. Every user can come and login to their registered user account. User can select username as per their wish. They can follow and unfollow another user. User can create post after logging in to their account. They can also visit their timeline and view all posts of their following users and their itself (if any post exists).

**Files / Pages:**

In this assignment, I have created 2 types of Files i.e. files with “.py” extension and files with “.html” extension. Files with extension “.py” are responsible for backend working of this project and files with “.html” are responsible for frontend or UI working of this project. Below are the files in detail description of each.

HTML files involved are:

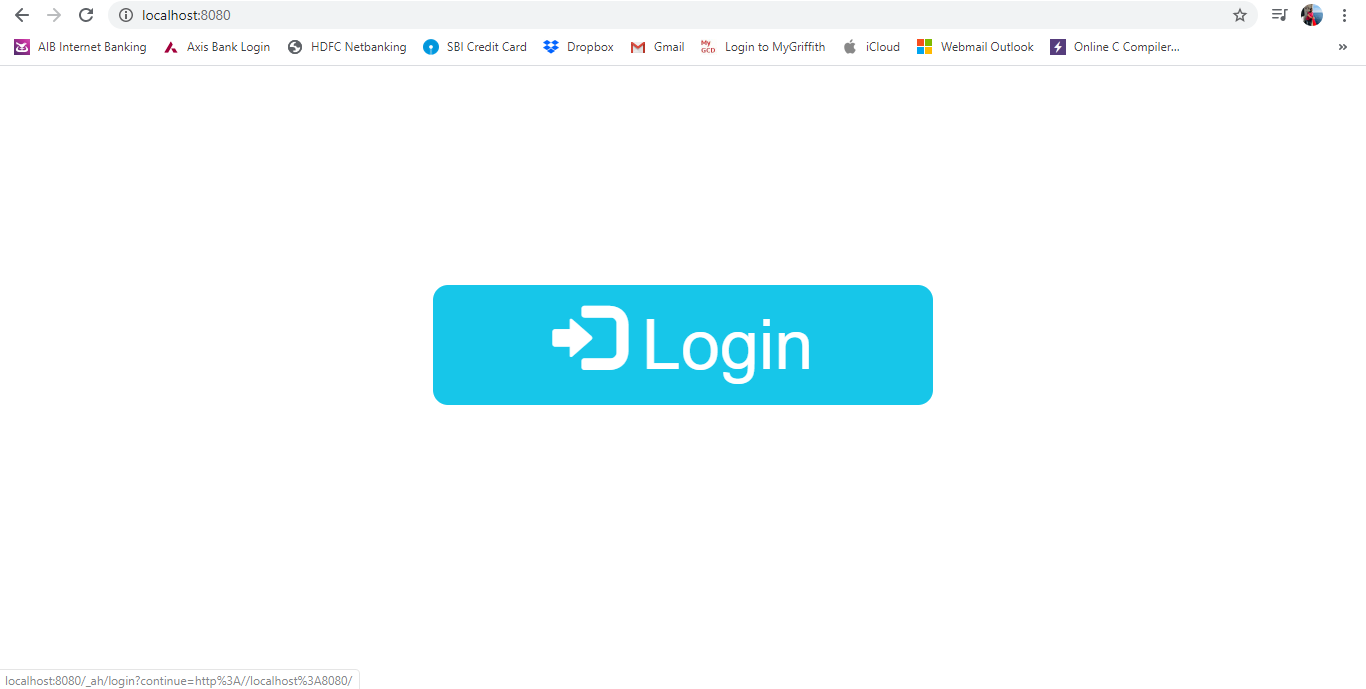
1. ProfilePage.html
2. CreateNewPost.html
3. Followers.html
4. Following.html
5. SearchUser.html
6. OtherUserProfile.html
7. Timeline.html
8. ViewAllComments.html

Python files involved are:

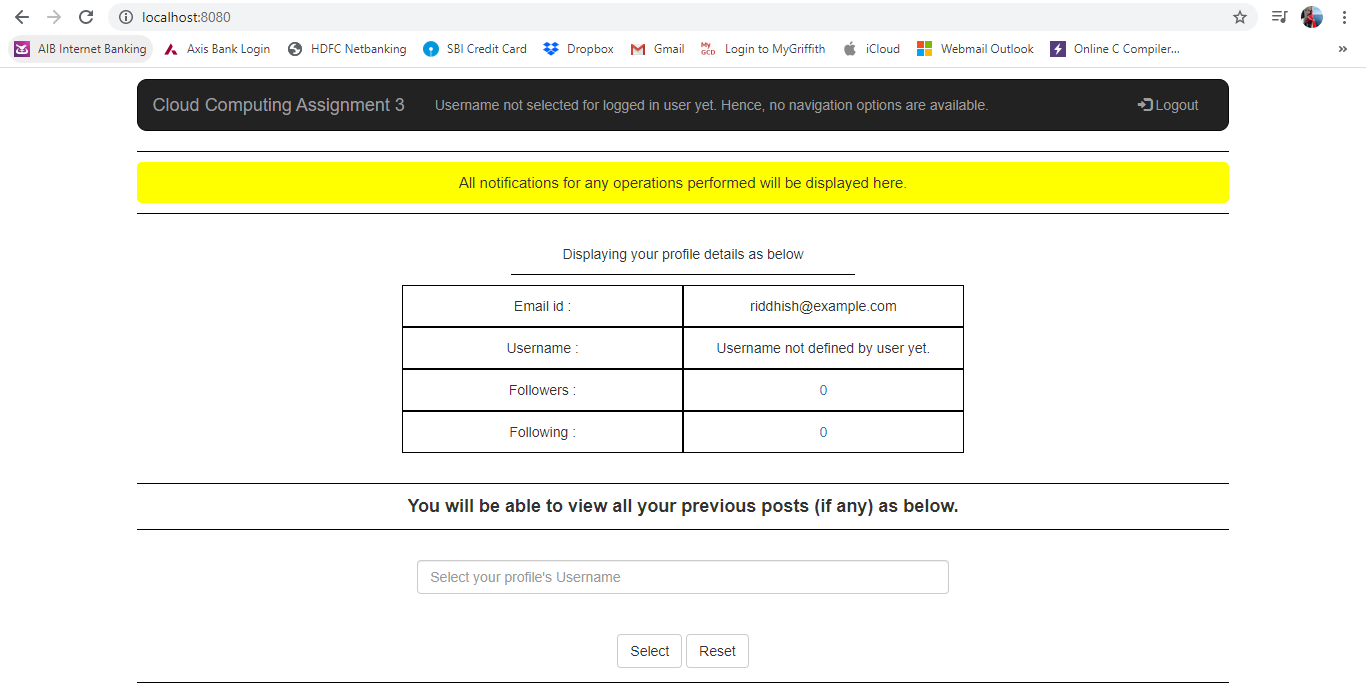
1. ProfilePage.py
2. CreateNewPost.py
3. Followers.py
4. Following.py
5. SearchUser.py
6. OtherUserProfile.py
7. Timeline.py
8. ViewAllComments.py
9. UsersDB.py
10. PostsDB.py
11. CommentsDB.py

