

ITIS/ITCS 4180/5180 Mobile Application Development
Homework 6

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#_HW06.zip
7. **Failure to follow the above instructions will result in point deductions.**

Homework 6 (100 Points)

This assignment is a continuation of last homework assignment. In this assignment you will get familiar with SQLite Operations. You will build a simple weather application that is not only for checking the weather, but also enables user planning activities associated with a specific day such as hiking, shopping, etc. The plans will be saved as short notes associated with a specific day in a specific city.

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 and forecast10day API for this application.

For information related to the API please check

<http://www.wunderground.com/weather/api/d/docs?d=data/hourly>. and

<http://www.wunderground.com/weather/api/d/docs?d=data/forecast10day>

Figure 1(a), Hourly XML Response

```
</qpf_day>
▼<qpf_night>
  <in>0.02</in>
  <mm>1</mm>
</qpf_night>
▼<snow_allday>
  <in>0.0</in>
  <cm>0.0</cm>
</snow_allday>
▼<snow_day>
  <in>0.0</in>
  <cm>0.0</cm>
</snow_day>
▼<snow_night>
  <in>0.0</in>
  <cm>0.0</cm>
</snow_night>
▼<maxwind>
  <mph>35</mph>
  <kph>56</kph>
  <dir>SSW</dir>
  <degrees>202</degrees>
</maxwind>
▼<avewind>
  <mph>25</mph>
  <kph>40</kph>
  <dir>SSW</dir>
  <degrees>202</degrees>
</avewind>
<avehumidity>76</avehumidity>
<maxhumidity>0</maxhumidity>
<minhumidity>0</minhumidity>
</forecastday>
►<forecastday>...</forecastday>
►<forecastday>...</forecastday>
►<forecastday>...</forecastday>
►<forecastday>...</forecastday>
►<forecastday>...</forecastday>
►<forecastday>...</forecastday>
►<forecastday>...</forecastday>
►<forecastday>...</forecastday>
</forecastdays>
</simpleforecast>
</forecast>
```

Figure 1(b), 10day Forecast XML Response

The API details is as follows:

- Endpoint:
[http://api.wunderground.com/api/<<api_key>>/\(hourly/forecast10day\)/q/<<state>>/<<city_name>>.xml](http://api.wunderground.com/api/<<api_key>>/(hourly/forecast10day)/q/<<state>>/<<city_name>>.xml)
- Arguments:
 - `api_key`: this is the key that was generated earlier.
 - `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

http://api.wunderground.com/api/api_key/forecast10day/g/CA/San_Francisco.xml

The response will be as shown in Figure 1(a) and Figure 1(b).

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. Create the SQLite database with tables:
 - a. **Cities** with columns citykey, cityname, state.
 - b. **Notes** with columns citykey, date, note
2. The activity should initially display the list of cities that have already been saved in the Cities table along with the current temperature as shown in the figure 3a
3. If the Cities table has no records then the activity should display a Default TextView showing **'There are no cities to display. Add using the Add City from the menu'** for the first time as shown in Figure 2(a).
4. The activity should maintain an ArrayList of cities which are retrieved from Cities table
5. Menu Items:
 - a. **Add city:** navigates to the **Add City Activity** (see Figure2(b)). This activity enables the user to enter a city and state values and save them in the database. Note: the Main Activity should be updated to reflect what has been in the database (see Figure2(d)).
 - b. **Clear Saved Cities:** deletes all the entries in both the tables in the database. The activity should show the TextView showing **'There are no cities to display. Add using the Add City from the menu'** in the place of ListView (see Figure 4). Display a Toast message indicate the result of the operation (success/ failure).
 - c. **View Notes:** navigates to the NotesActivity only if there are any saved cities otherwise it should prompt user to add city (see Figure 5).
6. Once the List has one or more cities, the Default TextView should be removed and a ListView should be created with custom adapter displaying the city names and current temperature of that city as shown in Figure 2(d). Current temperature should be retrieved through API call in the back ground.
7. Setup OnClickListener for the ListView: clicking a City Name should start the **CityData Activity** and send the cityname and state name through Extras.
8. **"Delete City"** functionality should be provided. Long Pressing the City Name will remove it from the ArrayList and the database and then refresh the ListView. See Figure 3

