ITIS/ITCS 4180/5180 Mobile Application Development

Midterm

In-Class Programming Assignment

Basic Instructions:

---

1. This is the Midterm Exam, which will count for 25% of the total course grade.
2. In every file submitted you MUST place the following comments:
   a. Midterm.
   b. File Name.
   c. Your Full Name.
3. This Midterm is an individual effort. Each student is responsible for her/his own Midterm and its submission.
4. Once you have picked up the exam, you may not discuss it in any way with anyone until the exam period is over.
5. During the exam, you are allowed to use the course videos, slides, and your code from previous homeworks and in class assignments. You are NOT allowed to use code provided by other students or from other sources.
6. This exam is composed of 3 parts, answer all of the parts.
7. Please download the support files provided with the Midterm and use them when implementing your project.
8. **Export your Android project as follows:**
   a) From eclipse, choose "Export…" from the File menu.
   b) From the Export window, choose General then File System. Click Next.
   c) Make sure that your Android project for this Midterm is selected. Make sure that all of its subfolders are also selected.
   d) Choose the location you want to save the exported project directory to. For example, your Desktop or Documents folder.
   e) The first time you export, make sure you select Create directory structure for files.
   f) Click Finish, and then go to the directory you exported the project to. Make sure the exported directory contains all necessary files, such as the .java and resource files.
9. Submission details:
   a) When you submit the Midterm, compress your exported Android project into a single zip file. The format of compressed file name is:
      i. Midterm.zip
   b) You should submit the Midterm through Moodle:
      i) Submit the zip file.
10. **Failure to do the above instructions will result in loss of points.**
11. **Any violation of the rules regarding consultation with others will not be tolerated and will result disciplinary action and failing the course.**

Midterm (100 Points)

In this assignment you will develop an app based on the 500px api and SQLite.

Important App Requirements:

Points will be deducted if your application does not follow the below requirements:

1. The required Android Virtual Device (AVD) should have **minimum SDK version set to 14 and target SDK at least 17**. The app should display correctly on 3.2" QVGA (ADP2) (320x480: mdpi). Your assignment will not be graded if it does not meet these requirements, and you will not be granted any points on your submission.
2. All API calls, JSON parsing and image downloading should be performed using Threads (or AsyncTask) and your code should not block the main thread.
3. Your code should use standard naming conventions, such as, uppercase class names, and lower case variable/method names. Also your variable and method names should be descriptive of the data or action performed.
4. All AsyncTask/Runnable and Parsing Classes should not be implemented as inner classes in an Activity instead should be implemented as separate files.
5. Your implementation should target the most efficient algorithms and data structures. You will be graded based on the efficiency of your implementation.

Initial Setup:

1. Go to https://developers.500px.com/ and create a new account.
2. After creating an account, you will receive an email from 500px you need to confirm your account by clicking on the link in the received email.
3. Go to https://developers.500px.com/ click on the *"**register your application**"* at the bottom of the page.
   a. Name your application "Midterm App" and provide a short description.
   b. Set the application url to www.uncc.edu/~your-uncc-id
   c. Provide your email as the developer email.
4. The figure below shows the application information, copy the application "Consumer Key", which is required by the photo search api.