**ProfilePage.html**

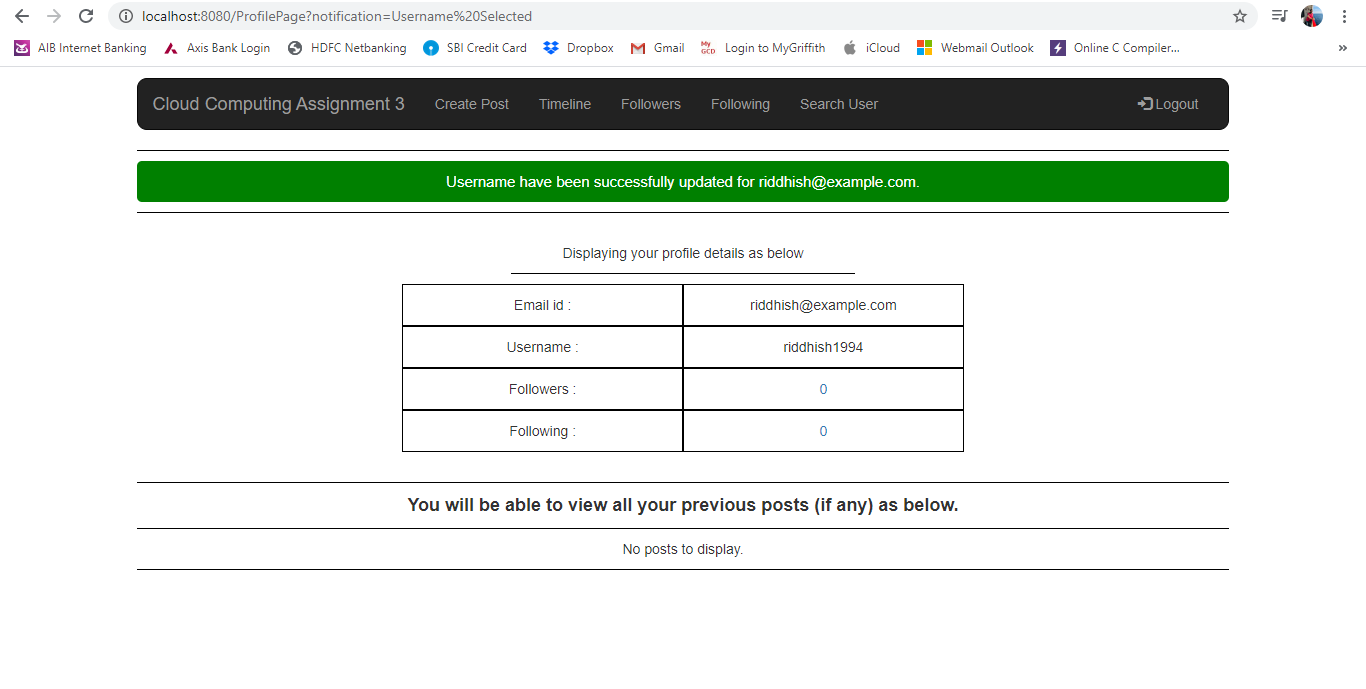
ProfilePage.html is the frontend page / landing page which will be displayed to user when he starts the web application. In case user is not logged in, ProfilePage.html: Image 1 will be displayed else, ProfilePage.html: Image 2 will be displayed. In case user have selected username, ProfilePage.html: Image 3 will be displayed. In case user have already created some posts, page in ProfilePage.html: Image 4.1 and Image 4.2 will be displayed.



