Lab 6: Mobile GIS Application: Part (2)

In previous lab, we made a simple Android application that can show all reports in the database on Google Map as markers. In this lab, we will extend the application to add two more features: 1) query reports; 2) submit a new report.

Objectives

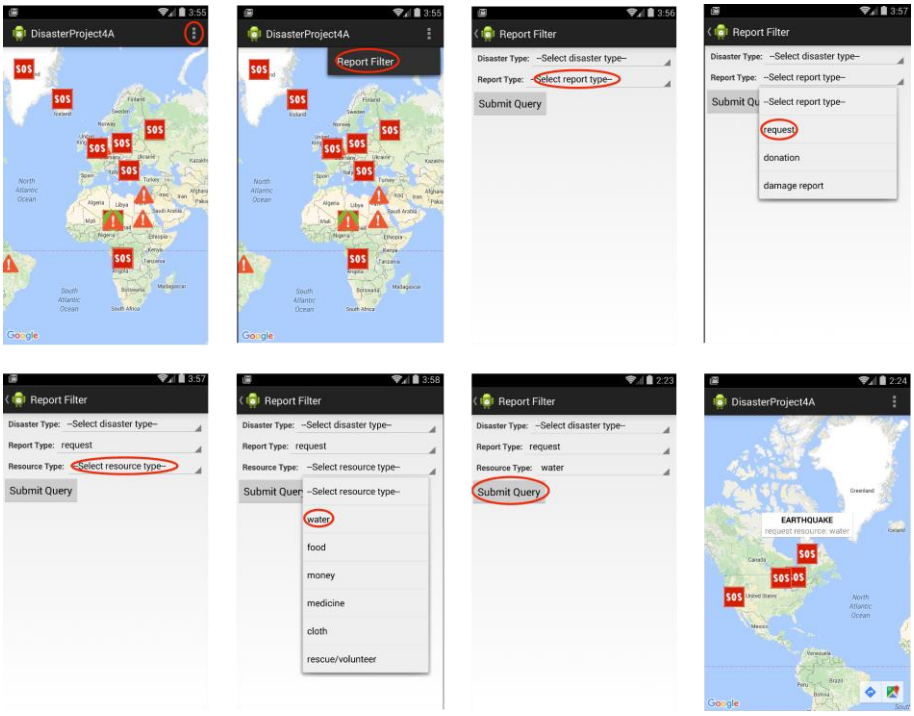
Through this lab, you will learn:

- Activity and Intent
- Android Application Location-Based Service (LBS)

Part 1: Lab Instructions

1. Query Report

In this section, you will add a feature to the application to allow users to query reports to show only a portion of the reports on the Google Map. Similar to the web implementation we already did in Lab 4, we will create a form where the users can submit query. Below is an example of how to query all water request reports.



As you can see, the user clicks "Report Filter" on the option menu and enters another activity. After the user clicks "Submit Query", he will go back to the MainActivity with only the reports satisfying the query showing on the Google Map.

Before you continue, please download the ***DisasterMngt4A.zip*** file and import the project to your workspace. This contains the answers to the Lab 5 questions. We will work based on this imported project in the following sections.

1.1 Create Layout

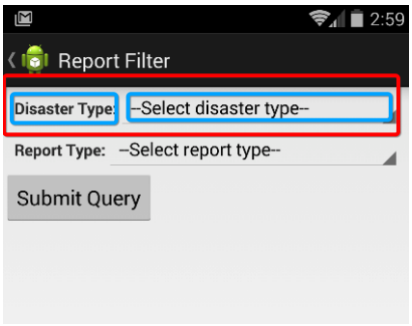
Our first step is to create the layout representing the activity. Create a file named ***activity_report_filter.xml*** under ***res/layout*** folder. Add a simple LinearLayout in the file as follows:

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical" >
6
7  </LinearLayout>
8
```

Refer the code in activity_report_filter_1.xml

A LinearLayout is a Layout that arranges its children in a single column or a single row (http://developer.android.com/reference/android/widget/LinearLayout.html). As specified in Line 5, it uses "vertical" as its orientation, meaning the childeren are in a single column.

Now, we will add the first child which itself is a LinearLayout containing two childeren aligned horizontally highlighted in the figure below.



Add the folloiwng code to your activity_report_filter.xml.

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical" >
6
7      <LinearLayout
8          android:paddingLeft="10dp"
9          android:paddingRight="10dp"
10         android:layout_width="match_parent"
11         android:layout_height="wrap_content"
12         android:orientation="horizontal">
13         <TextView
14             android:layout_width="wrap_content"
15             android:layout_height="wrap_content"
16             android:textStyle="bold"
17             android:text="Disaster Type: " />
18         <Spinner
19             android:id="@+id/disaster_type_spinner"
20             android:layout_width="fill_parent"
21             android:layout_height="wrap_content" />
22     </LinearLayout>
23
24 </LinearLayout>
25
```