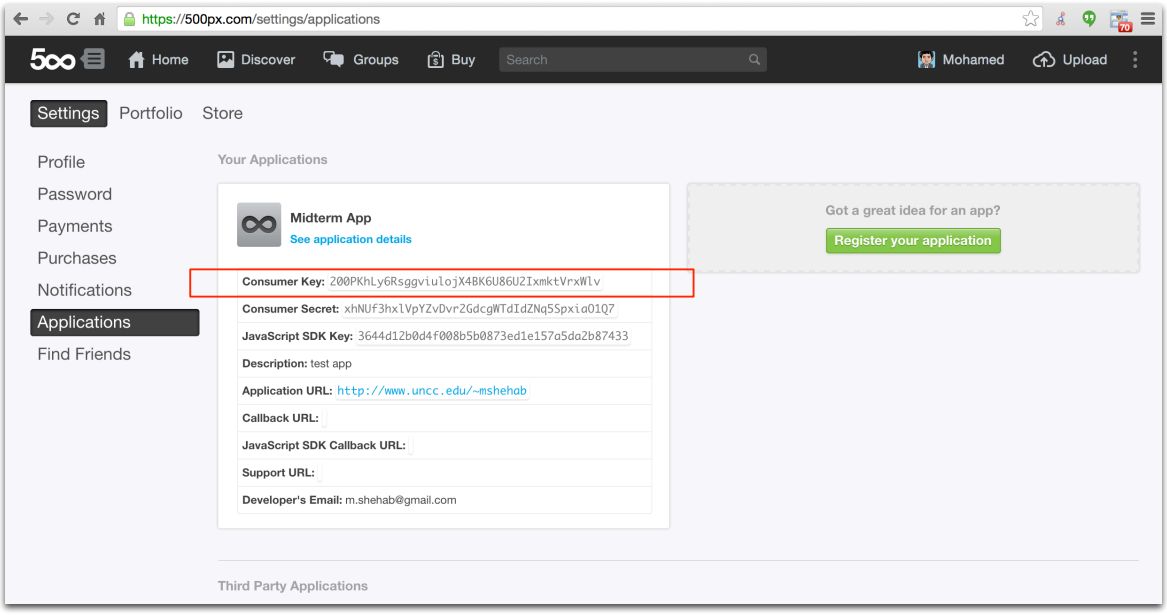


Figure 1, 500px Application Information

Part A: Main Activity (30 Points), (Estimated Time: 60 Minutes)

The Main activity displays a EditText that enables the user to enter a keyword search term. In addition, the user is able to retrieve the previously entered keyword search items. Figures 2 shows the Main activity. The requirements include:

1. The keyword search term can be entered directly in the provided EditText or a perviously used keyword search item can be used.
2. Clicking the image beside the EditText should display the search history alert dialog. The search history should be stored and retrieved from a local SQLite table. The search history results should be presented in ascending order based on the search keyword term entry date and time, such that the recent search terms should be at the top and the older terms at the bottom of the list. Upon selecting a keyword from the list, the EditText text should be set to the selected keyword, See Figure 2.
3. Upon clicking the submit button:
    a. Store the keyword term in the local SQLite history table.
    b. Start the Gallery Activity. The intent should include the search term using extras. You will not be given any points if you use a static variable to share data between activities.

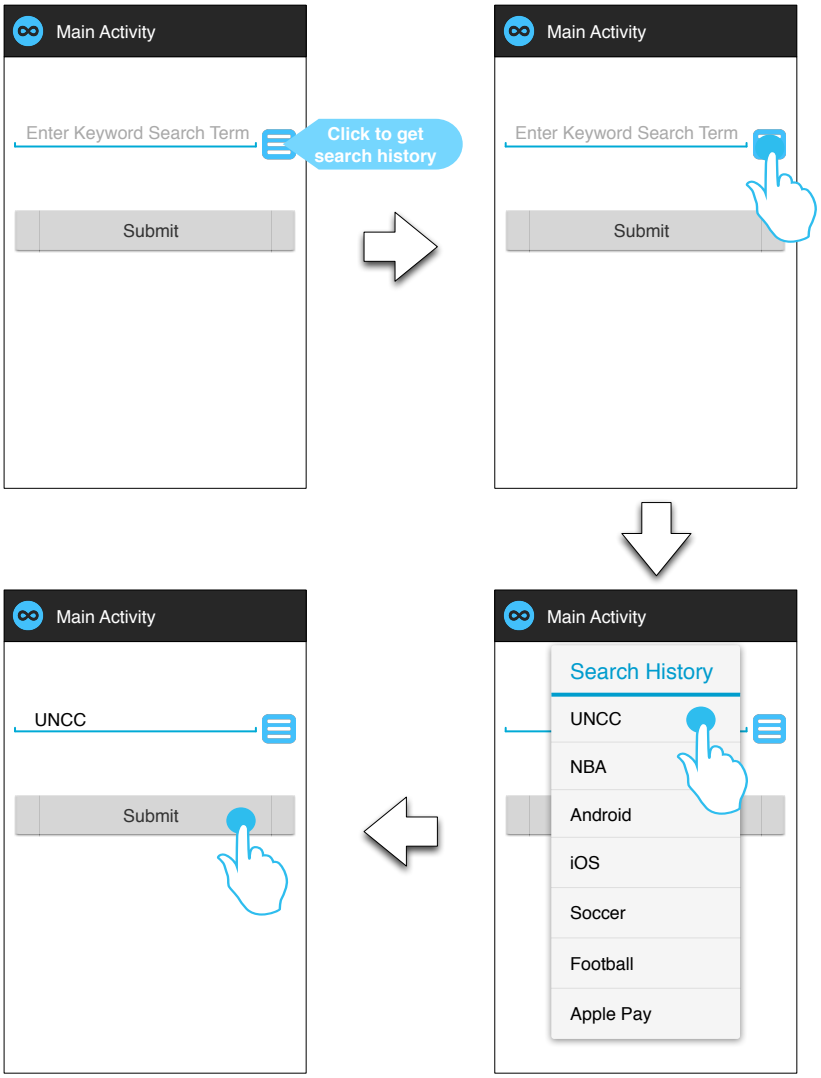


Figure 2, The Main Activity Wireframe

Part B: Gallery Activity (50 Points), (Estimated Time 90 Minutes)