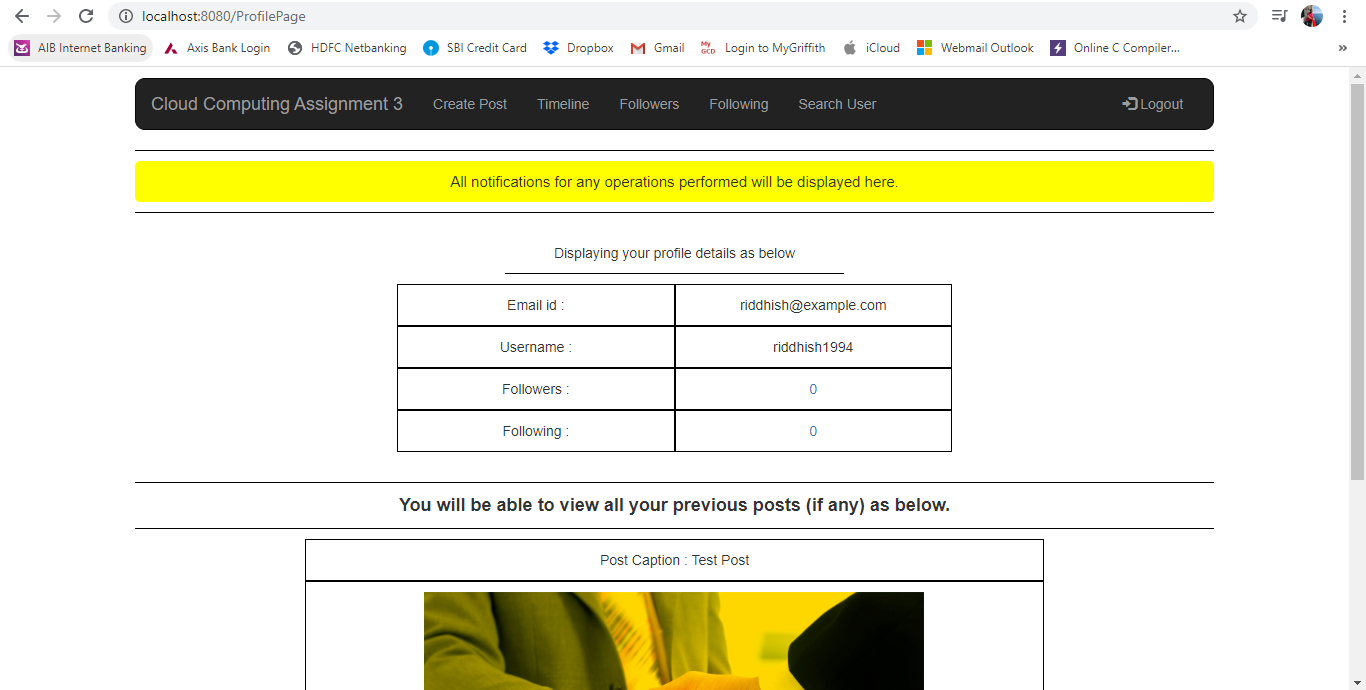
ProfilePage.html: Image 1



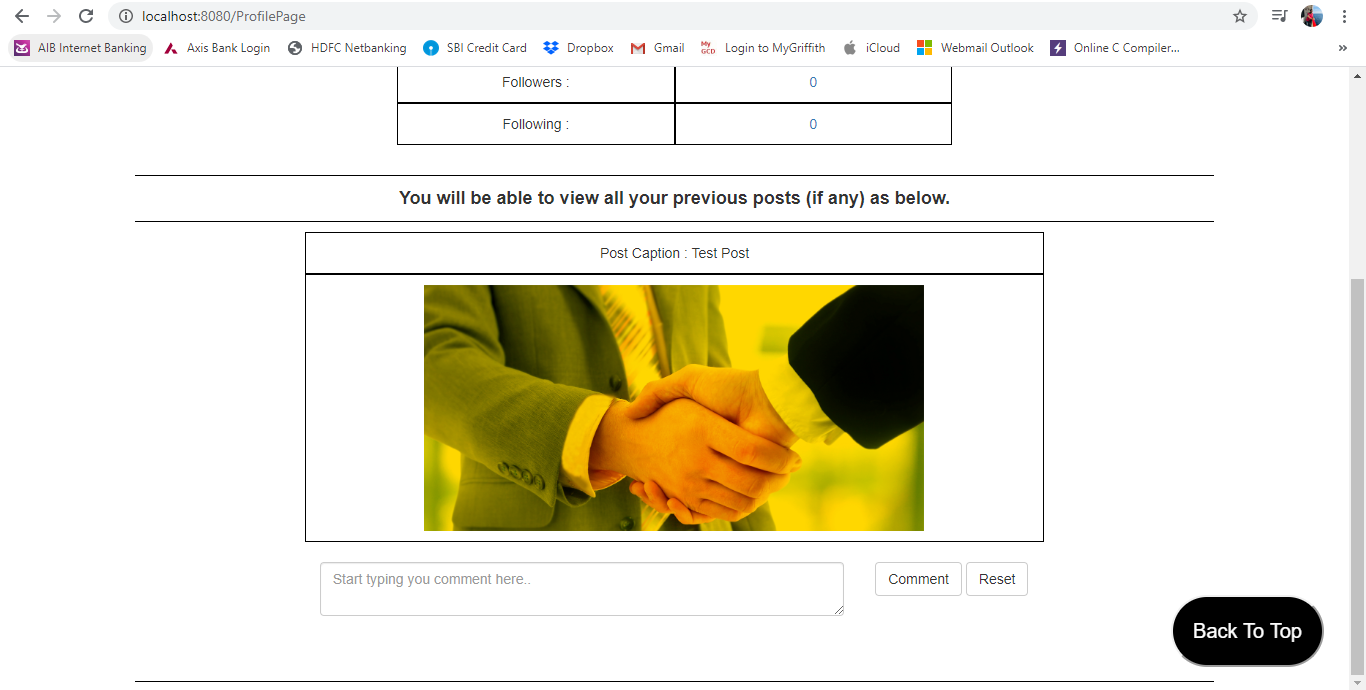
ProfilePage.html: Image 2



ProfilePage.html: Image 3



ProfilePage.html: Image 4.1



ProfilePage.html: Image 4.2

As show in above images, in case user logs in for first time, he will be forced to select username. After selecting username, page as per Image 3 will be displayed. If user have not created any posts yet, page in image 4.1 will be displayed and after creating at least 1 post, page in image 4.2 will be displayed.

**ProfilePage.py**

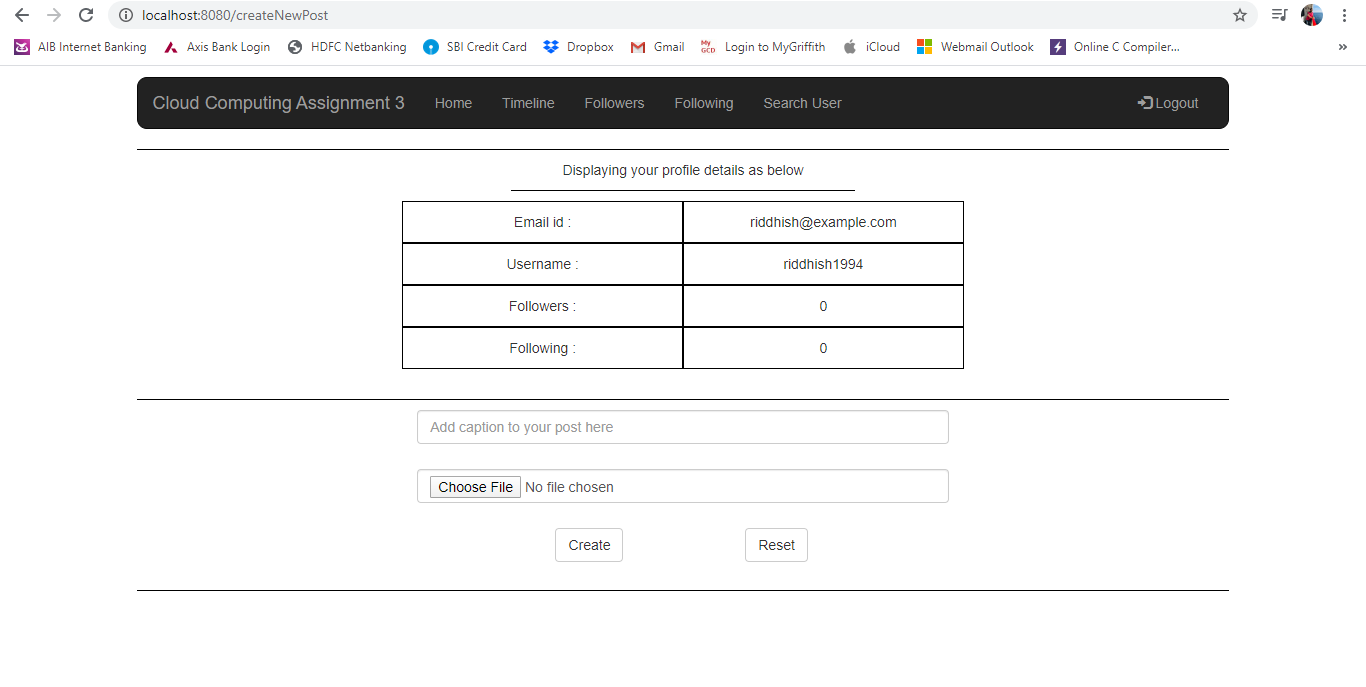
ProfilePage.py have 2 functions i.e. get and post function. Get function is called when user is redirected to this page and post function is called when any button is clicked on this page with method type post.

In get function, all the data of user, posts created by this user and comments for each post is pulled from datastore and passed on ProfilePage.html. There is a variable for notification being handled to display appropriate notification to user on performing appropriate actions. A variable is being used to handle if user have selected username or not. In case user have not selected username, he is forced to do so by restricting access to other functionalities and if username is already selected, he will be given access to all functionalities.

In post function, 2 logics are written. 1st logic is for users that login for first time. Users are forced to first select username. Hence, a logic to check if username already exists or not. If exists, user is not allowed to set that username and an appropriate notification is passed via URL. Else if username does not exist for any other user, same is set and user is directed to Profile page with appropriate notification message. 2nd logic is to handle comments on each post. When any user comments on any post, it is being added in CommentsDB.

**CreateNewPost.html**

CreateNewPost.html is next page of ProfilePage.html where user is redirected on clicking “Create Post” from navigation bar. This page allows user to create a new post as shown in Image 1.