Refer the code in activity_report_filter_2.xml

Code explanation:

Line 8-9: Add paddings on both left an right side.
Line 12: Specify that the children inside the Layout align horizontally (see the figure above).
Line 13-17: Create a text label with text "Disaster Type: ".
Line 16: Specify the text style to be bold.
Line 18-22: Create a Spinner. It is a dropdown list just like the <select> web element we learned in Lab 4. For more information about Spinner, please check

http://developer.android.com/guide/topics/ui/controls/spinner.html.

Line 19: Speficy an id for the Spinner so that we can refer to this element in Java code.

In Eclipse, On Line 17, you should notice a warning saying "Hardcoded string; should use @string resource". To fix this, first edit string.xml under res/values directory to create a string resource as follows:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">Disaster Management</string>
5      <string name="action_settings">Settings</string>
6      <string name="disaster_query_label">Disaster Type: </string>
7
8  </resources>
9
```

Refer the code in string_1.xml

Next, replace Line 17 in ***activity_report_filter.java*** with:

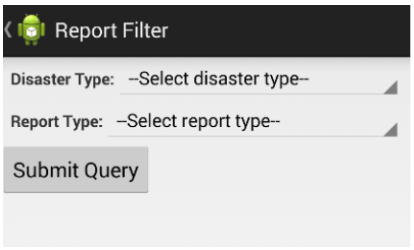
```
17              android:text="@string/disaster_query_label" />
```

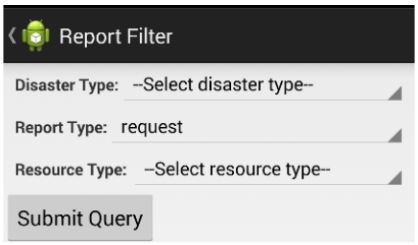
Similarily, create another LinearLayout for report type by editing ***activity_report_filter.java***. Add another string resource in ***string.xml***.

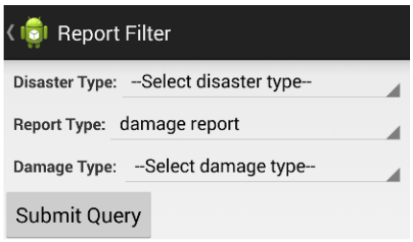
[Show the code here]

Refer the code in activity_report_filter_3.xml and string_2.xml

As you should notice, there is a hidden field in the ReportFilter activity. After selecting a report type, the filed will show as either resource type or damage type according to the different report types selected (See the figure below). Recall the Query Report tab form in Lab 4, we have already implemented a web version of this feature.







Thus, we edit ***activity_report_filter.java*** as follows:

```
41      <LinearLayout
42          android:id="@+id/resource_or_damage_type_layout"
43          android:paddingLeft="10dp"
44          android:paddingRight="10dp"
45          android:layout_width="match_parent"
46          android:layout_height="wrap_content"
47          android:orientation="horizontal"
48          android:visibility="gone">
49          <TextView
50              android:id="@+id/resource_or_damage_type_text_view"
51              android:layout_width="wrap_content"
52              android:layout_height="wrap_content"
53              android:textStyle="bold"/>
54          <Spinner
55              android:id="@+id/resource_or_damage_type_spinner"
56              android:layout_width="fill_parent"
57              android:layout_height="wrap_content" />
58      </LinearLayout>
```

Refer the code in activity_report_filter_4.xml

Code explanation:

Line 48: Set the visibility of the Layout to be hidden ("gone"). You can also use "invisible". For more information about this attribute, please refer at http://developer.android.com/reference/android/view/View.html#attr_android:visibility.

Line 50: Since the label text will change according to the report type selected, we specify its element id as a reference.

The last step is to add a button for submitting the query as follows:

```
59
60      <Button
61          android:layout_width="wrap_content"
62          android:layout_height="wrap_content"
63          android:text="@string/submit_button"
64          android:onClick="submitQuery" />
65
66  </LinearLayout>
```

Refer the code in activity_report_filter_5.xml

Code explanation:

Line 64: Register an onClick event for the button. "submitQuery" is the name of the method that will be exxecuted on button clicked. You will see where and how to define this method later.

Also add the string resource for button text:

Refer the code in string_3.xml

1.2 Create Activity

As we have already created the layout file, we need to create the activity for it. Create a ***ReportFilterActivity*** class in your ***org.disasterproject4a*** package. There are several purposes of the activity. First, we will initialize the items in the spinners. Next, we need to dynamically show the hidden Layout on report type selected. Lastly, we need to create the ***submitQuery*** method for report submission.

To initialize the spinners, first we need to add some string resources for each spinner. Each spinner corresponds to an array of strings. For example, the report type spinner should have three types of report: request, donation and damage report. In this case, it's better to use string array resources. You can read the documentation on http://developer.android.com/guide/topics/resources/string-resource.html#StringArray for more information about string array.

Add the following four string arrays to ***string.xml***:

```
<string name="disaster_query_label">Disaster Type: </string>
<string name="report_type_query_label">Report Type: </string>
<string name="submit_button">Submit Query</string>

<string-array name="disaster_type_spinner">
    <item>--Select disaster type--</item>
    <item>flood</item>
    <item>wildfire</item>
    <item>earthquake</item>
    <item>tornado</item>
    <item>hurricane</item>
    <item>other</item>
</string-array>
<string-array name="report_type_spinner">
    <item>--Select report type--</item>
    <item>request</item>
    <item>donation</item>
    <item>damage report</item>
</string-array>
<string-array name="resource_type_spinner">
    <item>--Select resource type--</item>
    <item>water</item>
    <item>food</item>
    <item>money</item>
    <item>medicine</item>
    <item>cloth</item>
    <item>rescue/volunteer</item>
</string-array>
<string-array name="damage_type_spinner">
    <item>--Select damage type--</item>
    <item>polution</item>
    <item>building damage</item>
    <item>road damage</item>
    <item>casualty</item>
    <item>other</item>
</string-array>

</resources>
```

Refer the code in string_4.xml

Both the string arrays "resource_type_spinner" and "damage_type_spinner" are for Spinner with id "resource_or_damage_type_spinner". When the user selects request or donation in report type Spinner, it uses the string array "resource_type_spinner" to fill the Spinner list; when the user selects damage report, it uses "damage_type_spinner" instead.

Now add the following code to ***ReportFilterActivity.java***:

Refer the code in ReportFilterActivity_1.java

Code explanation:

Line 13-17: Define five private members of the activity representing the elements in the layout. Three of them correspond to the Spinners. One represens the hidden LinearLayout. Another one is for the TextView in the hideen LinearLayout. The reason we keep these five private members is that we want to manipulate these elements in our code (i.e. fill contens in the spinners, set the Layout visibility, set the text for the TextView).

Line 22: Set the content view using the layout resource.

Line 24: Find the specific view using the id defined in the layout file.

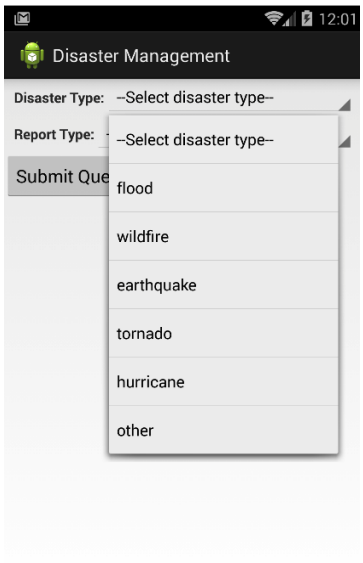
Line 25: Use the string array resource to initialize the spinner list.

Line 36: Create a helper function for setting spinner dropdown list content. The function takes a Spinner and the id for a string array as inputs. For more information, please refer at http://developer.android.com/guide/topics/ui/controls/spinner.html#Populate.

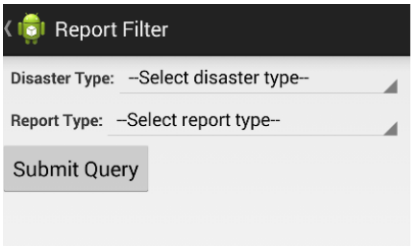
Let's test whether the code works! You can simply test the activity by modify the launch activity in ***AndroiManifest.xml***:

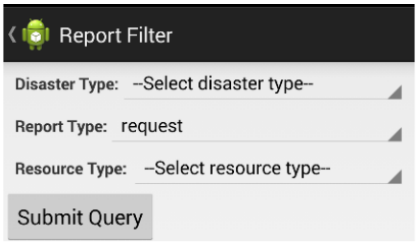
```
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".ReportFilterActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <meta-data
```

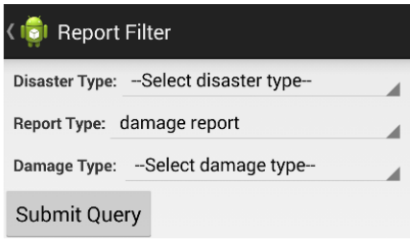
Now run the application and you should see the following screen:



Question 1: At this stage, you have configured the first two spinners. Now you need to configure the hidden LinearLayout. When the user selects a report type, the hidden Layout will show up. If the user selects request or donation report type, the screen should look like the middle figure below. If the user selects damage report type, the screen should look like the right figure below. If the user selects "—Select report type--", the Layout will be hidden again as the left figure below shows. You should use the string arrays we defined previously for the task. In this question, you only need to edit ***ReportFilterActivity.java*** file. Please look up documentations on how to set visibility of a View and text of a TextView. For the spinner selection event handler, you should read the instructions on http://developer.android.com/guide/topics/ui/controls/spinner.html#SelectListener.







*** Please submit your ***ReportFilterActivity.java*** file to dropbox ***

1.3 Activity and Intent

An activity can start another activity and this is how Android applications transit from one page to another page. You can start another activity by calling ***startActivity*** (a public method of **Activity** class), passing it an **Intent** object which describes the activity you want to start. An intent is an abstract description of an operation to be

performed (http://developer.android.com/reference/android/content/Intent.html).

In our application, when the user clicks on "Report Filter" on the menu option bar, the application will go to the Report Filter activity we implemented in the previous section.

To be able to use ***startActivity*** in our application, we need to declare it in ***AndroidManifest.xml*** file:

```
14      <application
15          android:allowBackup="true"
16          android:icon="@drawable/ic_launcher"
17          android:label="@string/app_name"
18          android:theme="@style/AppTheme" >
19          <activity
20              android:name=".MainActivity"
21              android:label="@string/app_name" >
22              <intent-filter>
23                  <action android:name="android.intent.action.MAIN" />
24
25                  <category android:name="android.intent.category.LAUNCHER" />
26              </intent-filter>
27          </activity>
28
29          <activity
30              android:name=".ReportFilterActivity"
31              android:label="Report Filter"
32              android:parentActivityName=".MainActivity" >
33              <meta-data
34                  android:name="android.support.PARENT_ACTIVITY"
35                  android:value=".MainActivity" />
36          </activity>
37
38          <meta-data
39              android:name="com.google.android.gms.version"
40              android:value="@integer/google_play_services_version" />
41
42          <meta-data
43              android:name="com.google.android.geo.API_KEY"
44              android:value="AIzaSyABi8n_DNb5r0ivLT7qVIzkAoN2LtNdnQw"/>
```

Refer the code in AndroidManifest_1.xml

Code explanation:

Line 32: The class name of the logical parent of the activity. The android system reads this attribute to determine which activity should be started when the user presses the "Back" button in the navigation bar. For more information, please refer at http://developer.android.com/guide/topics/manifest/activity-element.html.

Line: 33-35: Used to support API 4-16. For more information, please refer at http://developer.android.com/guide/topics/manifest/activity-element.html.

Next, let's edit the action bar layout file under ***res/menu/main.xml*** to show the action button for report filter:

```
1  <menu xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      tools:context="org.disasterproject4a.MainActivity" >
4
5      <item
6          android:id="@+id/report_filter"
7          android:orderInCategory="100"
8          android:showAsAction="never"
9          android:title="@string/report_filter"/>
10
11 </menu>
12
```

Refer the code in main_1.xml

Code explanation:

Line 1: Define a menu. Menus are a common user interface component in many types of applications (http://developer.android.com/guide/topics/ui/menus.html).

Line 5: A menu can have many menu items.

Line 7: The "showAsAction" attribute defines the order of "importance" of the item, within a group. http://developer.android.com/guide/topics/resources/menu-resource.html

Line 8: The "showAsAction" attribute defines when and how this item should appear as an action item in the app bar. "Never" means never place this item in the app bar. Instead, list the item in the app bar's overflow menu.

Line 9: The item title as a string resource. Note you will need to add the string resource in ***strings.xml*** like the figure below shows:

```
3
4      <string name="app_name">Disaster Management</string>
5      <string name="report_filter">Report Filter</string>
6      <string name="disaster_query_label">Disaster Type: </string>
7      <string name="report_type_query_label">Report Type: </string>
```

Refer the code in strings_5.xml

Now edit the code in ***MainActivity*** to start the ***ReportFilterActivity*** when clicks on the menu item:

```
69   @Override
70   public boolean onOptionsItemSelected(MenuItem item) {
71     // Handle action bar item clicks here. The action bar will
72     // automatically handle clicks on the Home/Up button, so long
73     // as you specify a parent activity in AndroidManifest.xml.
74     int id = item.getItemId();
75     if (id == R.id.report_filter) {
76       Intent intent = new Intent(this, ReportFilterActivity.class);
77       startActivity(intent);
78       return true;
79     }
80     return super.onOptionsItemSelected(item);
81   }
82
83 }
```

Refer the code in MainActivity_1.java

Code explanation:

Line 69: This method is called whenever an item in your options menu is selected (http://developer.android.com/reference/android/app/Activity.html#onOptionsItemSelected(android.view.MenuItem).

Line 74-76: If selected the "Report Filter" item, start the ***ReportFilterActivity***.

Line 75: Create an Intent with the ***MainActivity*** as the context and ***ReportFilterActivity*** as the target activity.

For more information of the constructor method, please refer at http://developer.android.com/reference/android/content/Intent.html#Intent(android.content.Context, java.lang.Class<?>).

1.4 Pass data Between Activities

Now you should be able to transfer between the two activities. There is one thing left for the ***ReportFilterActivity***: the submit query button action. When the user clicks on the button, the ***MainActivity*** will show up. In this process, some data (containing the query detail) is passed from ***ReportFilterActivity*** to the ***MainActivity***. When the ***MainActivity*** receives the data, it will then send the query to the server using ***AsycHttpPost*** and update the map. Only reports satisfying the query conditions will be shown on the map.

To make it convenient for using ***AsycHttpPost***, the data passed from ***ReportFilterActivity*** to ***MainActivity*** will be of type HashMap<String, String> representing the key-value pairs to be sent along with the HTTP POST request.

In last section, we used ***startActivity*** to start the ***ReportFilterActivity***. Sometimes, you might want to receive a result (i.e. in our case, "result" refers to the data to be passed between activities) from the activity that you start. In that case, start the activity by calling ***startActivityForResult***. To then receive the result from the subsequent activity (i.e. ***ReportFilterActivity***), implement the ***onActivityResult*** callback method (in ***startActivity***). When the subsequent activity is done, it returns a result in an Intent to your ***onActivityResult*** method (http://developer.android.com/guide/components/activities.html#StartingAnActivityForResult). Now let's edit the ***MainActivity*** as follows:

Refer the code in MainActivity_2.java

Code explanation:

Line 78: Change ***startActivity*** to ***startActivityForResult***. The second parameter of the method ***startActivityForResult*** is the request code. This code will be returned in ***onActivityResult*** when the ***ReportFilterActivity*** ended. The code is to identify the ended activity in ***onActivityResult*** since there can be more than one activity started by ***MainActivity***.We defined the code QUERY_REPORT_REQUEST at the beginning of the class. Another code corresponding to another activity will be defined later. For more information about the method ***startActivityForResult***, please refer to http://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int).

Line 86: This method is called when an activity you launched exits, giving you the requestCode you started it with, the resultCode it returned, and any additional data from it (http://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)).

Line 88: Verify the request code.

Line 90-91: Remove all items in the cluster manager on the Google Map and recluster (https://github.com/googlemaps/android-maps-utils/issues/90). The cluster manager is used to manage marker clusters in Google Map. You should have already implemented it in Lab 5.

Line 94: Get the additional data sent from ***ReportFilterActivity*** with key "data".

Line 96-97: Use the data to make the HTTP POST request to query reports and add the reports to the Google Map. Since this time the data defines what kinds of reports to query (e.g. donation reports for hurricane events), there will be less markers on the map.

Line 27: Define a static string representing the URL to send the POST request to.

Next step is to edit the ***ReportFitlerActivity*** for the submit query button on-click event. Recall we hooked method "submitQuery" to the submit button in ***activity_report_filter.xml***. Now let's create this method in ***ReportFilterActivity***:

