Link to GitHub repository:
(be sure to add https://github.com/cs3083fall2019 as a contributor. See instructions in Nov 24 announcement on NYU Classes)

Date of last commit to repository: 12/11/2019
Number of Team members: 3

Names and netIDs of Team members (one per line):
Sally Thompson sft259
Anuska Rungta ar5323
Jason Zilberkweit Jlz293

==
For each extra feature (i.e. the 2*n features beyond features 1 –5), include the following.

Please copy/paste the following template as many times as needed and then add your answers. Add page breaks between descriptions of different features. You should have at least 2*n pages where n is the number of people in your group.
When this is finished save it in a pdf file named *ProjectSummary_<your-name>.pdf* with the name of the person who will hand it in for the group. Then hand it in on GradeScope. Remember to use the group handin option if you have more than one person on your team.

1. Name of the feature
2. The full name of the team member who is primarily in charge of implementing this feature
3. The queries (and any other SQL statements) used in your implementation of the feature. (If there are standard queries used in most or all of your features, you don't have to include them here; just include the SQL statements that do the main work for this feature.)
4. A clear indication of where to find the application source code for the feature within your GitHub repository (filename and where to look in the file).
5. One or more screenshots or a short video demonstrating the feature, showing how it appears in the browser; Also show the relevant data that's in the database when you execute this demonstration (before and after if the feature changes the data), either as screenshots or as text. You may either include this in your GitHub and provide the link here or add screenshots here or on a separate page. Make it clear where the graders should look for this.
6. For features other than those I suggested, also include:
   i. What the feature is (in the style of the 6 requirements above)
   ii. A sentence or two on why this is a good feature to add to Finstagram
   iii. A clear explanation of any changes to the database schema that are needed (including additional tables, additional attributes, or additional constraints)

THE LINK TO THE VIDEO OF ALL OF THESE FUNCTIONS WORKING WILL BE IN THE README IN GITHUB!!!!

1. **View Visible Photos: Sally**
   The code for this can be found in init.py lines 118-135 in the home function.
   The SQL query that does this is:
   'SELECT * FROM Photo JOIN Person ON (photoPoster = username) WHERE photoID IN (SELECT photoID FROM Follow JOIN Photo ON (Follow.username_followed = Photo.photoPoster) WHERE allFollowers = 1 AND acceptedFollow = 1 AND username_follower = %s) OR photoID IN (SELECT photoID from Photo WHERE photoPoster = %s) OR photoID IN (SELECT photoID FROM SharedWith WHERE groupName IN (SELECT groupName FROM BelongTo WHERE member_username = %s OR owner_username = %s)) ORDER BY postingdate DESC'

   The html for this is found in home.html

2. **View further photo info: Anuska**
   The code for this can be found in init.py in the view_further_info function on lines 280-315.
   The SQL query that gets all of the photo info is:
   'SELECT firstName, lastName, postingdate, filepath FROM Photo JOIN Person ON Photo.photoPoster = Person.username WHERE photoID = %s'

   The SQL query that gets who is tagged in the photo is:
   'SELECT username, firstName, lastName FROM Tagged NATURAL JOIN Person WHERE photoID = %s AND tagstatus = 1'

   The SQL query that gets the likes/ratings of the photo is:
   'SELECT username, rating FROM Likes WHERE photoID = %s'

   The SQL query that gets the number of likes of the photo is:
   'SELECT COUNT(username) AS num_likes FROM Likes WHERE photoID = %s'

   The SQL query that gets all of the comments of the photo is:
   'SELECT username, commenttime, theComment FROM Comments WHERE photoID = %s'

   The html for this is found in view_further_info.html

3. **Post A Photo: Jason**
   The code for this feature can be found in init.py in the post function from lines 138-176.
   The SQL query that inserts the photo into the database is:
   'INSERT INTO Photo (photoPoster, filepath, caption, postingdate, allFollowers) VALUES(%s, %s, %s, %s, %s)'

   The SQL query that inserts the photo into the chosen friendgroup (if they choose to do that):
   'INSERT into SharedWith(groupName, groupOwner, photoID) VALUES(%s, %s, %s)'
   The html is in upload.html

4. **Manage Follows: Jason**
   The code for this feature is in init.py in the manage_follow_requests(), follow_accept(), and follow_decline() functions on lines 353-396. The SQL query that selects all of the accounts who submitted a follow request to the users account is:
   'SELECT * FROM Person WHERE username != %s AND username IN (SELECT username_follower FROM Follow WHERE username_followed = %s AND acceptedFollow = 0)'

   Follow_accept() allows users to accept a follow request using the following query:
   "UPDATE Follow SET acceptedFollow = 1 WHERE username_follower = %s AND username_followed = %s"