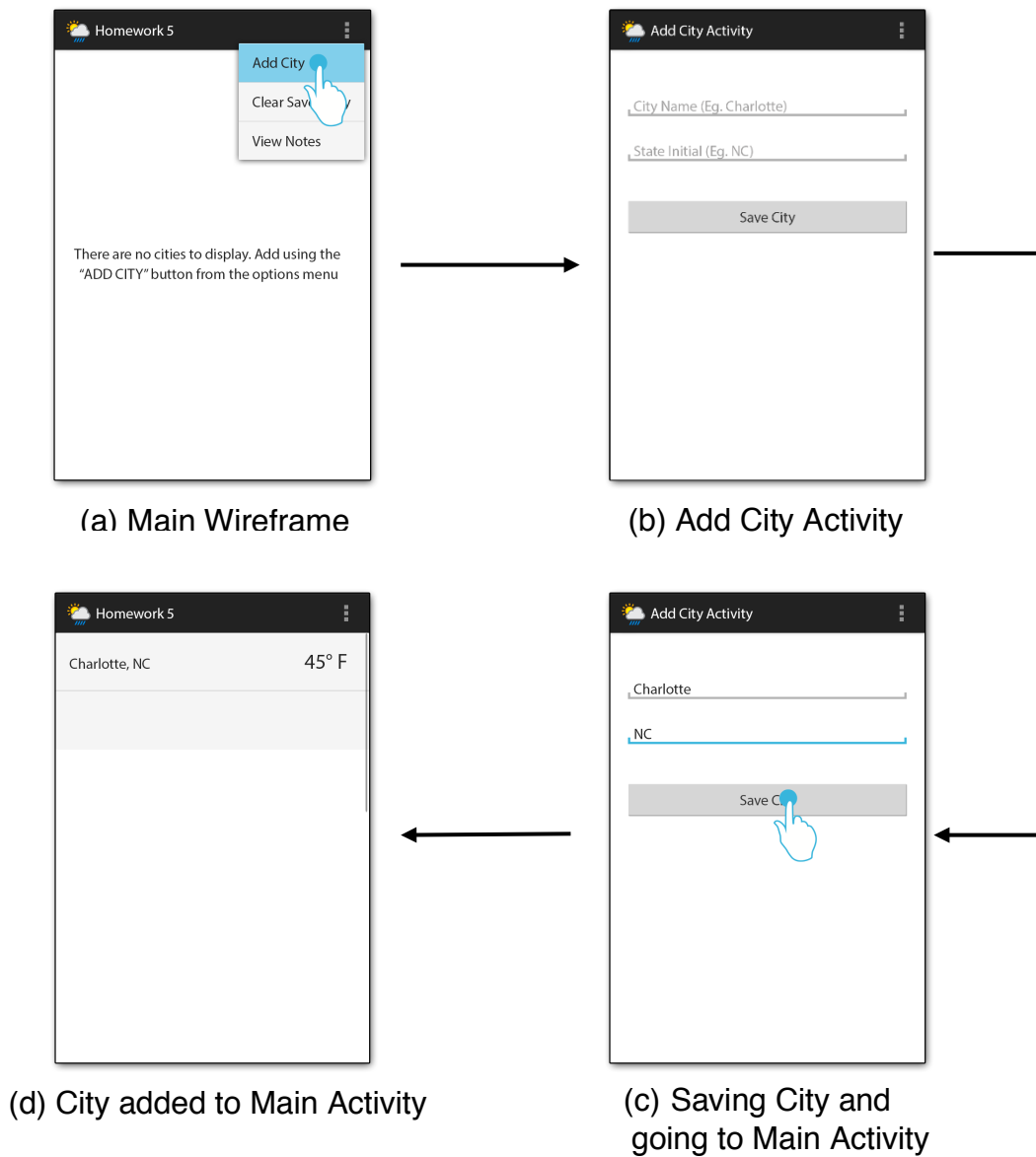
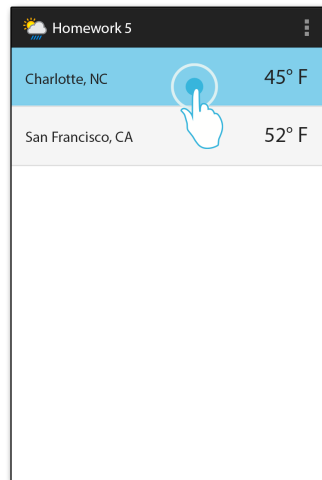
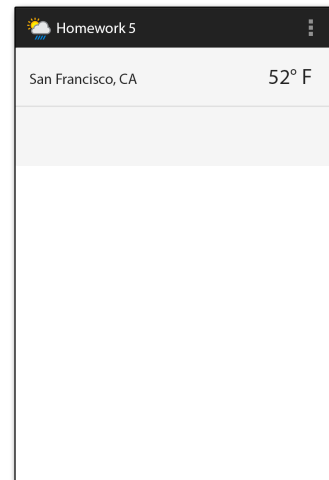


