ITIS/ITCS 4180/5180 Mobile Application Development
Midterm
Starts: 03/21/2016 at 18:30
Ends: 03/21/2016 at 21:30

Basic Instructions:

1. This is the Midterm Exam, which will count for 20% of the total course grade.
2. In every file submitted you MUST place the following comments:
   a. 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 home works and in class assignments. You are NOT allowed to use code provided by other students or from other sources.
6. Answer all the exam parts, all the parts are required.
7. Please download any support files provided with the Midterm and use them when implementing your project.
8. Your assignment will be graded for functional requirements and efficiency of your submitted solution. You will lose points if your code is not efficient, does unnecessary processing or blocks the UI thread.
9. Export your Android project and create a zip file which includes all the project folder and any required libraries. The file name is very important and should follow the following format: **Midterm.zip**
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 a venue information application using Foursquare API. The application should be able to load different venues into a list and user should be able to see each venue in a detailed activity. Users of the application will also be able to mark a venue for future reference.

API Setup

1. Go to https://foursquare.com, and create a new account
2. Go to https://developer.foursquare.com, and then click "Get Started"
3. Click on create an app
4. Fill all the information needed, and then click "Save changes"
5. This will create 2 parameters **`Client ID`** & **`Client Secret`**. These parameters will be used later in the search API.

Part 1: Main Activity (20 points):

The Main activity displays a Spinner that enables the user to select a City. The Activity UI should match the UI presented in Figure 1(a) and below are the requirements:

1. Spinner should have a Dropdown with the list of following cities: **"Chicago, IL", "Atlanta, GA", "Charlotte, NC", "San Francisco, CA", "Seattle, WA", "Orlando, FL"**
2. Upon clicking the submit button start the Venues Activity. The intent should include the City name using extras. You will not be given any points if you use a static variable to share data between activities

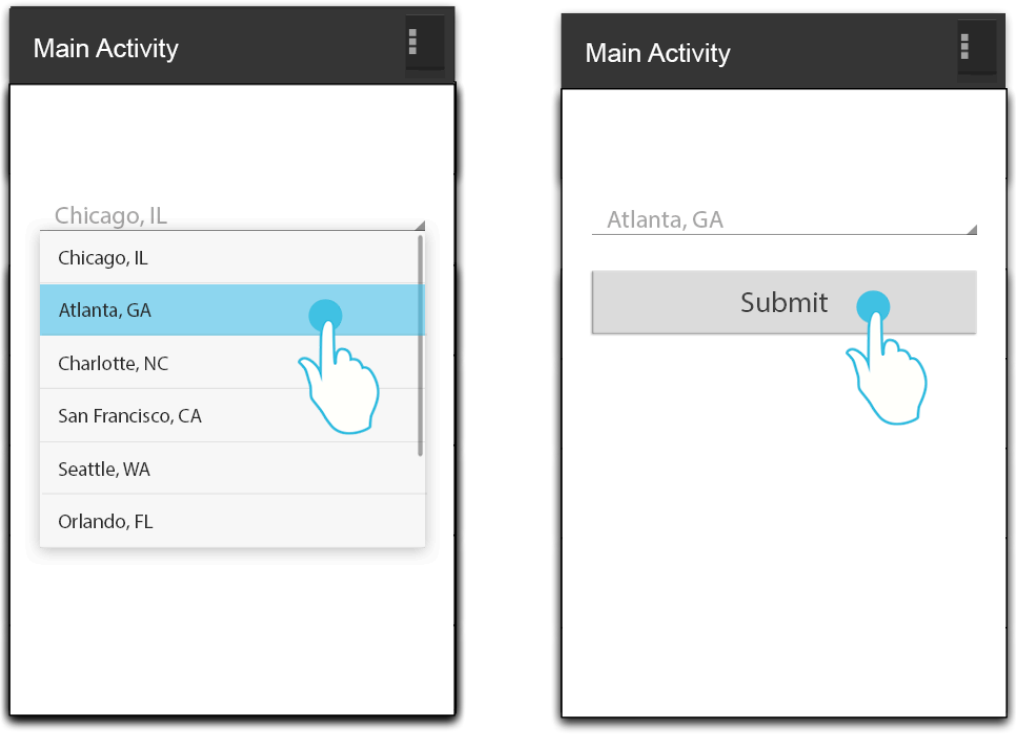

Figure 1 . The Main Activity Wireframe

Part 2: Venues Activity (80 points):

The Venues activity is responsible for retrieving the Venue search results based on the city/ state values sent from the Main Activity. You will use search API to get a list of venues.