CreateNewPost.html: Image 1

**CreateNewPost.py**

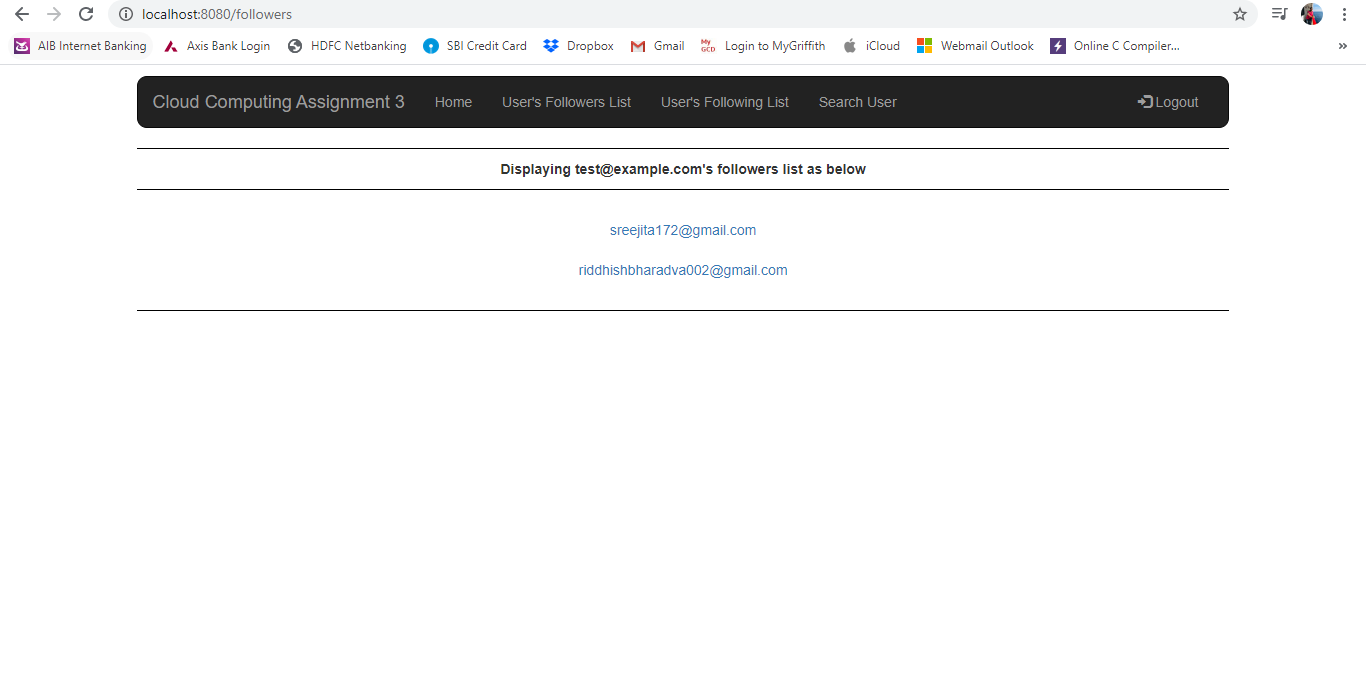
CreateNewPost.py have 2 functions i.e. get and post function. Get function is called when user is redirected to this page and post function is called when any button is clicked on this page with method type post.

In get function, initially it is checked if user is logged in or not. If not logged in (or if clicked on logout button from navigation bar), he will be redirected to ProfilePage else if logged in, all the user data like email id, username, number of followings, number of followers are pulled and passed on CreateNewPost.html page.

In post function, a logic is written to get post caption and image from html page and store it in PostsDB with appropriate key in PostsDB. Here, a condition to check if there is already a record for this user is check. In case this user has already created any post previously then a new post data will be appended else if not, a new record in datastore for this user will be created.

**Followers.html**

Followers.html is the page where user is redirected by clicking on Followers option from navigation bar. Here list of Followers for currently logged in user will be displayed and this list is hyperlink clicking on which user will be redirected to OtherUserProfilePage.html. Image 1 shows Followers.html page.



Followers.html: Image 1

**Followers.py**

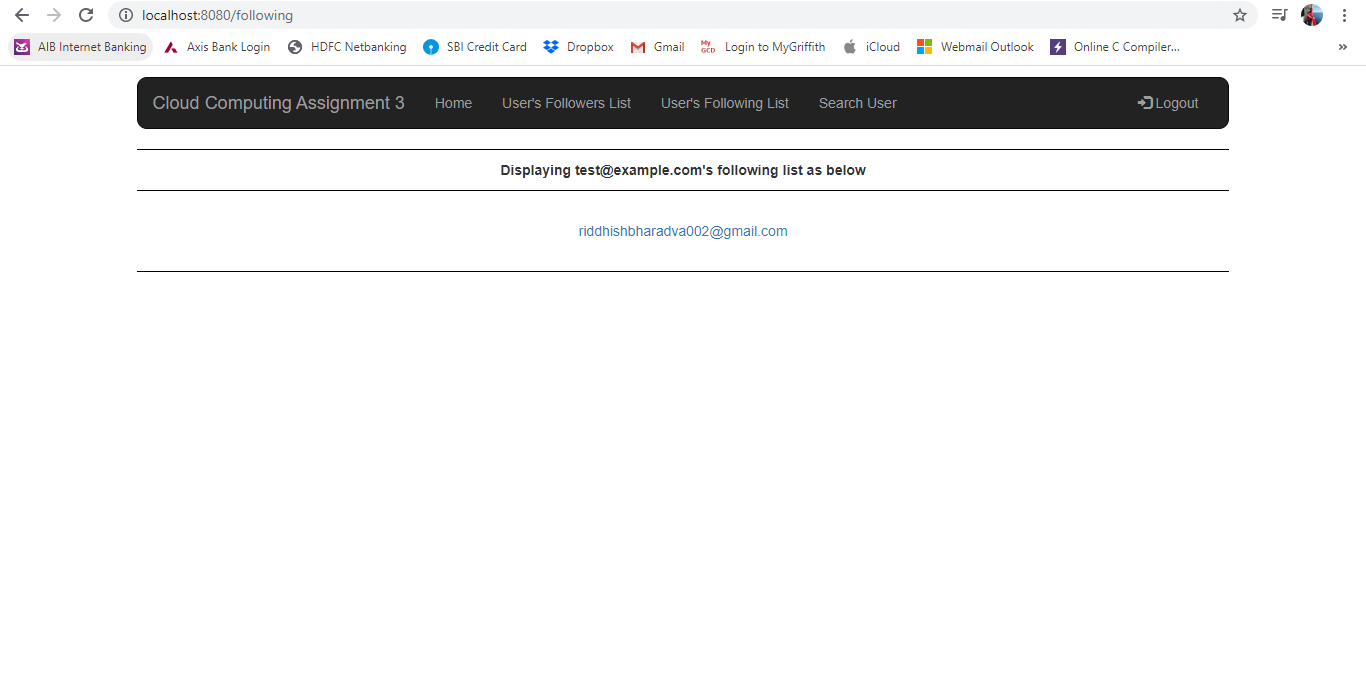
Followers.py have 2 functions i.e. get and post function. Get function is called when user is redirected to this page and post function is called when any button is clicked on this page with method type post.

In get method, initially it is checked if user is logged in or not. If logged in, all the followers data of currently loggedin user will be pulled from UsersDB and passed on to Followers.html page. In case user is not logged in (or clicks on logout button from navigation bar), he will be redirected to ProfilePage.html.

In post method, nothing is checked as there are no buttons to be handled on Followers.html page. Hence, user will be simply redirected to ProfilePage.html.

**Following.html**

Following.html is the page where user is redirected by clicking on Following option from navigation bar. Here list of Followings for currently logged in user will be displayed and this list is hyperlink clicking on which user will be redirected to OtherUserProfilePage.html. Image 1 shows Following.html page.