Figure 2, Main Activity and AddCity Activity

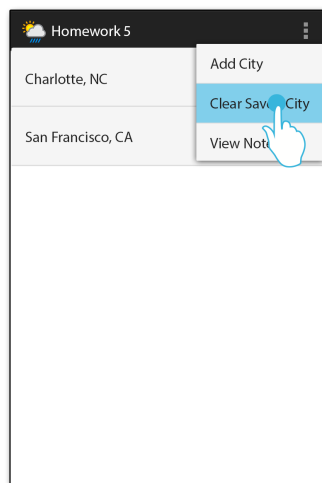


(a) Long Pressing a city name

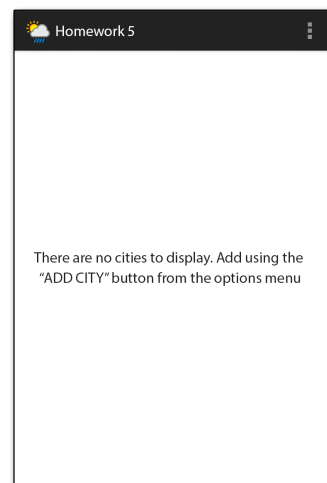


(b) City Deleted

Figure 3, Delete Functionality

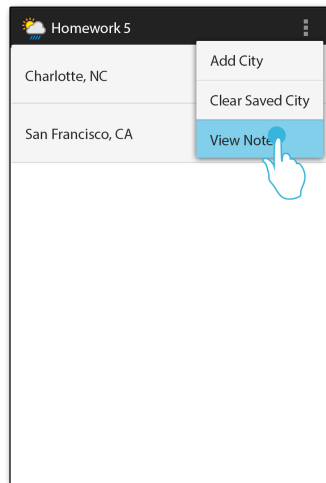


(a) Clear Saved Cities

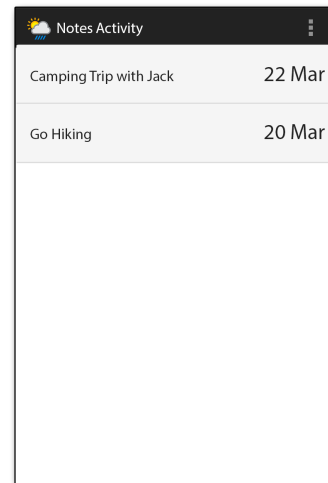


(b) Cities deleted from database

Figure 4, Clear Cities Functionality



(a) View Notes



(b) Notes Activity with list of Notes from database

Figure 5, View Notes Functionality

Part 2: Add City Activity (10 Points)

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”

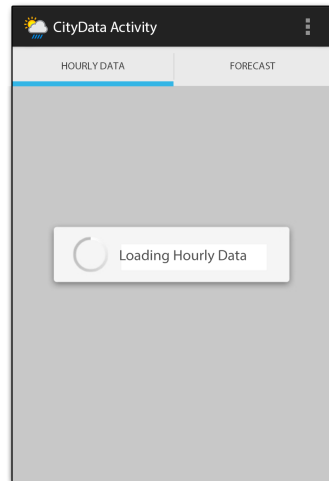
4. Clicking the “**Save City**” button will finish this activity and go back to the main activity, which should display updated list of cities added.

Part 3: CityData Activity (20 Points):

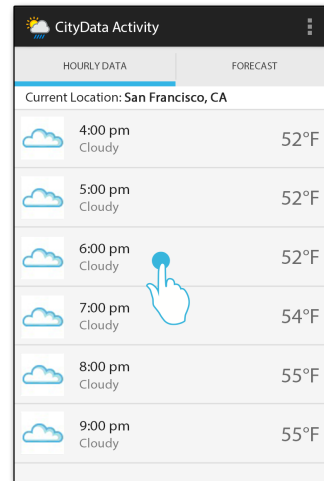
This Activity extends TabActivity and the layout should create a tab host as shown in the Figure 5. This activity should pass to the extras received from the MainActivity to each intent of the tab host. Please refer this online tutorial for creating the tab host

"<http://www.androidhive.info/2011/08/android-tab-layout-tutorial/>"

1. The first tab should contain **HourlyData Activity** and it is the default tab for the tab host.
2. The Second tab should contain **Forecast Activity**.