```
47  public void submitQuery(View view) {
48    int p1 = mDisasterSpinner.getSelectedItemPosition();
49    int p2 = mReportSpinner.getSelectedItemPosition();
50
51    Intent returnIntent = new Intent();
52    HashMap<String, String> data = new HashMap<String,String>();
53    data.put("tab_id","1");
54    if (p1!=0){
55      String text = mDisasterSpinner.getSelectedItem().toString();
56      data.put("disaster_type",text);
57    }
58    if (p2!=0) {
59      String text = mReportSpinner.getSelectedItem().toString();
60      data.put("report_type",text);
61    }
62
63    returnIntent.putExtra("data", data);
64    setResult(Activity.RESULT_OK,returnIntent);
65    finish();
66  }
```

Refer the code in ReportFilterActivity_2.java

Code explanation:

Line 47: This method is called when the user clicks on "Submit Query" button.
Line 48-49: Get the positions of selected items in two spinners.
Line 52-53: Create the data to be passed to ***MainActivity***. Refer to previous labs to understand why we need to add the key "tab_id".
Line 54: We need this condition statement because the position "0" is the placeholder text of the spinner (e.g. "---Select disastet type---").
Line 55-56: Get the spinner value and add to the "data" object (http://stackoverflow.com/questions/1947933/how-to-get-spinner-value).
Line 63: Put extra data to the intent to be returned.
Line 64: When an activity exits, it can call ***setResult*** to return data back to its parent. It must always supply a result code. Here the code is "Activity.RESULT_OK". The second parameter is the intent to propagate back to the originating activity (http://developer.android.com/reference/android/app/Activity.html#setResult(int,%20android.content.Intent)).
Line 65: Exit the ***ReportFilterActivity***.

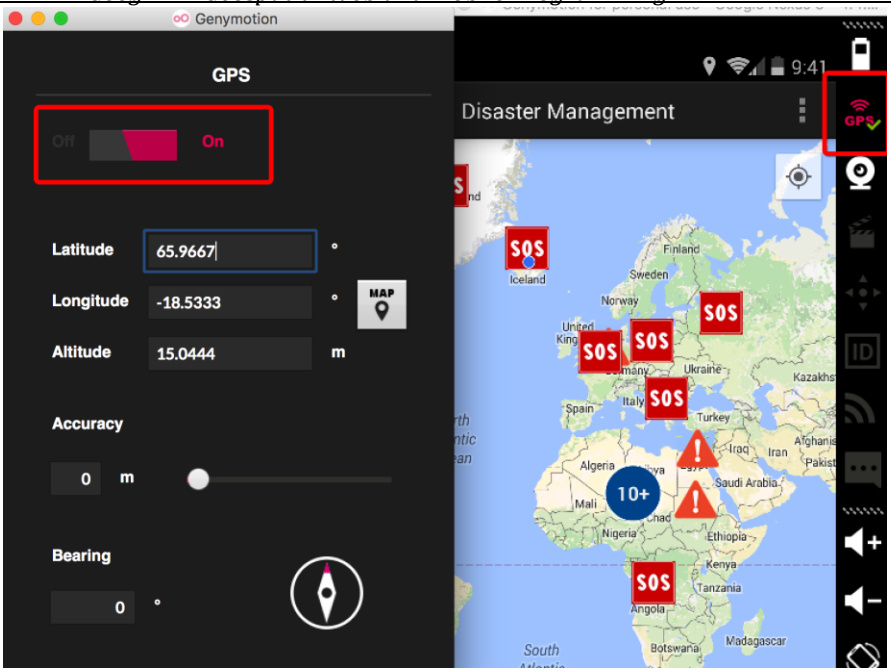
You also need to insert "mMap.clear();//add this to clear the existing markers" in the first line of method "onQueryReportExecute" in AsyncHttpPost.java.

Congratulations! Now you should be able to submit query on your need and see the results on Google Map.!

2. Submit Report

2.1 Enable Location-Based Service (LBS)

In this section, you will implement the submit report function just like you did in the web version in Lab 4. First you should enable Location-Based Service (LBS) on your device. Use Genymotion, you should click on the side toolbar to enable GPS as follows:



Now open your ***AndroidManifest.xml*** and add two more permissions to access the user's location (https://developers.google.com/maps/documentation/android-api/location?hl=en):

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.disasterproject4a"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

Then enable the My Location layer in ***MainActivity.java***'s ***onCreate*** method:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  map = ((MapFragment) getFragmentManager().findFragmentById(R.id.map))
      .getMap();

  mClusterManager = new ClusterManager<ReportClusterItem>(this, map);
  mClusterManager.setRenderer(new ReportRenderer());
  map.setOnCameraChangeListener(mClusterManager);

  map.setMyLocationEnabled(true);

  HashMap<String, String> data = new HashMap<String, String>();
  data.put("tab_id", "1");
  AsyncHttpPost asyncHttpPost = new AsyncHttpPost(data,map,mClusterManager);
  asyncHttpPost.execute(servletURL);

}
```

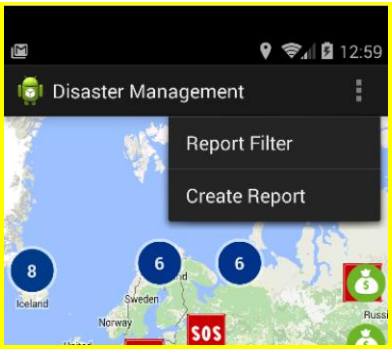
Now you are able to access the user's current location for report submission.

