ITIS/ITCS 4180/5180 Mobile Application Development
Homework 5

Basic Instructions:

1. In every file submitted you MUST place the following comments:
    a. Assignment #.
    b. File Name.
    c. Full name of all students in your group.
2. Each group should submit only one assignment. Only the group leader is supposed to submit the assignment on behalf of all the other group members.
3. 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.
4. Please download the support files provided with this assignment and use them when implementing your project.
5. Export your Android project and create zip file which includes all the project folder and any required libraries.
6. Submission details: The file name should follow the following format: **Group#_HW05.zip**
7. **Failure to follow the above instructions will result in point deductions.**

Homework 5 (100 Points)

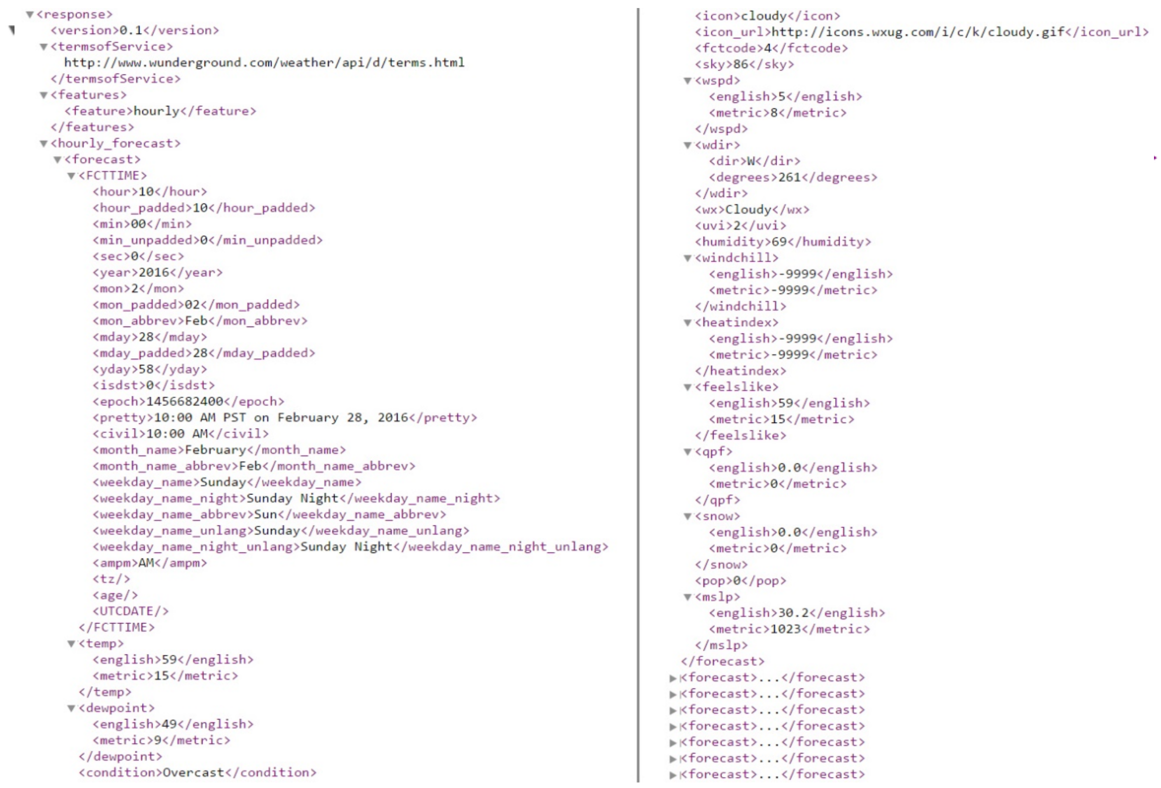
In this assignment you will get familiar with Android Concurrency, HTTP connections and XML parsing. You will build a simple weather application.