Following.html: Image 1

**Following.py**

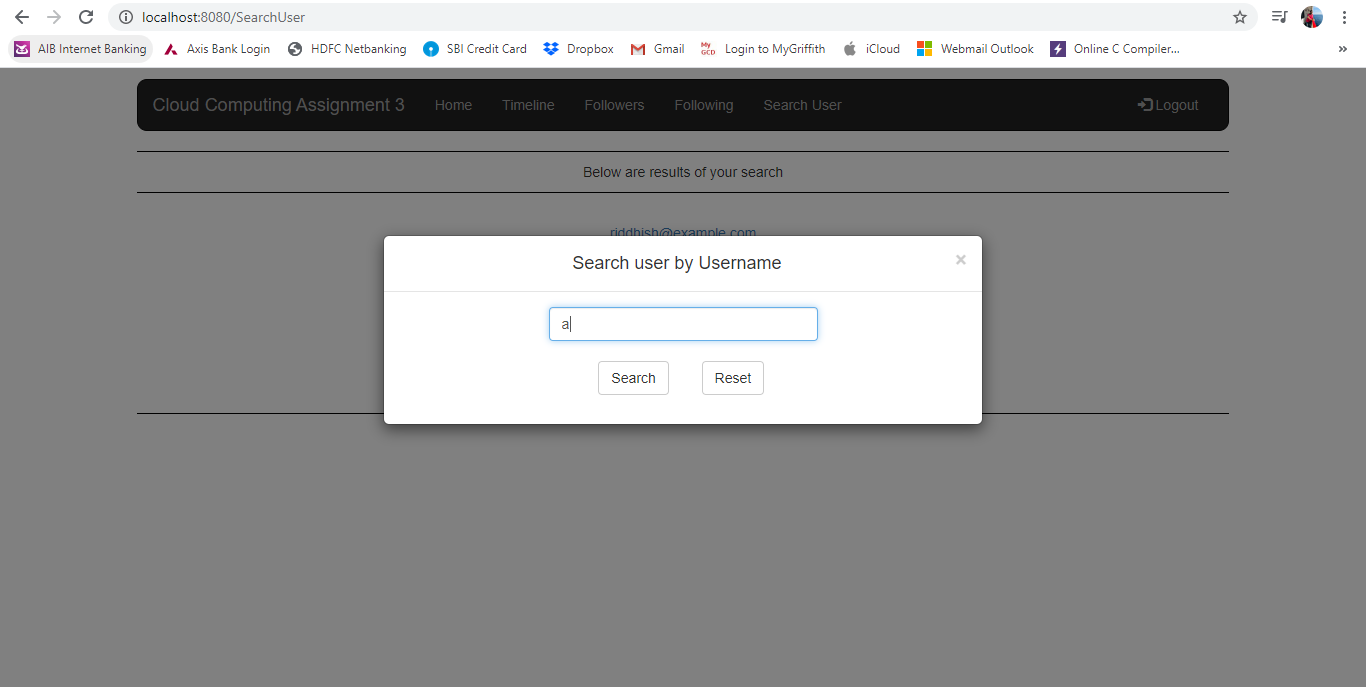
Following.py have 2 functions i.e. get and post function. Get function is called when user is redirected to this page and post function is called when any button is clicked on this page with method type post.

In get method, initially it is checked if user is logged in or not. If logged in, all the following data of currently loggedin user will be pulled from UsersDB and passed on to Following.html page. In case user is not logged in (or clicks on logout button from navigation bar), he will be redirected to ProfilePage.html.

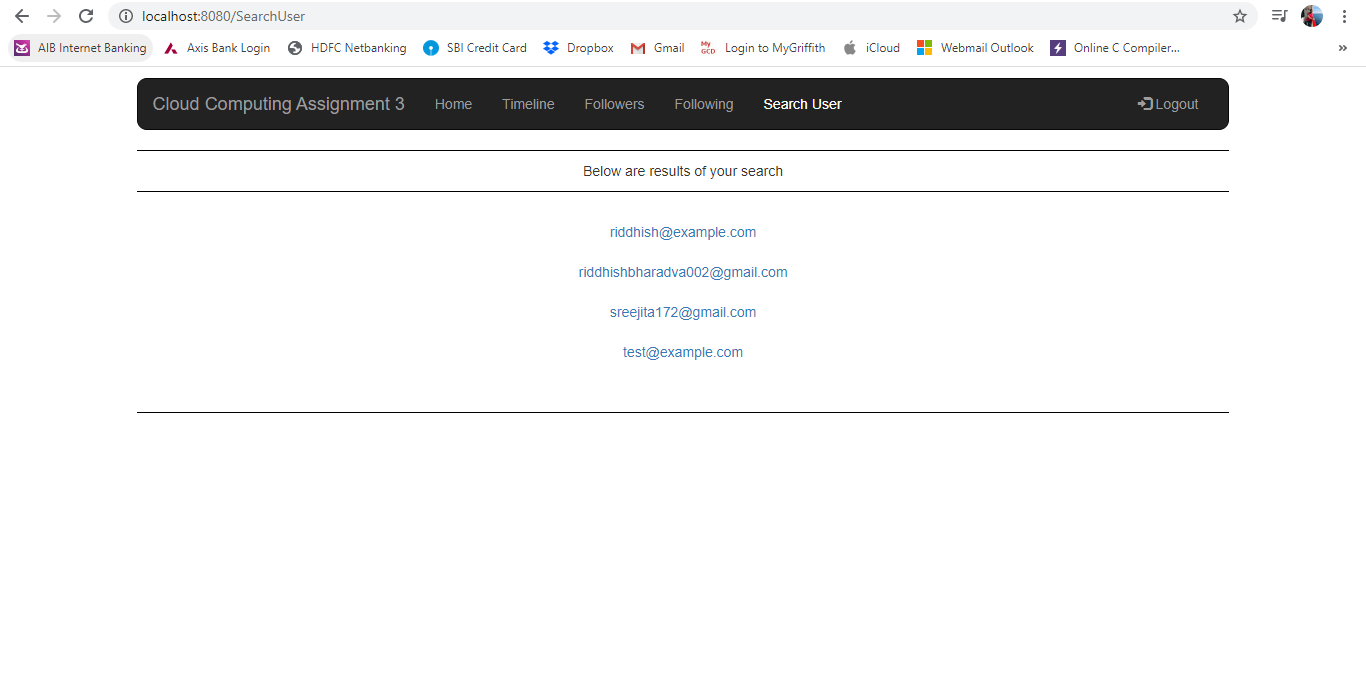
In post method, nothing is checked as there are no buttons to be handled on Following.html page. Hence, user will be simply redirected to ProfilePage.html.

**SearchUser.html**

SearchUser.html is the page where user is redirected by clicking on Search button from search model on each page. A Search option is given on navigation bar clicking on which a modal will be popup. Here list of user matching search criteria will be displayed on SearchUser.html page and this list is hyperlink clicking on which user will be redirected to OtherUserProfilePage.html. Image 1 and Image 2 shows search functionalities.