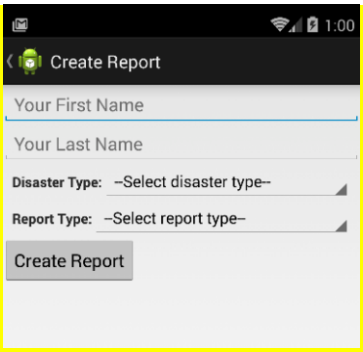
2.2 Layout

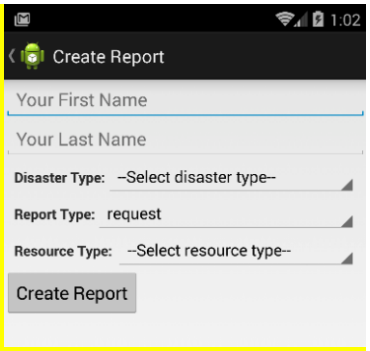
To create the create report functionality, we first need to add an item to the menu bar option list. Next, we need

to create another layout for the new activity. Please read the following instruction to complete question 2.

Question 2: In this question, you will focus on the layout files for the create report feature. Your option menu should look like the figure (a) below. You should also create an ***activity_create_report.xml*** file under res/layout representing the new activity. The layout is similar to the ReportFilter layout. In the layout, there should be two ***EditText*** fields for name input, three spinners (one is hidden) and one button. ***EditText*** fields allow text input and you can specify placeholder text. For the button's ***onClick*** attribute, you should use "createReport" method instead of "submitQuery". Your layout should look like the figure (b) and (c).

(a) 

(b) 

(c) 

*** Please submit the related files (e.g. main.xml, string.xml, activity_create_report.xml) to dropbox ***

2.3 Create Activity

Now we need to create a new activity for creating report. When the user clicks on "Create Report" button, a new report is created in the database and shown on the Google Map. In this stage, you should be able to do it by yourself.

Question 3: Create a new activity ***CreateReportActivity*** under the same package as ***ReportFilterActivity***. Add the activity in your Manifest. You should create a ***createReport*** method in the activity and it will be called when the user clicks on the button. Then the ***CreateReportActivity*** exits and you will go back to its parent activity, the ***MainActivity***, with the new report you just created. Remember how we learned to use ***startActivityForResult***, when the user clicks on "Create Report" item in ***MainActivity***'s menu bar, it will start the ***CreateReportActivity*** with CREATE_REPORT_REQUEST code. Define the code in the beginning of the class and assign it integer value 2. As you can see in the figures in Question 2, there is no input fields for coordinates. In our application, we can use ***map.getMyLocation().getLatitude()/getLongitude()*** for the "latitude" and "longitude" fields. Make sure every report created has a coordinate or your will lose points! In this question, you should also show the hidden spinner if the report type is selected. You should have already done that in Question 1 and be able to adapt to the ***CreateReportActivity***.

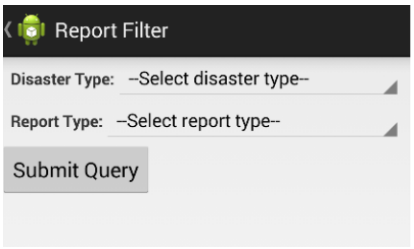
Hint: 1) For report creation, "tab_id" should be value "0" (refer to Lab 4); 2) You should modify ***AsyncHttpPost*** to let it be able to handle both query report and create report HTTP requests; 3) You should "refresh" the map after creating a report. Think of how to do it and where to add the code.