Initial Setup and API Description

You should use the Weather Underground API provided by (http://www.wunderground.com/) for getting the weather information. The API of interest is the Current Weather Data API which is based on the city name and state initials. You need to create an account in order to create an API key. Follow the below steps:

- Go to http://www.wunderground.com/weather/api/d/login.html and create a new account.
- After clicking on the activation link, login to your account and click on **"Explore My Options"**
- Keep the "STRATUS PLAN" selected and click on **"Purchase Key"**
    - Fill in the given form with required information
    - Click on **"Purchase Key"**
- Now that your key is generated, you can use that to make API calls.

We will be using wunderground hourly API for this application. For information related to the API please check http://www.wunderground.com/weather/api/d/docs?d=data/hourly.



```
▼<response>
   <version>0.1</version>
  ▼<termsofService>
     http://www.wunderground.com/weather/api/d/terms.html
   </termsofService>
  ▼<features>
     <feature>hourly</feature>
   </features>
  ▼<hourly_forecast>
    ▼<forecast>
      ▼<FCTTIME>
         <hour>10</hour>
         <hour_padded>10</hour_padded>
         <min>00</min>
         <min_unpadded>0</min_unpadded>
         <sec>0</sec>
         <year>2016</year>
         <mon>2</mon>
         <mon_padded>02</mon_padded>
         <mon_abbrev>Feb</mon_abbrev>
         <mday>28</mday>
         <mday_padded>28</mday_padded>
         <yday>58</yday>
         <isdst>0</isdst>
         <epoch>1456682400</epoch>
         <pretty>10:00 AM PST on February 28, 2016</pretty>
         <civil>10:00 AM</civil>
         <month_name>February</month_name>
         <month_name_abbrev>Feb</month_name_abbrev>
         <weekday_name>Sunday</weekday_name>
         <weekday_name_night>Sunday Night</weekday_name_night>
         <weekday_name_abbrev>Sun</weekday_name_abbrev>
         <weekday_name_unlang>Sunday</weekday_name_unlang>
         <weekday_name_night_unlang>Sunday Night</weekday_name_night_unlang>
         <ampm>AM</ampm>
         <tz/>
         <age/>
         <UTCDATE/>
       </FCTTIME>
      ▼<temp>
         <english>59</english>
         <metric>15</metric>
       </temp>
      ▼<dewpoint>
         <english>49</english>
         <metric>9</metric>
       </dewpoint>
       <condition>Overcast</condition>
       <icon>cloudy</icon>
       <icon_url>http://icons.wxug.com/i/c/k/cloudy.gif</icon_url>
       <fctcode>4</fctcode>
       <sky>86</sky>
      ▼<wspd>
         <english>5</english>
         <metric>8</metric>
       </wspd>
      ▼<wdir>
         <dir>W</dir>
         <degrees>261</degrees>
       </wdir>
       <wx>Cloudy</wx>
       <uvi>2</uvi>
       <humidity>69</humidity>
      ▼<windchill>
         <english>-9999</english>
         <metric>-9999</metric>
       </windchill>
      ▼<heatindex>
         <english>-9999</english>
         <metric>-9999</metric>
       </heatindex>
      ▼<feelslike>
         <english>59</english>
         <metric>15</metric>
       </feelslike>
      ▼<qpf>
         <english>0.0</english>
         <metric>0</metric>
       </qpf>
      ▼<snow>
         <english>0.0</english>
         <metric>0</metric>
       </snow>
       <pop>0</pop>
      ▼<mslp>
         <english>30.2</english>
         <metric>1023</metric>
       </mslp>
     </forecast>
    ▶<forecast>...</forecast>
    ▶<forecast>...</forecast>
    ▶<forecast>...</forecast>
    ▶<forecast>...</forecast>
    ▶<forecast>...</forecast>
    ▶<forecast>...</forecast>
    ▶<forecast>...</forecast>
```

Figure 1, XML Response

The API details is as follows:

- Endpoint: http://api.wunderground.com/api/<<api_key>>/hourly/q/<<state>>/<<city_name>>.xml
- Arguments:
  - `api_key`: this is the key that was generated eearlier.
  - `state`: this should be a 2 letter initial representing a state. Example: `CA`.
  - `city_name`: this should be the city name. Space is replaced by "_". Example: `San_Francisco`

For example to retrieve the weather for San Francisco, the url should be setup as follows (replace api_key with your key):

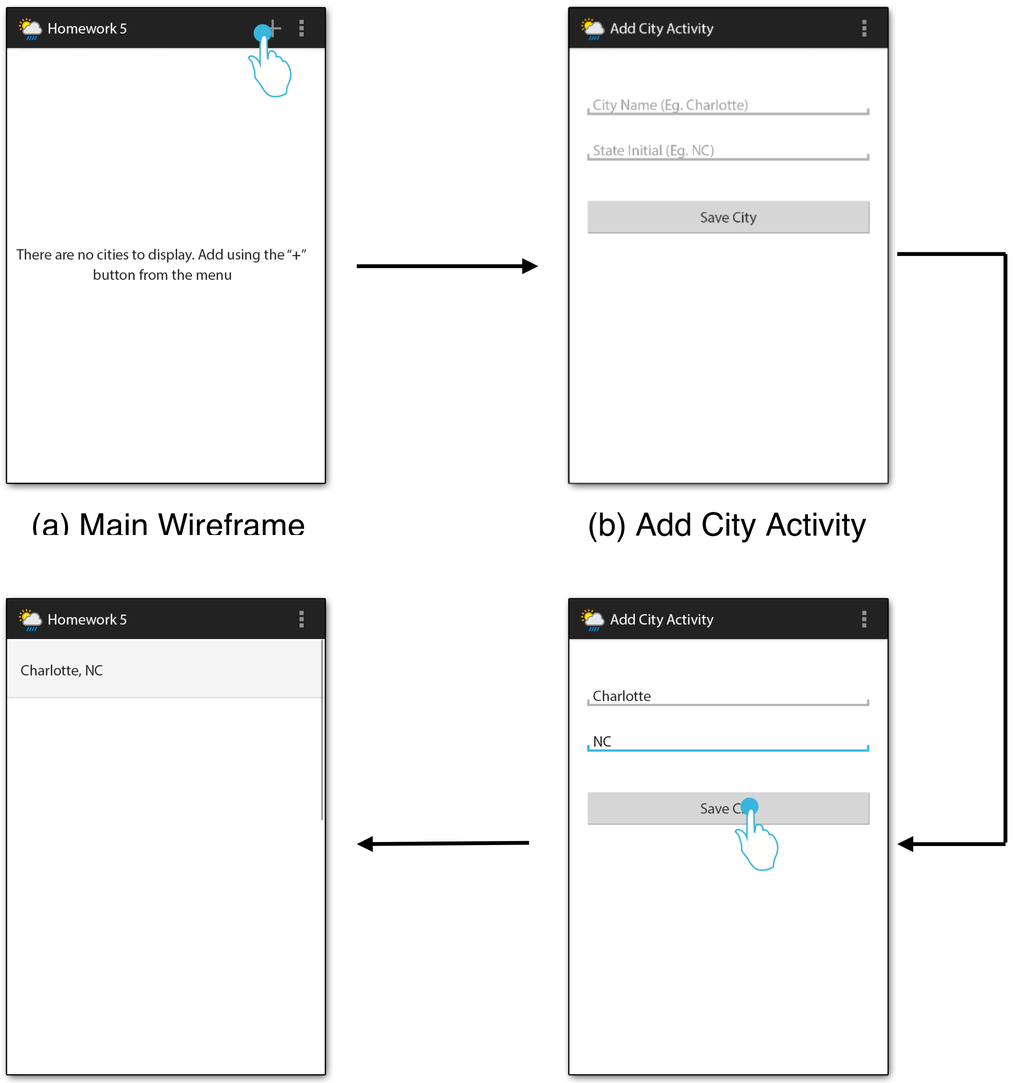
http://api.wunderground.com/api_key/hourly/q/CA/San_Francisco.xml

The response will be as shown in Figure 1.

