Mobile App Development
Midterm Exam

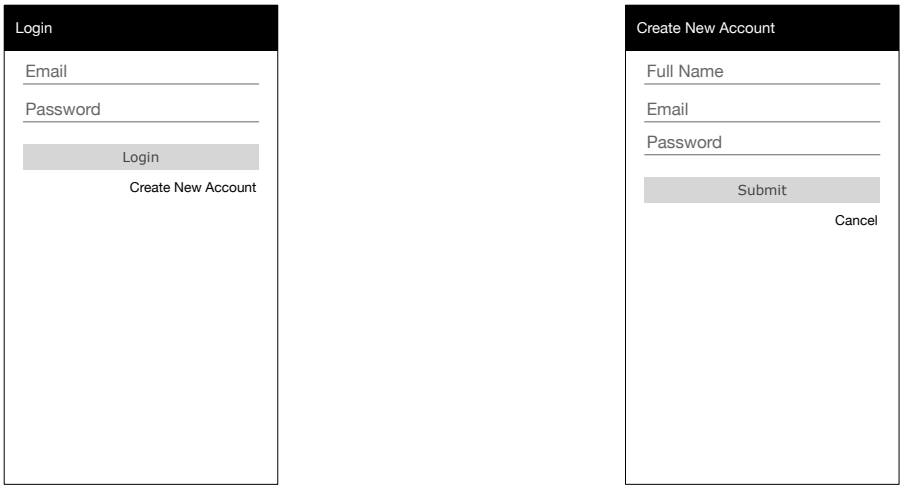
Basic Instructions:

1. This is the Midterm Exam, which will count for 15% of the total course grade.
2. This Midterm is an individual effort. Each student is responsible for her/his own Midterm and its submission.
3. Once you have picked up the exam, you may not discuss it in any way with anyone until the exam period is over.
4. During the exam, you are allowed to use the course videos, slides, and your code from previous home works and in class assignments. You can use the internet to search for answers. You are NOT allowed to use code provided by other students or solicit help from other online persons.
5. Answer all the exam parts, all the parts are required.
6. Please download the support files provided with the Midterm and use them when implementing your project.
7. Your assignment will be graded for functional requirements and efficiency of your submitted solution. You will loose points if your code is not efficient, does unnecessary processing or blocks the UI thread.
8. Create a zip file which includes all the project folder, any required libraries, and your presentation material. Submit the exported file using the provided canvas submission link.
9. **Do not try to use any Social Messenger apps, Emails, Or Cloud File Storage services in this exam.**
10. **Failure to follow the above instructions will result in point deductions.**
11. **Any violation of the rules regarding consultation with others will not be tolerated and will result disciplinary action and failing the course.**

Midterm Exam (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 Fragment (b) Register Fragment

Figure 1, Application Wireframe

Part 1: Login Fragment (10 Points)

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

1. This should be the launcher fragment that is displayed when the app starts.
2. 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 List Fragment.
   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.
3. 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 and **replace** the current fragment with the Posts List 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 List Fragment (60 Points)

This screen enables the user to view the posts list. As shown in Figure 4(a), 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.
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 4(a). 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 4(a) 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 a horizontal RecyclerView that will be used for paging as shown in Figure 4(a).
   a. The text above the RecyclerView 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. The RecyclerView should display all the page numbers available starting from 1 to the last page.
   c. Upon clicking on a page row item the **/posts** api should be called to retrieve the posts for the selected page. 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 and the pages RecyclerView should be updated to display the updated pages list. 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.

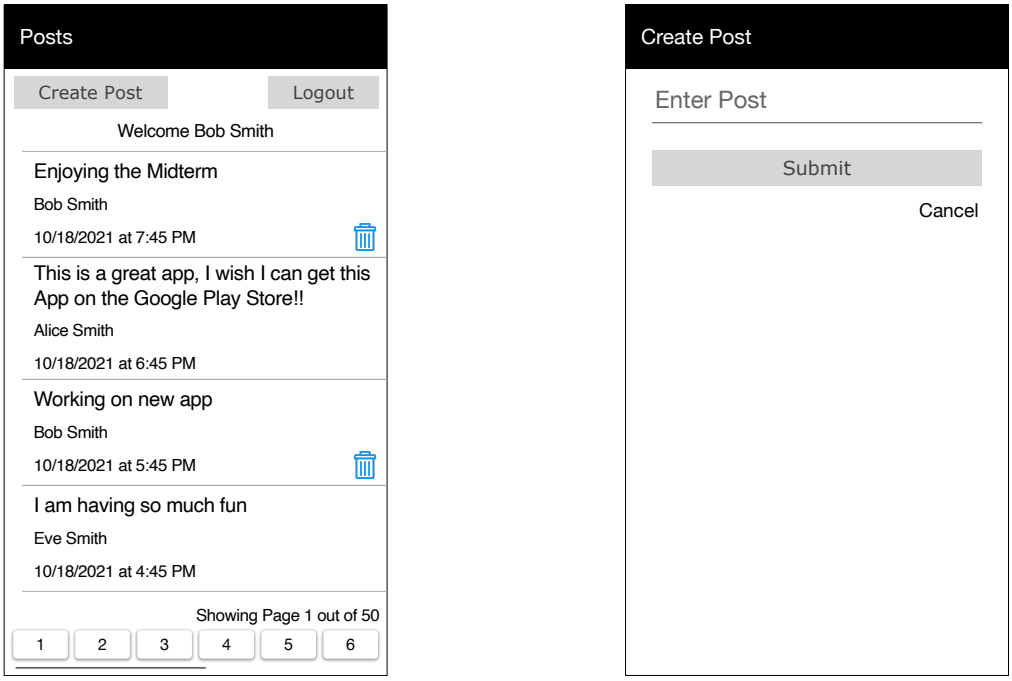
Part 4 : Create Post Fragment (20 Points)

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.



(a) Posts List Fragment

(b) Create Post Fragment

Figure 4, Application Wireframe

Item	Grade
Part 1 : UI, Error Validation, and Fragment Transitions	5
Part 1 : Multithreading, API call and parsing, and data passing to activity.	5
Part 2 : UI, Error Validation, and Fragment Transitions	5
Part 2 : Multithreading, API call and parsing, and data passing to activity.	5
Part 3 : UI, and Fragment Transitions (logout and create post)	5
Part 3 : Data passing and display Welcome message	5
Part 3 : Posts List display, event handling, refreshing and API call and parsing	20
Part 3 : Posts List delete post API call and parsing, posts list reloading.	15
Part 3 : Pages List display, event handling, refreshing and API call and parsing	15
Part 4 : UI, Error Validation, and Fragment Transitions	10
Part 4 : Multithreading, API call and parsing, and data passing to activity.	10
Total	**100**