(a) Parsing XML and showing ProgressDialog



(b) HourlyData Activity

Figure 5, HourlyData Activity

Part 4: HourlyData Activity using XML Pull/SAX Parser (20 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 CityData 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 5(a).

4. Figure 1(a), 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 attribute “icon_url.” See Figure 5(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 5: Forecast Activity using XML Pull/SAX Parser (20 Points)

The Forecast activity is responsible for retrieving the 10 days Forecast Data by calling the wunderground api with the results based on the “cityname” and “state name” sent from the CityData Activity. The Requirements is as follows:

1. Form the URL to call the API from the Extras values.

Example:

Values sent from Main Activity:

city_name – San_Francisco
state_initial – CA

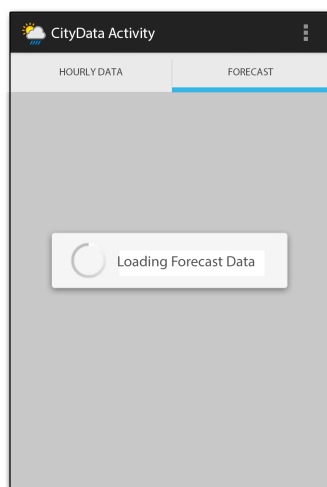
URL to call:

http://api.wunderground.com/api/api_key/forecast10day/q/state_initial/city_name.xml

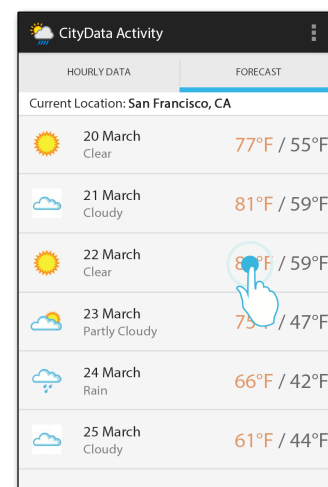
The URL will be formed as below using the Extras values:

http://api.wunderground.com/api/api_key/forecast10day/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



(a) Parsing XML and showing ProgressDialog



(b) Forecast Activity

Figure 6, Forecast Activity

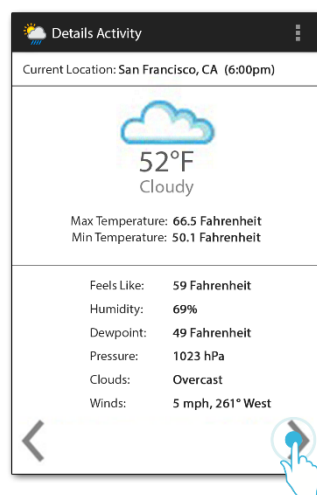
4. Figure 1(b), shows a sample xml feed. Each hour forecast details are under the <forecastdays> tag. The required information is as follows:
 - a. **Date** details are under <pretty.>
 - b. **HighTemp** is under <Fahrenheit> tag. Use the value from < high> tag
 - c. **lowTemp** value is under <Fahrenheit>. Use the value from <low> tag
 - d. Use the value from <condition> for **Clouds**.
 - e. **Icon URL** is present under <icon_url>
 - f. **maxwindSpeed** value is under < mph > (Use the value from <maxwind > tag) and **Wind Direction** is under <dir >.

- g. **avghumidity** is stored under <avehumidity > tag.
5. Create a Forecast class containing the following string variables:
Date, Hightemp, lowTemp, clouds, iconUrl, maxwindSpeed, windDirection, avghumidity
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 forecast day information and store each forecast day in a Forecast 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 *Date and month*, ImageView displaying the corresponding *forecast icon*(from <icon_url>), using the iconURL string that is retrieved from the XML attribute "icon_url." See Figure 6(b).
9. You must use Picasso Library to retrieve and display the thumbnails.
Check <http://square.github.io/picasso/>
10. OnLongClick on the forecast item it should navigate to Add Notes Activity sending date and city as extras.

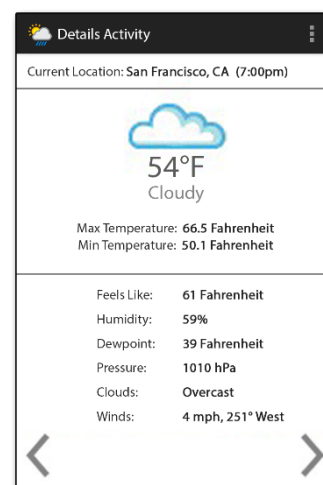
Part 6: Details Activity (15 Points)

The activity layout should match the one in Figure 7. 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 7(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.