   Follow_decline() allows users to deny a follow request using the following query:
   "DELETE FROM Follow WHERE username_follower = %s AND username_followed = %s"

   The following had to be added to the Follow table in the database:
   acceptedFollow int

   The html for this can be found in manage.html, follow.html, and unfollow.html

5. **Like Photo: Sally**
   This feature allows users to both like and rate a photo! This is a great feature to add because how would anyone know who liked their photo/ what other people thought about it. No changes were made to the database. The code for this feature are from likes 200-224 in init.py in the functions likePhoto() and likedAlready(). likedAlready() is looking to see if a user has already liked a photo, as they can only like a photo once. The query for that is: "SELECT EXISTS(SELECT * FROM Likes WHERE photoID=%s AND username=%s) "

   The insert statement for entering a new like into the database is in the likePhoto() function and is: "INSERT INTO Likes (username, PhotoID, liketime, rating) values (%s, %s, %s, %s)"

   The html for this section can be found in home.html

6. **Unfollow: Jason**

   The unfollow feature allows users to stop following people that they are currently following. This feature is very important because if a user doesn't want to see someone's posts, they can now do that. Or maybe a user was stocking someone and accidently started following them, if they could not unfollow that would not be good!

   The code for this feature is on lines 398-425 in init.py in unfollow() and unfollow_action(). the query that selects all of the people you can unfollow Is in unfollow() and is:
   'SELECT * FROM Person WHERE username != %s AND username IN (SELECT username_followed FROM Follow WHERE username_follower = %s)'

   The query that actually unfollows a user is in unfollow_action() and is:
   "DELETE FROM Follow WHERE username_followed = %s AND username_follower = %s"

   The html is for this is in unfollow.html

7. **Add Comments: Sally**
The add comments feature is fantastic because users get the opportunity to share EXACTLY what they are thinking! Without add comments, how would a user be able to tell someone exactly what they think about their photo! Each person is only able to leave 1 comment to prevent any spamming. We don't want that.

The code for this feature is in init.py from lines 229-259 in the functions leaveComment() and alreadyCommented(). The SQL code that leaves a comment is in leaveComment() and is:
"INSERT INTO Comments (username, PhotoID, commenttime, theComment) values (%s, %s, %s, %s)"

If a user has already commented, the alreadyCommented() function will find it by using the following SQL query:
"SELECT EXISTS(SELECT * FROM Comments WHERE photoID=%s AND username=%s) "

The following table had to be added to our database to accommodate this feature:

```
Create Table Comments (

   username VARCHAR(20),

   photoID int,

   commenttime DATETIME,

   theComment VARCHAR(255),

   PRIMARY KEY(username, photoID),

   FOREIGN KEY(username) REFERENCES Person(username),

   FOREIGN KEY(photoID) REFERENCES Photo(photoID)

);
```

The html for this is in home.html

8. **Add Friend: Anuska**

   The add friend feature is great because it allows a user to add friends to their friend groups! A friend group isn’t a friend group without friends, so it is important to be able to use this feature. This feature is in init.py in the addFriend() function from lines 467-488.

   The insert statement for adding a friend is: 'INSERT INTO BelongTo(member_username, owner_username, groupName) VALUES (%s, %s, %s)'

   The html for this is in add_FriendGroup.html

9. Add Friend Group: Anuska

Add friend group gives users the opportunity to share their photos only with a particular group. This is a very nice feature because sometimes, you don't want to post a photo for all of your followers to see. You may only want a few of your friends to see a photo, and that is what add friend group allows you to do.

The code for this feature is in init.py on lines 443-465 in the add_FriendGroup() function. There are 2 query's that do this, the first creates the friend group and is:
'INSERT INTO Friendgroup(groupOwner, groupName, description) VALUES (%s, %s, %s)'

The other query adds the user to the friend group they just created.
'INSERT INTO BelongTo(member_username, owner_username, groupName) VALUES (%s, %s, %s)'

The html for this is in add_FriendGroup.html.

10. **Find User with Most Followers: Jason**

    This feature will tell you the name of the user who has the most followers on Finstagram. With this function, users don’t have to search to see who has the mot followers, they can just click a button and see who it is. Because we are creative, we titled this function who_is_king(), since whoever has the most followers is the KING of Finstagram.

    This function can be found in init.py on lines 427-440 in the who_is_king() function. The SQL that finds the user with the most followers is:
    "(SELECT username, Person.firstName, Person.lastName, count(username_follower) AS num_followers FROM Follow JOIN Person ON Follow.username_followed = Person.username GROUP BY username_followed, Person.firstName, Person.LastName ORDER BY count(username_follower) DESC LIMIT 1)"