SearchUser.html: Image 1



SearchUser.html: Image 2

**SearchUser.py**

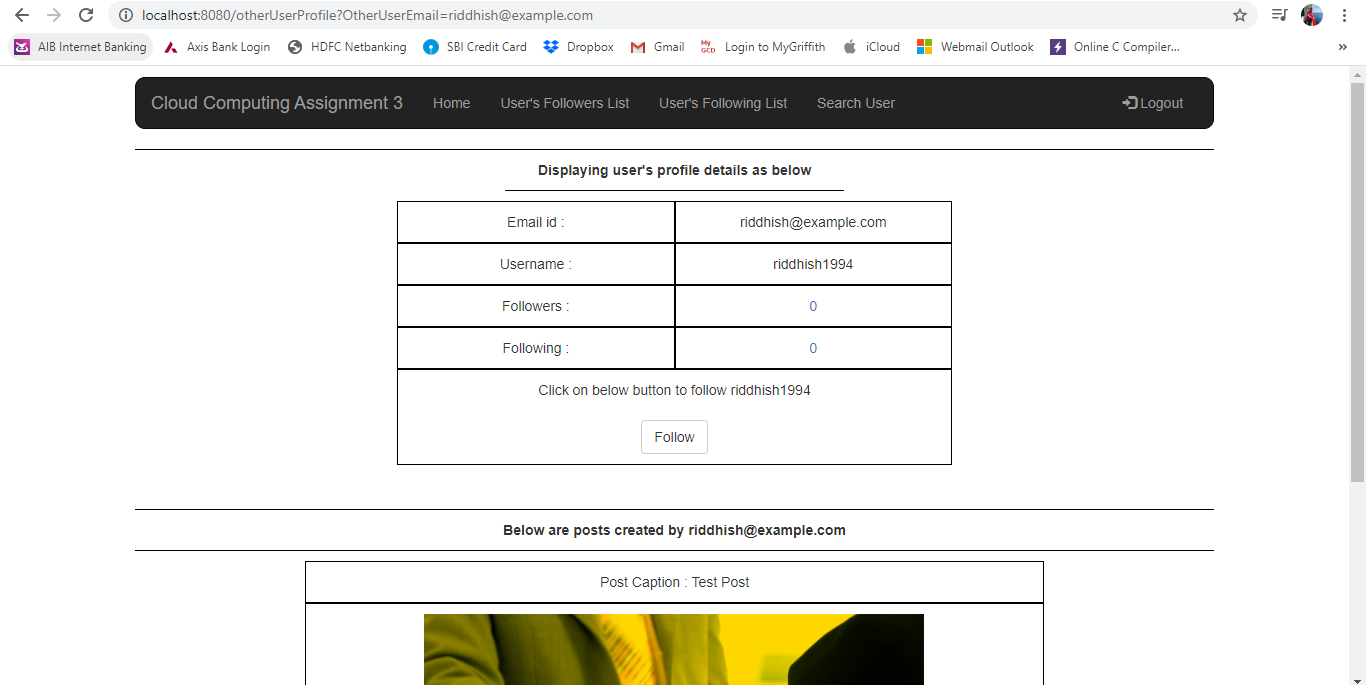
SearchUser.py have 2 functions i.e. get and post function. Get function is called when user is redirected to this page and post function is called when any button is clicked on this page with method type post or if user is redirected from any page to this page using method type post.

In get method, nothing is checked as there are no buttons to be handled on SearchUser.html page. Hence, user will be simply redirected to ProfilePage.html.

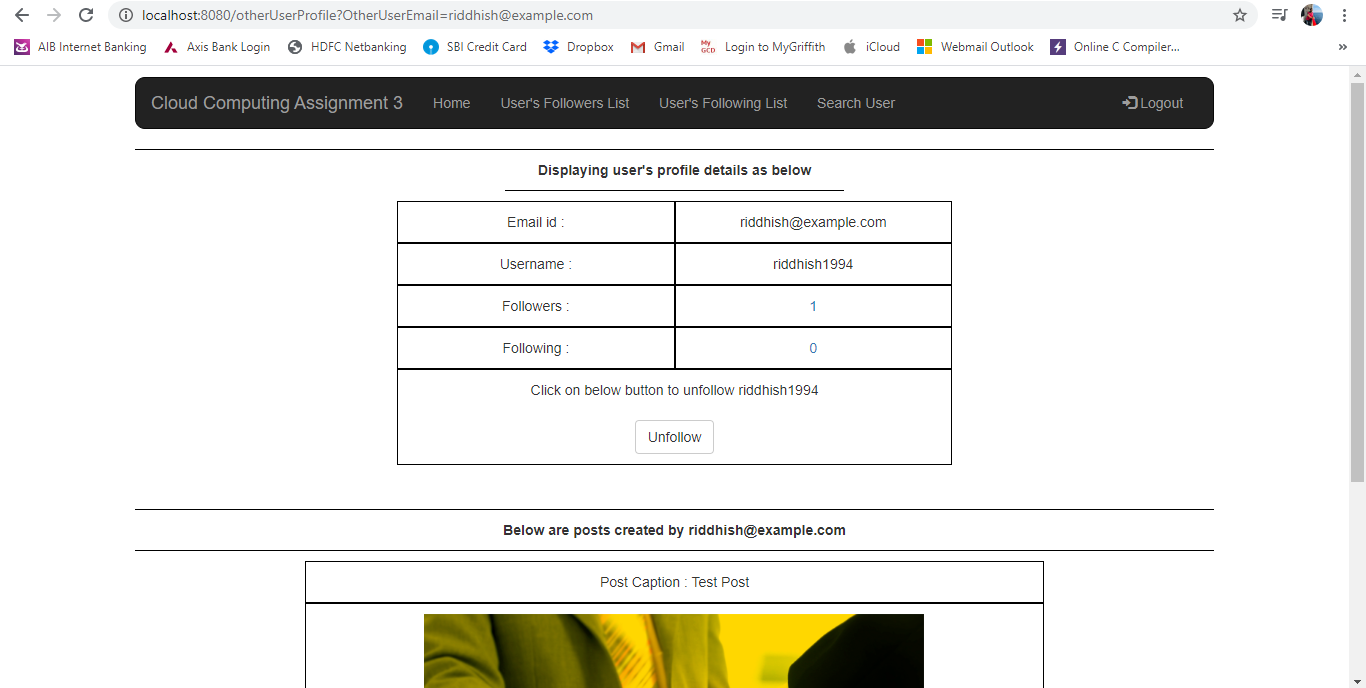
In post method, a logic is written to check if any user meets search criteria of user or not. First logic checks in user\_Email field of UsersDB in case no results are found, it will check in user\_Name field of UsersDB and result will be passed on to SearchUser.html page. In case no users have found, it will pass blank result and an appropriate message will be displayed on SearchUser.html page.

**OtherUserProfile.html**

SearchUser.html is the page where user is redirected by clicking on hyperlink of results on SearchUser.html or Followers.html or Following.html pages. This page is similar to that of ProfilePage.html. Only difference is Follow / Unfollow button displayed in table as shown in Images 1 and 2.



OtherUserProfile.html: Image 1



OtherUserProfile.html: Image 2

**OtherUserProfile.py**

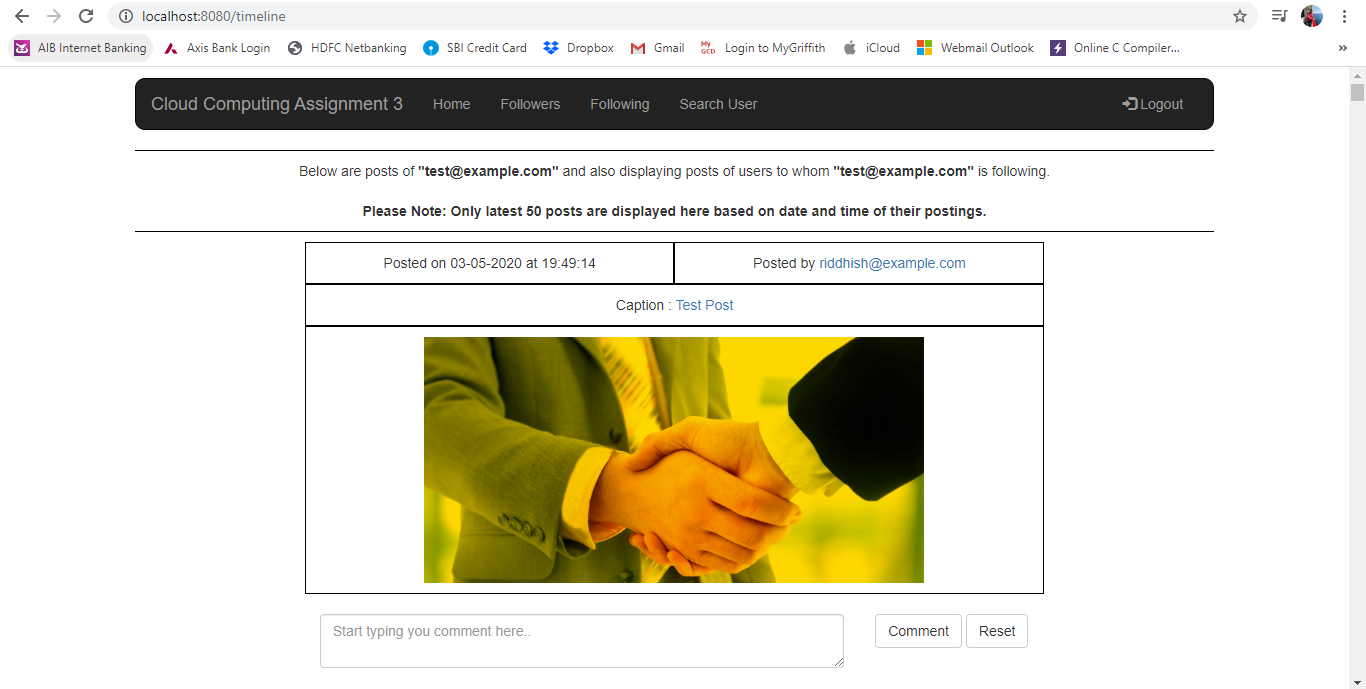
OtherUserProfile.py have 2 functions i.e. get and post function. Get function is called when user is redirected to this page and post function is called when any button is clicked on this page with method type post.