Part 1: Main Activity (5 Points)

The Activity UI should match the UI presented in Figure 2(a) and Figure 2(d). Below are the requirements:

1. The activity should display a Default TextView showing **'There are no cities to display. Add using the "+" button from the menu'** for the first time as shown in Figure 2(a).
2. The activity should maintain an ArrayList of cities which are added from the Add City Activity.
3. There should be a **"+"** button on the Action Bar which will take you to the **Add City Activity.**
4. The Main Activity should get the Extra Value sent by the Add City Activity and update its List of cities.
5. Once the List has one or more cities, the Default TextView should be removed and a ListView should be created displaying the city names as shown in Figure 2(d).
6. Setup OnClickListener for the ListView. So, clicking a City Name should start the **HourlyData Activity** and send the cityname and state name through Extras.
7. **"Delete City"** functionality should be provided. Long Pressing the City Name will remove it from the ArrayList and refresh the ListView. See Figure 3



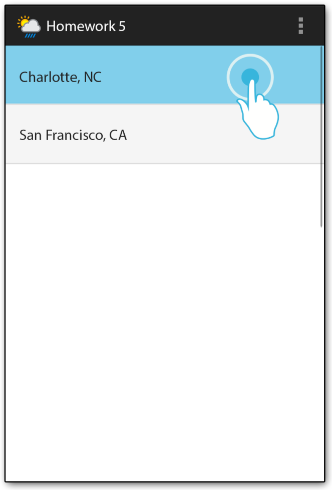
(a) Main Wireframe

(b) Add City Activity

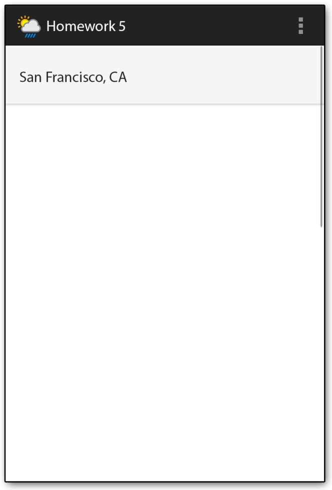
(d) City added to Main Activity

(c) Saving City and going to Main Activity

Figure 2, Main Activity and AddCity Activity



(a) Long Pressing a city name



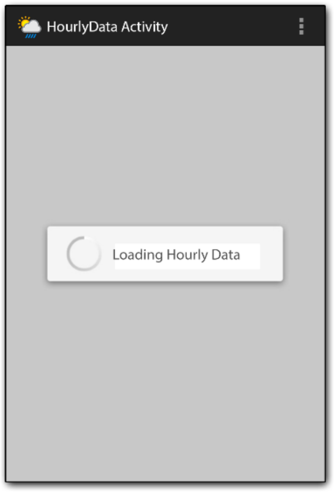
(b) City Deleted

Figure 3, Delete Functionality

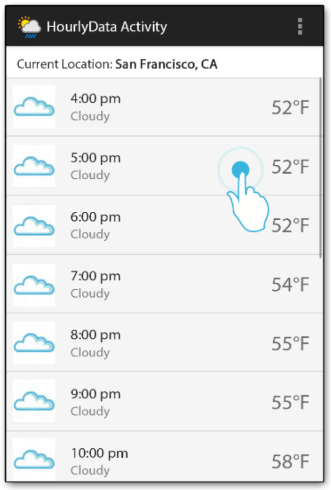
<u>Part 2: Add City Activity (10 Points)</u>

Figure 2(b) and Figure 2(c) shows the Add City Activity. This activity will add cities to an ArrayList. This list of cities from ArrayList will be displayed on the Main Activity.