1. The Foursquare Search Venue API should be used to retrieve the results matching the section. The API follows the following format:

```
https://api.foursquare.com/v2/venues/search?client_id
=CLIENT_ID&client_secret=CLIENT_SECRET&v=20140806&nea
r=CITY,STATE
```

Parameters:

- Client_id & Client_secret: you should copy these values from the generated keys on the website
- v: this parameter represents the date, i.e. YYYYMMDD. Use today's date.
- near: is the value selected from the previous activity

Example:
https://api.foursquare.com/v2/venues/search?client_id=0KMRPVTCV2THNZ5MLI5PLH023AC2AFTWDSGSSD5VWSVOKHBH&client_secret=QPMSBLPOEQISLMODLIJMHZ0IYCMH2BTDIDRM4BCHOAFBXYKG&v=20160320&near=Charlotte,NC

2. The response is in JSON format.
3. All the Venue items are under venues. Each array item includes the Venue id, Venue name, Categories and many other fields. In addition, each story item includes a thumbnail
4. Create a Venue class which contains the following attributes:
    - **VenueID – venues -> id**
    - **VenueName – venues -> name**
    - **CategoryName – venues-> categories -> name**
    - **CategoryIcon – venues -> catergories -> icon -> prefix+suffix**
      Example: Catergory icon should be concatenation of prefix and suffix. In the above screen shot the category icon will be
    - **Check in Count – venues ->stats -> checkinCount**
5. 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, display a ProgressDialog as indicated in Figure 2a. The JSON parser should return a list of Venue objects.

```
response: {
  - venues: [
      - {
            id: "4b05863df964a520c95922e3",
            name: "Charlotte",
          + contact: {…},
          + location: {…},
          - categories: [
              - {
                    id: "50aa9e094b90af0d42d5de0d",
                    name: "City",
                    pluralName: "Cities",
                    shortName: "City",
                  - icon: {
                        prefix: "https://ss3.4sqi.net/img/categories_v2/parks_outdoors/neighborhood_",
                        suffix: ".png"
                    },
                    primary: true
                }
            ],
            verified: false,
          - stats: {
                checkinsCount: 45363,
                usersCount: 15119,
                tipCount: 48
            },
            url: "http://www.charmeck.org",
          + specials: {…},
            storeId: "",
          + hereNow: {…},
            referralId: "v-1458352174",
            venueChains: [ ]
        },
      + {…},
```