In get function, all the functionality is similar to that of ProfilePage.py except the Follow / Unfollow logic. A logic in get function checks if other user who’s profile page is visited is in following list of currently logged in user or not. If found or not found, appropriate variable value will be set to display appropriate Follow or Unfollow button on OtherUserProfile.html page. Another logic is written to check if Other user is same as currently logged in user, user will be redirected to ProfilePage.html page. This is because same user profile must not be able to follow itself.

In post function, a logic is written to follow and unfollow a user. When currently logged in user click on Follow button, email id of user who’s profile page is visited, is stored in followings list of currently logged in user and email id of currently logged in user is stored in followers list of user who’s profile page is visited. In case user clicks on Unfollow button, a logic is written to delete email of Other user from Following’s list of currently logged in user and also delete email id of currently logged in user from followers list of other user. Another logic to handle comment is written. This logic is similar to that on ProfilePage.py page.

**Timeline.html**

Timeline.html is next page where user is redirected on clicking “Timeline” from navigation bar. This page allows user to view posts of itself and also posts of users to whom currently logged in user is following. Image 1 shows design of Timeline.html page.



Timeline.html: Image 1

**Timeline.py**

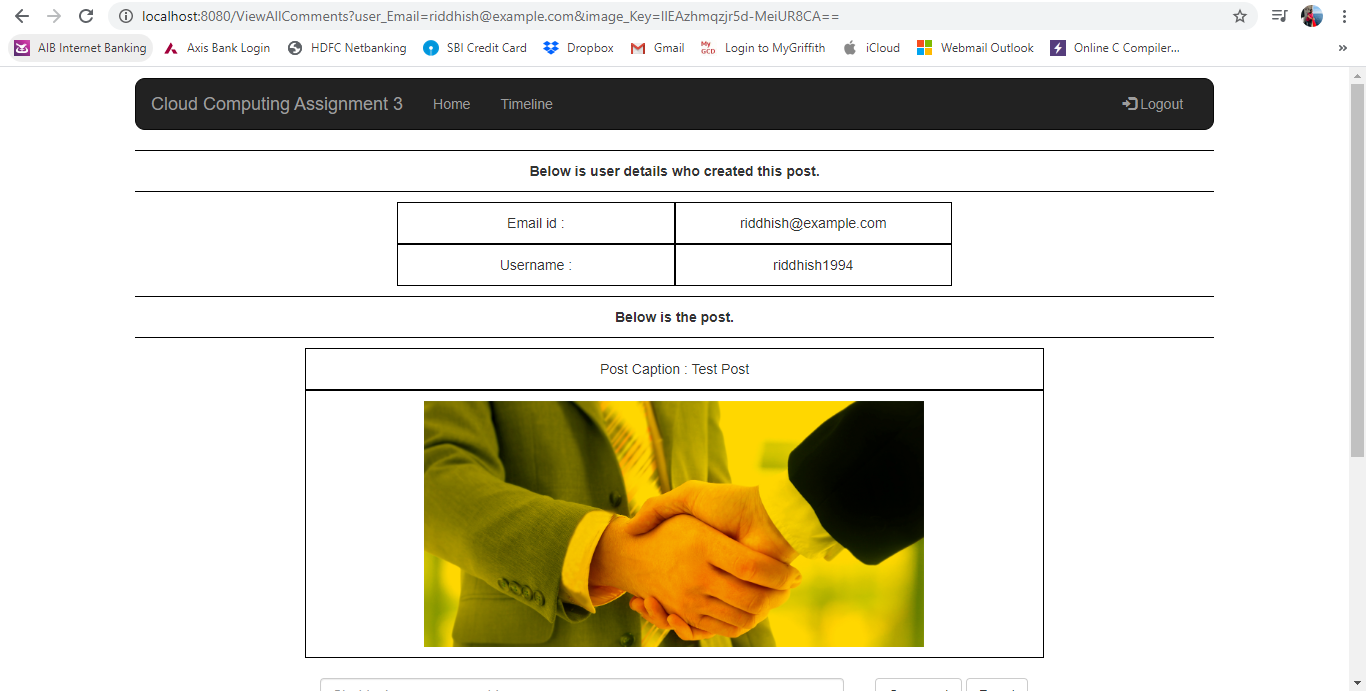
Timeline.py have 2 functions i.e. get and post function. Get function is called when user is redirected to this page and post function is called when any button is clicked on this page with method type post.

In get method, in order to display posts on timeline page, I have pulled all Date and Time of users followed by currently logged in user. This is because, combination of Date and Time in YYYYMMDDHHMMSS format acts as unique key as well. Due to this, it becomes easy for me to pull latest 50 posts based on date time. Based on these datetimes, I am pulling all the post data. I am running all loops on timeline page for just 50 iterations in case number of posts of currently logged in user and their followings have more than 50 posts. This prevents unnecessary pulling of my remaining posts from Database and prevents time to run loops unnecessary making my page easy to load.