1. The First EditText will hold city name and the second EditText will hold the 2 letter initial of the state. The values will be stored in an ArrayList.
2. User input should be properly validated. i.e. The State and the city should actually represent a valid State, City in US.
3. While storing the cityname, if there is a space in between the name then it should be replaced by an underscore sign.
   Example: "San Francisco" ➔ "San_Francisco"
4. Clicking the **"Save City"** button will finish this activity and go back to the main activity, which should display updated list of cities added.



(b) Parsing XML and showing ProgressDialog



Current Location: San Francisco, CA

4:00 pm	Cloudy	52°F
5:00 pm	Cloudy	52°F
6:00 pm	Cloudy	52°F
7:00 pm	Cloudy	54°F
8:00 pm	Cloudy	55°F
9:00 pm	Cloudy	55°F
10:00 pm	Cloudy	58°F

(c) HourlyData Activity

Figure 4, HourlyData Activity

Part 3: HourlyData Activity using XML Pull/SAX Parser (60 Points)

The HourlyData activity is responsible for retrieving the Hourly Data by calling the wunderground api with the results based on the “cityname” and “state name” sent from the Main Activity. The Requirements is as follows:

1. Form the URL to call the API from the Extra values.
   Example:

   Values sent from Main Activity:

   ```
   city_name – San_Francisco
   state_initial - CA
   ```

   URL to call:
   http://api.wunderground.com/api_key/hourly/q/**state_initial**/**city_name**.xml

   The URL will be formed as below using the Extra values:
   http://api.wunderground.com/api_key/hourly/q/**CA**/**San_Francisco**.xml

2. If “cityname”, “statename” have wrong values then API will return “No cities match your query”. In that case display a Toast message to the user indicating the error and finish the current and navigate back the previous activity.
3. After forming the URL, send a GET request and retrieve the results for the selected section. 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 XML is being downloaded and parsed you should display a progress dialog as indicated in Figure 4(a).
4. Figure 1, shows a sample xml feed. Each hour forecast details are under the <forecast> tag. The required information is as follows:
   a. All the **timing** details are under FCTTIME.
   b. Current Hour **temperature** is under <temp> tag. Use the value from <english> tag
   c. **Dewpoint** value is under <dewpoint>. Use the value from <english> tag
   d. Use the value from <condition> for **Clouds**.
   e. **Icon URL** is present under <icon_url>
   f. **Wind Speed** value is under <wspd> (Use the value from <english> tag) and **Wind Direction** is under <wdir>.
   g. **Climate Type** is stored under <wx> tag.
   h. **Humidity** is under <humidity> tag.
   i. **Feels Like** is present under <feelslike>
   j. **Pressure** is under <mslp> tag. Use the value from <metric> tag
   k. In addition, each story item includes a thumbnail under multimedia and format: <“Standard Thumbnail”> and a normal image under <format: “Normal”>.
5. Create a weather class containing the following string variables:
   time, temperature, dewpoint, clouds, iconUrl, windSpeed, windDirection, climateType, humidity, feelsLike, maximumTemp, minimumTemp and pressure.
6. You should use a separate thread to perform data retrieval from the server and data parsing. Do not use the Main Thread. Use an AsyncTask or a Thread/Handler.
7. Use Pull/SAX Parser for parsing the XML. Implement an XML Parser and pass the

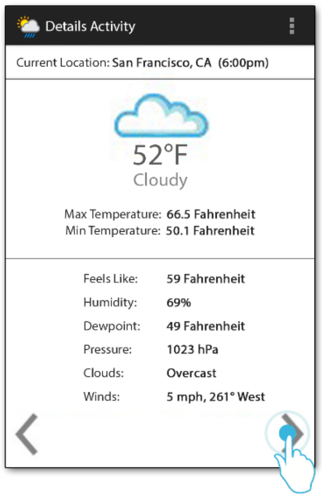
document stream to the parser. Parse the weather forecast information and store each hour forecast in a weather object.