Figure 2. Sample JSON result of venues search API

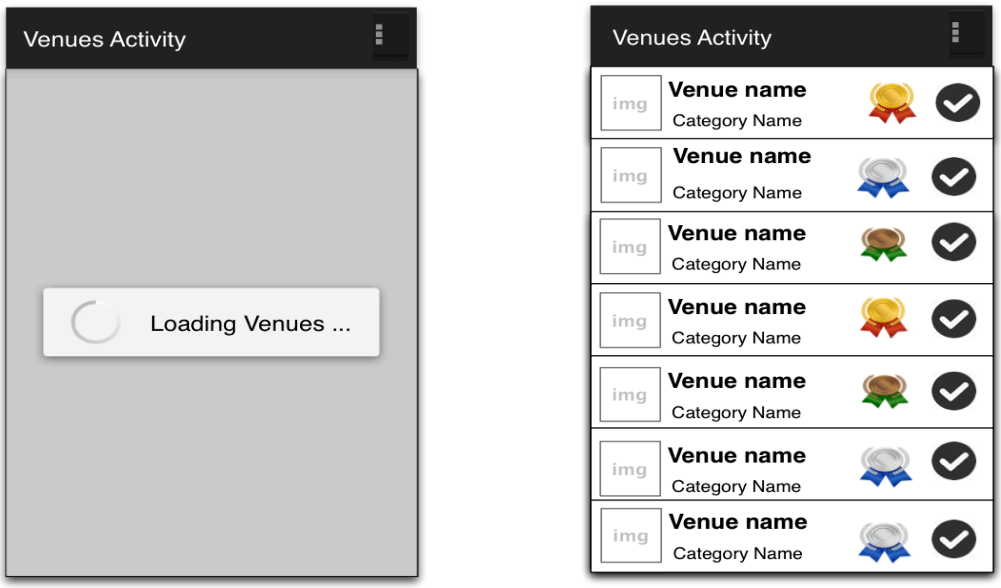

Figure 3. The Venues Activity Wireframe

6. Create a custom list view that shows list of Venue objects created and displays Venue name, Category name, Category Icon, Unmarked Icon as default , performance badge icon (red/silver/bronze) based on the check in count for every item as shown in the figure
7. The performance badge images are decided based on the check in count by following the rules below:
    - The **check in count** is between 0-100 then use bronze.png
    - The **check in count** is between 101-500 then use Silver.png
    - The **check in count** is > 500 then use gold.png
8. A LongClick on the list item should have the following behavior:
    a) If the VenueID of the item is not saved in the Shared Preferences, then save it in the Shared preference and update the UI by marking the item showing the Green Color tick mark icon (see Figure 4).
    b) If the VenueID is already in Shared Preferences, then delete the Venue item from the Shared Preferences and update UI by showing the black tick mark icon.
    c) Display a toast message indicating the status of the performed operation, successful addition/ deletion from the database or error.

<u>NOTE:</u> Please refer the code snippet below at the end of this document, for creating and accessing shared Preferences.

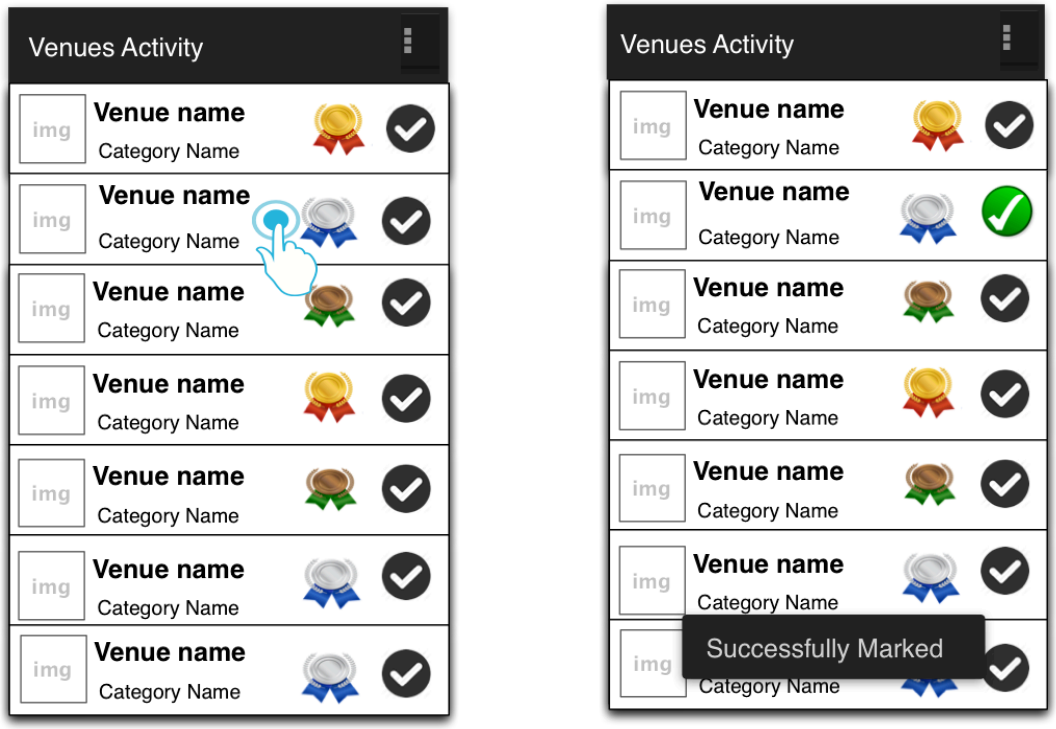

Figure 4 . Creating Marked Venues

9. The Action Bar menu should display two options, “Show Bookmarks” and “Clear All Bookmarks,” as shown in Figure 5.

a) “View Marked Venues”: Clicking this menu item, filters the list to show the Venues that have been previously marked i.e. saved in Shared Preferences (see Figure 5).
b) User should be able to clear a specific marked item from this list as well.
c) “Clear All”: Clicking this menu item should delete all the marked items in the Shared Preferences. Note that UI should also be updated, i.e. all the list items should show black bookmark icon, as shown in the Figure 6.