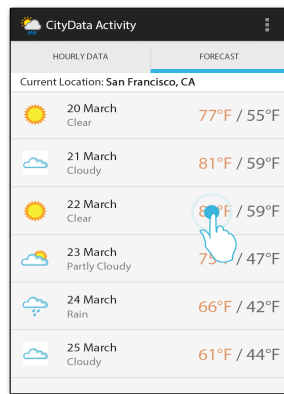
(a) Details Activity and click 12 of 14
on next



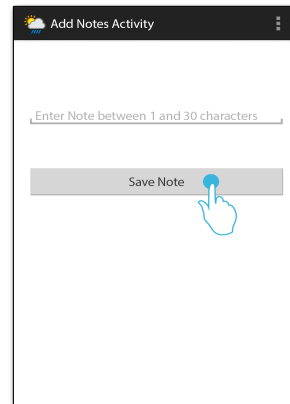
(b) HourlyData for next hour

Figure 7, Details Activity

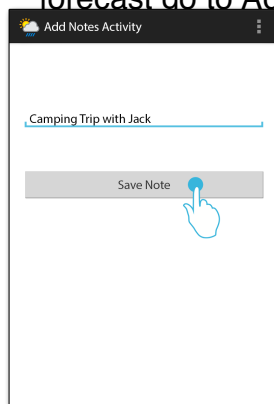
5. The wind direction should have West for “W”, North for “N”, East for “E” and South for “S”.
6. Pressing back should take you to the HourlyData Activity.
7. Clicking on the arrows should change the forecast displayed to the next time block. The retrieved location, max temperature, and min temperature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



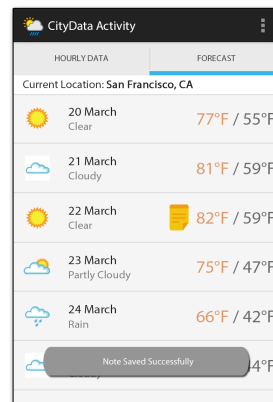
(a) Onclicking on a forecast go to Add



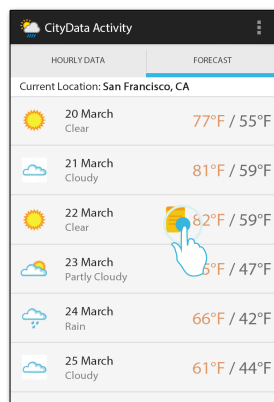
(b) Add Notes Activity



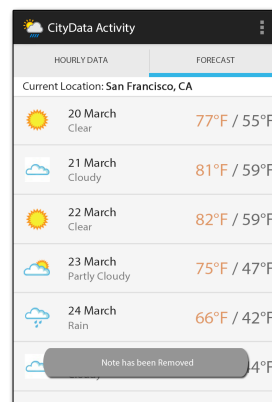
c) Save note stores the note in database and navigates to fore cast activity



d) Show Notes Icon and show toast message



e) On Long Clicking the forecast item Corresponding note should be removed



(f) Toast upon removing the

Figure 8, Add Notes Activity

Part 7: AddNotesActivity (5 Points)

This Activity is used to add the Notes to the database table. This activity receives the extras as date and the city name as shown in the figure 8.

1. The Activity should have an EditText to input the Notes and a submit button as shown in figure 8(b)
2. Upon clicking on the submit button the Notes should be saved with date and city key(retrieved using city name from City Table) into the Notes table in the database.
3. User should be prompted with the message that note has been saved and Activity should finish and show Forecast activity as shown in figure 8(d).

Part 8: NoteActivity (5 Points)

This Activity shows all the Notes saved in the Notes table in the ListView as shown in the figure. Use the custom adapter to include date for the Notes.

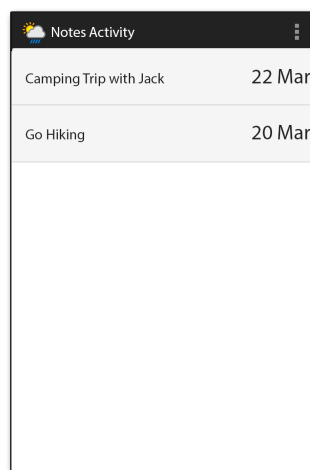


Figure 9, Notes Activity