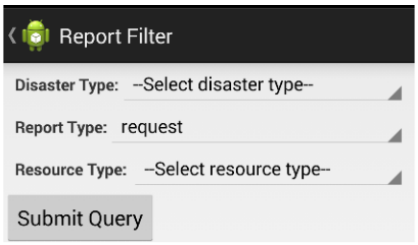
*** Please submit all related files (e.g. MainActivity.java, CreateReportActivity.java, AsycHttpPost.java, and AndroidManifest.xml) to dropbox ***

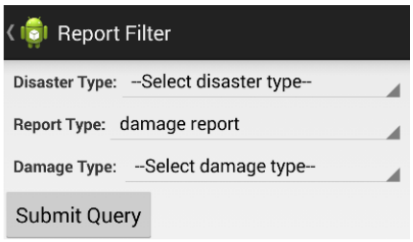
Part 2: Questions

Question 1: 30 points

At this stage, you have configured the first two spinners. Now you need to configure the hidden LinearLayout. When the user selects a report type, the hidden Layout will show up. If the user selects request or donation report type, the screen should look like the middle figure below. If the user selects damage report type, the screen should look like the right figure below. If the user selects "—Select report type--", the Layout will be hidden again as the left figure below shows. You should use the string arrays we defined previously for the task. In this question, you only need to edit ***ReportFilterActivity.java*** file. Please look up documentations on how to set visibility of a View and text of a TextView. For the spinner selection event handler, you should read the instructions on http://developer.android.com/guide/topics/ui/controls/spinner.html#SelectListener.





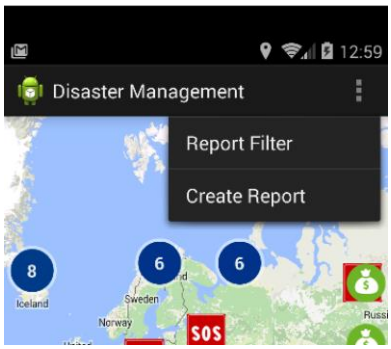


*** Please submit your ***ReportFilterActivity.java*** file to dropbox ***

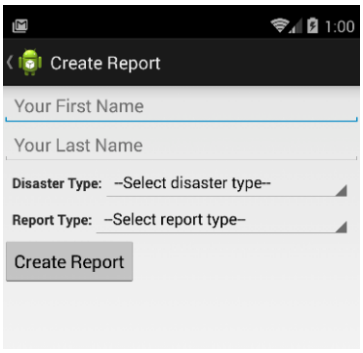
Question 2: 20 points

In this question, you will focus on the layout files for the create report feature. Your option menu should look like the figure (a) below. You should also create an ***activity_create_report.xml*** file under res/layout representing the new activity. The layout is similar to the ReportFilter layout. In the layout, there should be two ***EditText*** fields for name input, three spinners (one is hidden) and one button. ***EditText*** fields allow text input and you can specify placeholder text. For the button's ***onClick*** attribute, you should use "createReport" method instead of "submitQuery". Your layout should look like the figure (b) and (c).

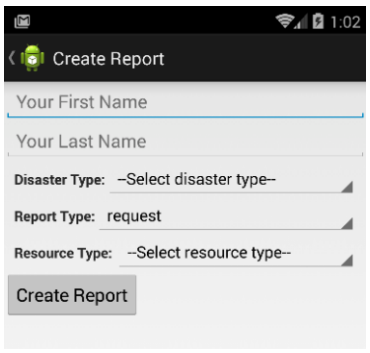
(a)



(b)



(c)



*** Please submit the related files (e.g. main.xml, string.xml, activity_create_report.xml) to dropbox ***

Question 3: 50 points

Create a new activity ***CreateReportActivity*** under the same package as ***ReportFilterActivity***. Add the activity in your Manifest. You should create a ***createReport*** method in the activity and it will be called when the user clicks on the button. Then the ***CreateReportActivity*** exits and you will go back to its parent activity, the ***MainActivity***, with the new report you just created. Remember how we learned to use ***startActivityForResult***, when the user clicks on "Create Report" item in ***MainActivity***'s menu bar, it will start the ***CreateReportActivity*** with CREATE_REPORT_REQUEST code. Define the code in the beginning of the class and assign it integer

value 2. As you can see in the figures in Question 2, there is no input fields for coordinates. In our application, we can use ***map.getMyLocation().getLatitude()/getLongitude()*** for the "latitude" and "longitude" fields. Make sure every report created has a coordinate or your will lose points! In this question, you should also show the hidden spinner if the report type is selected. You should have already done that in Question 1 and be able to adapt to the ***CreateReportActivity***.

Hint: 1) For report creation, "tab_id" should be value "0" (refer to Lab 4); 2) You should modify ***AsyncHttpPost*** to let it be able to handle both query report and create report HTTP requests; 3) You should "refresh" the map after creating a report. Think of how to do it and where to add the code.

*** Please submit all related files (e.g. MainActivity.java, CreateReportActivity.java, AsycHttpPost.java, and AndroidManifest.xml) to dropbox ***