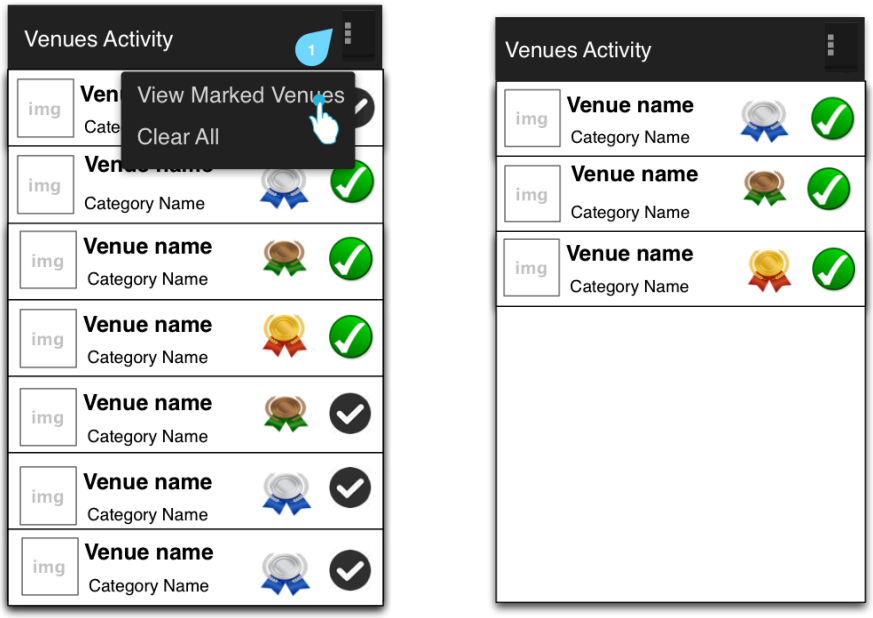

Figure 5 . The View Marked Items

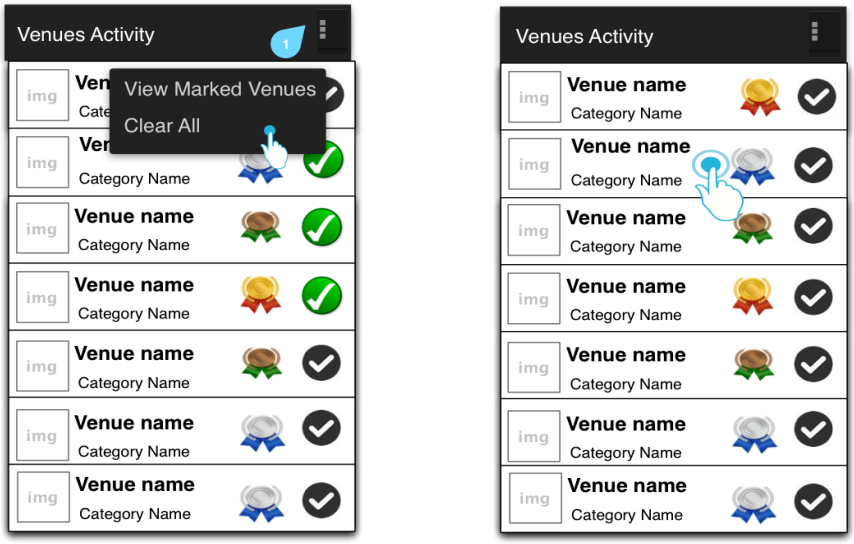

Figure 6. Clear All Marked Venues

Using Shared Preferences:

```
//Create Shared Preferences
SharedPreferences  pref = getSharedPreferences("pref", Context.MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();

editor.putBoolean("key_name1", true);           // Saving boolean - true/false
editor.putInt("key_name2", "int value");        // Saving integer
editor.putFloat("key_name3", "float value");    // Saving float
editor.putLong("key_name4", "long value");      // Saving long
editor.putString("key_name5", "string value");  // Saving string
editor.putStringSet("key_name6", "set values"); // Saving Set values

// Save the changes in SharedPreferences
editor.commit(); // commit changes

//Get SharedPreferences data:
pref.getBoolean("key_name1", true);         // getting boolean
pref.getInt("key_name2", 0);              // getting Integer
pref.getFloat("key_name3", null);           // getting Float
pref.getLong("key_name4", null);            // getting Long
pref.getString("key_name5", null);          // getting String
pref.getStringSet("key_name6",null);

//Deleting Key value from SharedPreferences:
editor.remove("key_name3"); // will delete key key_name3
editor.remove("key_name4"); // will delete key key_name4

// Save the changes in SharedPreferences
editor.commit(); // commit changes

//  Clear all data from SharedPreferences:
editor.clear();
editor.commit(); // commit changes
```