Mobile App Development

Assignment 08

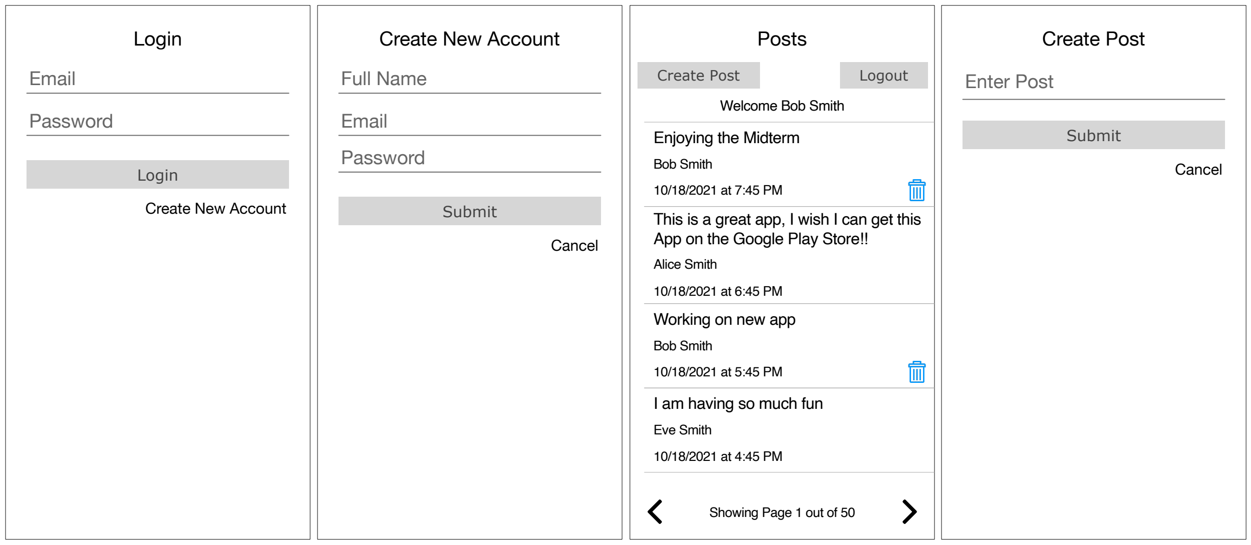
Basic Instructions:

1. In every file submitted you MUST place the following comments:
    a) Assignment #.
    b) File Name.
    c) Full name of the student.
2. Each group is required to submit the assignment on Canvas.
3. Submit Codes:
    a) Zip all the project folder to be submitted on canvas.
4. Submission details:
    a) The file name is very important and should follow the following format: **Assignment#.zip**
    b) You should submit the assignment through Canvas: Submit the zip file.
5. **Failure to follow the above instructions will result in point deductions.**

Assignment 8 (100 Points)

In this assignment you will develop a simple posts application, in which users can make short 140 character posts visible to other users on the app. You are provided with a Postman file that contains all the APIs for this app.

1. Use the OkHttp library in this app in order to make all the http connections and API calls. All the data returned by the APIs is in JSON format.
2. All the network calls should be done in a background thread.
3. All UI changes, updates and edits should be performed on the main thread.
4. This app will have one Activity and 4 fragments, all communication between fragments should be managed by the activity.



(a) Login **(b) Register** **(c) Posts** **(d) Create Post**

Figure 1, Application User Interface

Part 0: Checking User Authentication (10 Points)

You should use shared preferences to store and retrieve the authentication token and the user information. For information on shared preferences check the documentation https://developer.android.com/training/data-storage/shared-preferences. The requirements are as follows:

1. If the user has successfully logged in or registered, then the shared preferences should be used to store the retrieved information. Which implies that if the user has a token then they are authenticated.
2. When the Main Activity starts, you should check the shared preferences for the presence of the token and user information if present then the user has authenticated and the app should display the Posts Fragment. If the token and user information are not present in the shared preferences then the Login Fragment should be displayed.
3. Upon user logout the token and user information should be deleted from the shared preferences.

Part 1: Login Fragment (10 Points)

The interface should be created to match Figure 1(a). The requirements are as follows:

1. Upon entering the email and password:
   a. Clicking “Login” button, if all the inputs are not empty, you should attempt to login the user by using the **/posts/login** API.
   b. If login is successful, then communicate the returned and parsed authentication token and user information (See Figure 2) to the activity and **replace** the current fragment with the Posts Fragment. Store the token and user information to the shared preferences.
   c. If login is not successful, show an alert dialog showing the error message returned by the api.
   d. If there is missing input, show an alert dialog indicating missing input.
2. Clicking the “Create New Account” should **replace** this fragment with the Create New Account Fragment.

```
{
    "status": "ok",
    "token": “eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2MTc0MjgzMTUsImV..”,
    "user_id": 2,
    "user_fullname": “Alice Smith"
}
```