The Gallery activity is responsible for the retrieving the photo search results based on the search keyword term sent from the Main Activity. The Gallery Activity wireframe is shown in Figure 3. The requirements include:

1. If the search term keyword is a new keyword (*not present in local SQLite table*) then:
    a. The 500px photos/search api should be used to retrieve the photos matching the requested keyword term:
        - Use the api documentation provided in https://github.com/500px/api-documentation/blob/master/endpoints/photo/GET_photos_search.md
        - You should set the "**consumer_key**" parameter based on your 500px app's consumer key.
        - You should set the "**term**" parameter to the search term keyword.
        - You should set the "**image_size**" parameter to 4.
        - You should set the "**rpp**" parameter to 50.
        - Your URL should be constructed using the required **URL encoding**.
    b. Send a GET request and retrieve the photo search results for the selected keyword term. All the parsing and HTTP connections should be performed by a worker thread or an AsyncTask and should not be performed by the main thread. While, the JSON is being downloaded and parsed you should display a progress dialog as indicated in Figure 3.
    c. You should parse the JSON string provided by the API and should not use the 500px Android SDK. You will not be granted any points if you use the 500px Android SDK.
    d. Store the retrieved photo item results in a local SQLite table and associate these results with the requested search term keyword.
    e. The progress dialog should be dismissed after the parsing is completed, SQLite table storing and the GridView is setup and displayed.
2. If the search term is a perviously used keyword (*present in local SQLite table*) then:
    a. The local SQLite table should be used to retrieve the photos search results associated with the requested keyword term.
    b. While, the photos information is being loaded from the local SQLite table you should indicate that using the progress dialog as indicated in Figure 3.
    c. The progress dialog should be dismissed after the photo items are retrieved from the local SQLite table, the GridView is setup and displayed.
3. The GridView items should be displayed to include the photo thumbnail, and the photo name, as shown in Figure 3. For more information related to GridView refer to: http://developer.android.com/guide/topics/ui/layout/gridview.html
    a. You should create your custom adapter and a custom layout for the GridView items. You should reuse the views provided by the GridView adapter from the (scarp) recycle pool and should provide the synchronization needed to ensure the loading of the correct thumbnails.
    b. The images should be retrieved by the adapter using a separate thread or AsyncTask. The images should be downloaded only when needed, your implementation should not pre-download all the images. At any given time only maintain a reference to the displayed photo. Do not store the downloaded

images in memory as this is not efficient and will crash your application. Hint: You can use the Picasso library http://square.github.io/picasso/ .

4) Clicking a photo item should display the photo details by starting the Details Activity. The intent should include the required photo information to be sent to the Details activity using extras. You will not be given any points if you use a static variable to share data between activities.