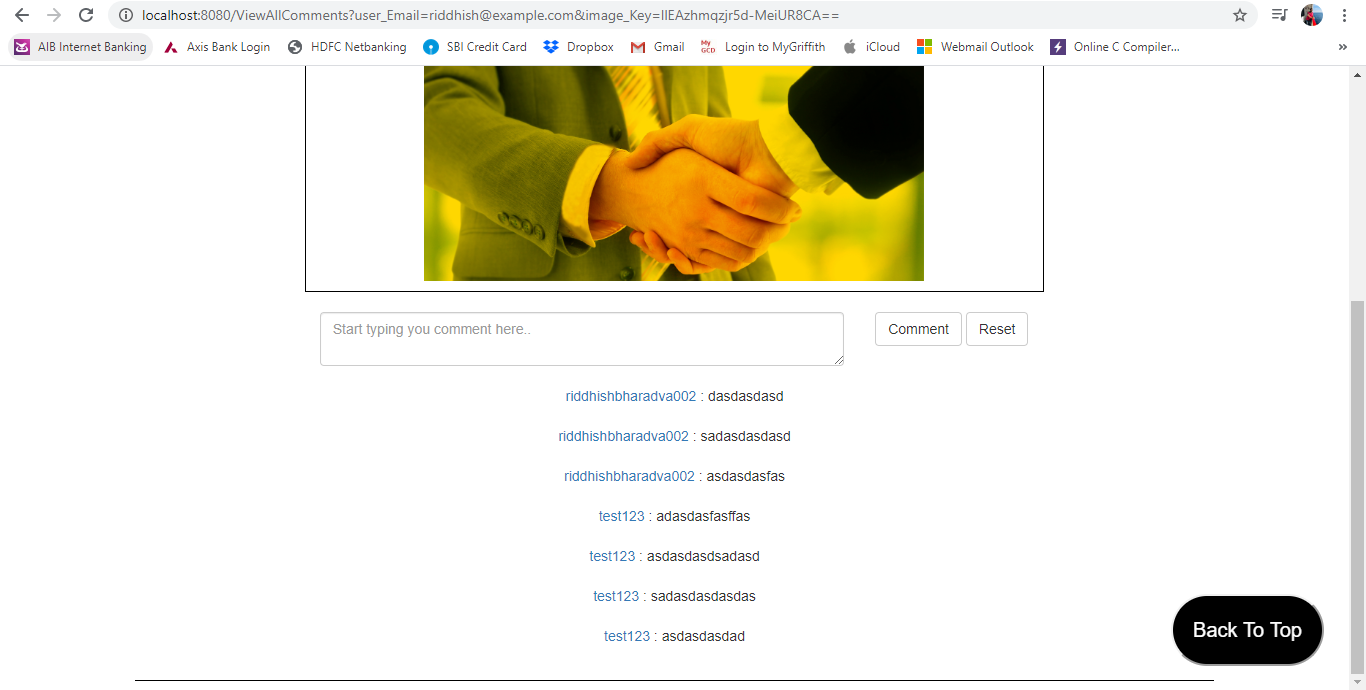
In post method, a similar logic is written to handle commenting of any user on post.

**ViewAllComments.html**

ViewAllComments.html is next page where user is redirected on clicking “Click here to view all comments” for any post having more than 5 comments. This page allows user to view posts and details of user who created this post along with all comments for this post. Image 1 and 2 shows design of ViewAllComments.html page.



ViewAllComments.html: Image 1



ViewAllComments.html: Image 2

**ViewAllComments.py**

ViewAllComments.py have 2 functions i.e. get and post function. Get function is called when user is redirected to this page and post function is called when any button is clicked on this page with method type post.

In get method, initially it is handled if user is logged in or not. If logged in, user will be able to view all details of the post. Else if not logged in, user will be redirected to ProfilePage.html. A logic is written to fetch all data regarding the post for which Blob Key is passed in URL in form of string. Using this blob Key all the data from CommentsDB is fetched and passed on to ViewAllComments.html page.

In post method, a similar logic is written to handle commenting of any user on post.

**Data structure & Models:**

I have created 3 models and 3 datastore tables in this project. The same are described as below:

Models:

1. UserDB
2. TaskBoardDB
3. TaskDB

Datastore Tables:

1. UsersDB
2. PostsDB
3. CommentsDB

In all the data models, I have imported ndb library to connect and interact with database.

1. UsersDB:

In this model, there is 1 datastore table declared i.e. UsersDB. UsersDB consist of Key, user\_Email, user\_Name, followers\_List and following\_List fields.

Key is default field in all the datastore tables created in google app engine.

1. user\_Email: This field is declared as String. This is because, email id can contain characters and numbers as well along with special characters like @,.,\_, etc.
2. user\_Name: This field is declared as String. This is because, username selected by any user can contain characters and numbers as well along with special characters like @,.,\_, etc.
3. followers\_List: This field is declared as String. This is because, this will store email ids of all users following logged in user and email id can contain characters and numbers as well along with special characters like @,.,\_, etc. This field have property of repeated = True. This is because there can be more than 1 user following logged in user.
4. following\_List: This field is declared as String. This is because, this will store email ids of all users followed by logged in user and email id can contain characters and numbers as well along with special characters like @,.,\_, etc. This field have property of repeated = True. This is because there can be more than 1 user followed by logged in user.
5. PostsDB:

In this model, there is 1 datastore table that contains all the details information about Posts created by logged in user. It contains of fields like Key, user\_Email, post\_Caption, post\_Image and post\_DateTime. All the fields except post\_Image and post\_DateTime are strings i.e. StringProperty. post\_Image is declared as BlobKeyProperty as this stores Blob keys of images uploaded in each post.

Key is default field in all the datastore tables created in google app engine. I am using email id of each user as key of this datastore table to uniquely pull data of posts for each user from this table.

1. user\_Email: This field is declared as String. This is because, email id can contain characters and numbers as well along with special characters like @,.,\_, etc. I am using email as key but still have this field to store email if of same user as I am fetching data on timeline page based on date and time from this table. Hence, this field is used to fetch data of user who have created this post.
2. post\_Caption: This field is declared as String. This is because, email id can contain characters and numbers as well along with special characters like @,.,\_, etc. This field have property of repeated = True. This is because there can be more than 1 post created by logged in user. Hence, there can be more than 1 caption for each user.
3. post\_Image: This field is declared as BlobKeyProperty. This is because, this field will be storing blob keys of all post images. This field have property of repeated = True. This is because there can be more than 1 post created by logged in user. Hence, there can be more than 1 image key for each user.
4. post\_DateTime: This field is declared as DateTimeProperty. This is because, this field will be storing date and time of all post created by user. This field have property of repeated = True. This is because there can be more than 1 post created by logged in user.
5. CommentsDB:

CommentsDB model consists of 1 datastore table CommentsDB. Fields in CommentsDB are Key, commenting\_User and comment.

Key is default field in all the datastore tables created in google app engine. I am using BlobKey of each post as key of this datastore table to uniquely pull comments for each post from this table.

1. commenting\_User: This field is declared as String. This is because, this field will be storing username of user commenting on any post and username can contain characters as well as numbers. This field have property of repeated = True. This is because there can be multiple users commenting on a single post. Hence, username of each user will be stored for each comment.
2. comment: This field is declared as String. This is because, this field will be storing comments being posted by user on any post and comments can contain characters, numbers and special characters. This field have property of repeated = True. This is because there can be multiple users commenting on a single post. Hence, comments by each of the user will be stored in this field.

**Design Decisions:**

1. In order to display posts on timeline page, I have pulled all Date and Time of users followed by currently logged in user. This is because, combination of Date and Time in YYYYMMDDHHMMSS format acts as unique key as well. Due to this, it becomes easy for me to pull latest 50 posts based on date time.
2. Based on these datetimes, I am pulling all the post data. In case 2 posts with same datetime is registered in database, which is possible in case of multiple user login at same time, I am skipping first post picked in loop in case loop is running for second time for same datetime. This is how duplicate posts are handled which makes my timeline correct.
3. I am running all loops on timeline page for just 50 iterations in case number of posts of currently logged in user and their followings have more than 50 posts. This prevents unnecessary pulling of my remaining posts from Database and also prevents time to run loops unnecessary making my page easy to load.
4. I have made a separate page for Creating a new post. This prevents creation of new sessions in Blob Upload Session unnecessarily. In case I keep creation of post on same page, a session will be created every time user refreshes that page or logs into his account.
5. On OtherUserProfile page, I am checking if other user email id passed in url is not same as currently logged in user. This makes it easy for me to not to allow users to follow themselves. In case both email ids are same, user is redirected to their own ProfilePage.html else they can proceed to follow user.