Figure 2, Highlighting the Auth Token returned by login and signup

Part 2: Create New Account Fragment (10 Points)

This fragment allows a user to create a new account. The interface should be created to match Figure 1(b). The requirements are as follows:

1. Upon entering the full name, email and password, clicking the Submit button should:
   a. If all the inputs are not empty, you should attempt to signup the user by using the **/posts/signup** API.
   b. If the registration is successful, then parse the returned response, and send the authentication token and user information to the activity. Store the token and user information to the shared preferences. **Replace** the current fragment with the Posts Fragment.
   c. If the registration is not successful, show an alert dialog showing the error message returned by the api.
   d. If there is missing input, show an alert dialog indicating missing input.
2. Clicking “Cancel” should **replace** this fragment with the Login Fragment.

```
//token received /posts/login or /posts/signup
String token = "eyJ0eXAiOiJKV1QiLCJhb......";
Request request = new Request.Builder()
        .url(postsUrl)
        .addHeader("Authorization", "BEARER "  + token)
        .build();
```

Figure 3, Code snippet showing how to add Authorization Header to Request

Part 3 : Posts Fragment (50 Points)

This screen enables the user to view the posts list. As shown in Figure 1(c), The requirements are as follows:

1. Clicking the “Logout” button should delete the authentication token and user information from the activity, then **replace** this fragment with the Login Fragment. Delete the token and user information from the shared preferences.
2. Clicking the “Create Post” button should **replace** the current fragment with the Create Post Fragment, and put the current fragment **on the back stack**.
3. The greeting TextView should show “Hello XX” where XX is the name of the logged in user. Note, this information should have been captured from the response of either the /posts/login or /posts/signup apis.
4. The list of posts should be retrieved be calling the **/posts** API. **Note that this API requires the Authorization header to include the token, please create the OkHttp request and include the header as shown in Figure 3.**
    a. The /posts api will return a single page of posts based on the provided “page” parameter, where each page includes 10 results. The api requires a “page” query parameter to indicate which result page is being requested. The page parameter starts from 1.
        i. The /posts api returns an array of posts based on the provided page parameter, in addition, the api will return the totalCount which is the total number of posts currently stored in the system.
        ii. Each post returned will include the post id, post text, post creation date/time, post’s creator id and name.
    b. Create a Post class, and parse the returned list of posts into an ArrayList containing the parsed Post objects. Use the parsed list of Post objects to display the posts list in RecyclerView.
    c. When the fragment first loads, the first page should be loaded by setting the page parameter to 1 when calling the /posts api.
5. For the posts list, each post row item should display the post text, creators name, creation date as shown in Figure 1(c). The trash icon should only be visible for post items that were created by the currently logged in user, you should use the user id to perform this comparison. For example, in Figure 1(c) the currently logged in user is “Bob Smith” who has created the first and third posts displayed.
    a. Clicking on the trash icon, the app should present an alert dialog asking the user if the selected post should be deleted. If the user picks “OK” the selected post should be deleted by using the **/posts/delete** api, note that this api requires the authorization header, see Figure (3) for reference. Upon successfully deleting a post item, the **/posts** API should be called to retrieve the latest list of posts and the posts list should be refreshed with the retrieved posts.
6. At the bottom of the fragment you should include the right, left and text to enable paging as shown in Figure 1(c).
    a. The text shows “Showing Page YY out of NN”, to indicate that the currently displayed page is page YY and there are a total of NN pages. The total number of pages should be calculated using the total number posts (totalCount) and the page length containing 10 posts (ceiling of totalCount/10). These values should be

updated whenever the **/posts** api is called.

b. Upon clicking on the right (>) button the **/posts** api should be called to retrieve the posts for the next page (current page + 1). Note that the pages start from 1 to totalCount/10, you should verify the next page number and avoid going beyond the total number of pages.

c. Upon clicking on the left (<) button the **/posts** api should be called to retrieve the posts for the previous page (current page - 1). Note that the pages start from 1 to totalCount/10, you should verify the previous page number is greater than or equal to 1.

d. Upon returning from the /post api the posts list be refreshed with the posts retrieved. In addition, the total number of pages should be computed using the returned totalCount. In addition, the "Showing Page YY out of NN" text should be updated based on the selected page and newly computed total number of pages.

Part 4 : Create Post Fragment (20 Points)

This screen enables the user to create a new post. The requirements are as follows:

1. Clicking the "Cancel" button should **pop the back stack** which should display the Post List Fragment.
2. Upon entering the post text, clicking the Submit button should:
   a. If all the inputs are not empty, a new post should be created using **/posts/create** api. Note that this api requires the authorization header, see Figure 3 for reference.
   b. If the api is successful, then **pop the back stack** which should display the Post List Fragment and should refresh the posts list to show the latest posts retrieved using the /posts api.