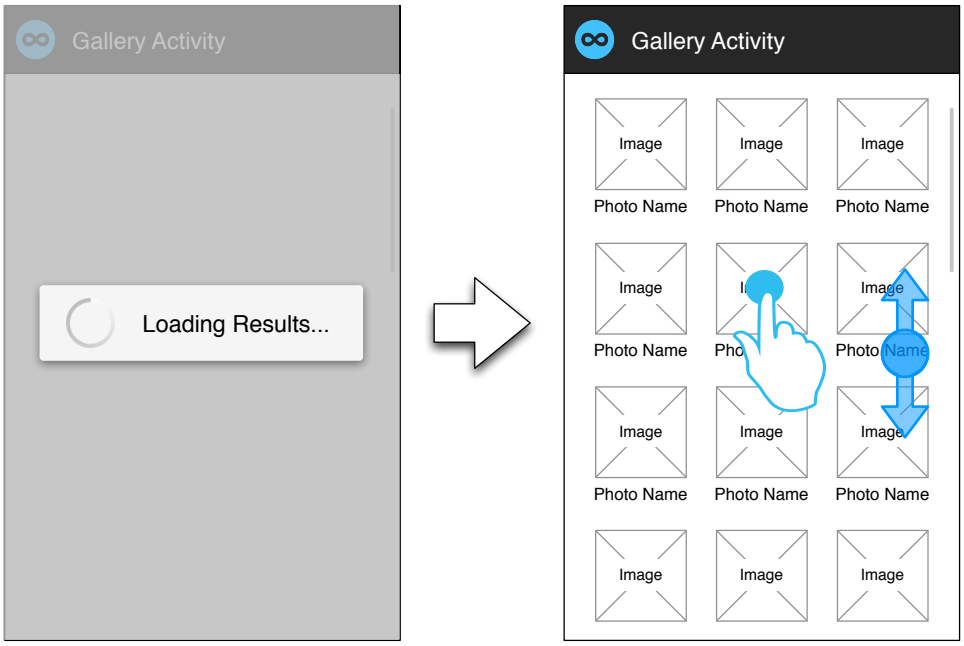


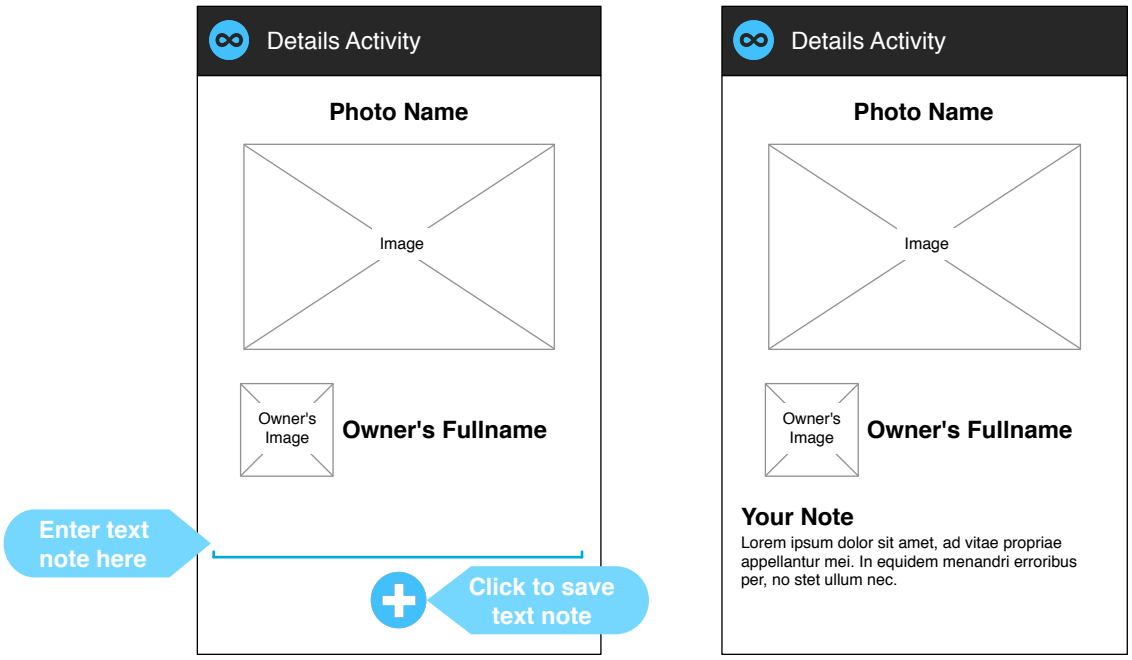
Figure 3, The Gallery Activity Wireframe

Part C: Details Activity (20 Points), (Estimated time 30 Minutes)

The Details activity will display more details related to the photo selected in the Gallery activity. The Details activity wireframe is shown in Figure 4. The requirements include:

1) The photo details displayed include, photo name, the photo's image, the owner's full name, and the owner's thumbnail image. In addition, the Details activity allows the user to add a single note to the image to be stored in a local SQLite table.
2) Figure 4(a), shows the Details activity for an image which does not have an associated note stored in the local SQLite table. The EditText should enable the user to enter a note, and clicking the plus sign under the EditText should store the entered note in a local SQLite table, and associate this record with the current photo item. Then the view should be changed to display a label and the note text as shown in Figure 4(b).
3) Figure 4(b), shows the Details activity for an image which does have an associated note stored in the local SQLite table. The note is retrieved from the SQLite table and displayed as indicated in Figure 4(b).
4) Loading the photo's image and the owner's image should not be performed by the main thread, you should use another thread or AsyncTask. If there is no photo image or owner image, use the provided "photo_not_found.png" and "user_not_found.png" respectively. The images should be downloaded only when needed, your

implementation should not pre-download all the images. At any given time only maintain a reference to the displayed photo. Do not store the downloaded images in memory as this is not efficient and will crash your application. Hint: You can use the Picasso library http://square.github.io/picasso/.



(a) Photo with no stored note (b) Photo with stored note

Figure 4, The Details Activity Wireframe