8. The progress dialog should be dismissed after the parsing is completed, and the hourly data items should be displayed in a list. This list should be created using a customized ListView. Each item in the ListView should contain a TextView showing the *time*, ImageView displaying the corresponding *weather icon(from <icon_url>)*, using the iconURL string that is retrieved from the XML atttribute “icon_url.” See Figure 4(b).
9. You must use Picasso Library to retrieve and display the thumbnails. Check http://square.github.io/picasso/
10. Calculate the maximum and minimum temperature over all forecast data, and save them in maximumTemp and minimumTemp respectively for each weather object.
11. Clicking on any item in listview should start the Details Activity and send the ArrayList of weather objects as Extras.

Part 4: Details Activity (25 Points)

The activity layout should match the one in Figure 5. It will display the details of the hour that has been clicked. You can also cycle through different hours using Right and Left Arrow Keys. The Requirements are as follows:

1. The activity should display the weather image from the iconUrl as shown in Figure 5(a).
2. Use TextViews to display the temperature, climateType, maximumTemp, minimumTemp, feelsLike, humidity, pressure, clouds, windSpeed, and windDirection.
3. Note that the units are supposed to be appended for temperature (Fahrenheit), humidity (%), dewpoint (Fahrenheit), feelsLike (Fahrenheit), pressure (hPa).
4. The wind speed and wind direction should be appended and shown with “Winds” TextView.



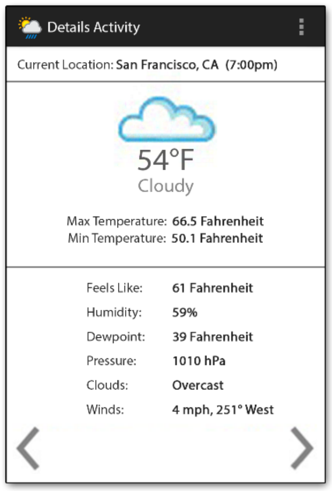
Current Location: San Francisco, CA (6:00pm)

52°F
Cloudy

Max Temperature: 66.5 Fahrenheit
Min Temperature: 50.1 Fahrenheit

Feels Like: 59 Fahrenheit
Humidity: 69%
Dewpoint: 49 Fahrenheit
Pressure: 1023 hPa
Clouds: Overcast
Winds: 5 mph, 261° West

(a) Details Activity and click on next



Current Location: San Francisco, CA (7:00pm)

54°F
Cloudy

Max Temperature: 66.5 Fahrenheit
Min Temperature: 50.1 Fahrenheit

Feels Like: 61 Fahrenheit
Humidity: 59%
Dewpoint: 39 Fahrenheit
Pressure: 1010 hPa
Clouds: Overcast
Winds: 4 mph, 251° West

(b) HourlyData for next hour

Figure 5, Details Activity

5. The wind direction should have West for “W”, North for “N”, East for “E” and South for “S”.
6. Pressing back should take you to the HourlyData Activity.
7. Clicking on the arrows should change the forecast displayed to the next time block. The retrieved location, max temperature, and min temperature should not change. See Figure 5(b).

<u>Bonus (0.5% of the overall course grade)</u>

Annotation Based Parsing simplifies parsing and conversion into Java objects. Instead of the SAX or XMLPull parsers use the simple XML serialization to parse the XML for this assignment. In addition use the disk cache to store the retrieved images to enhance the application performance and to avoid re-downloading perviously downloaded images.

1. For information about Simple XML Serialization:
   1.1. Main Information: http://simple.sourceforge.net/home.php
   1.2. Tutorial: http://simple.sourceforge.net/download/stream/doc/tutorial/tutorial.php
2. For image caching you should use the Picasso library:
   2.1. Main Information: http://square.github.io/picasso/
   2.2. If the image is present in the cache it should be retrieved from the cache and displayed. If the image is not in the cache then it should be downloaded, displayed and loaded in the cache.
3. In the submission comments section, you should specify which bonus parts have been implemented if any in order for the grader to award you points accordingly.