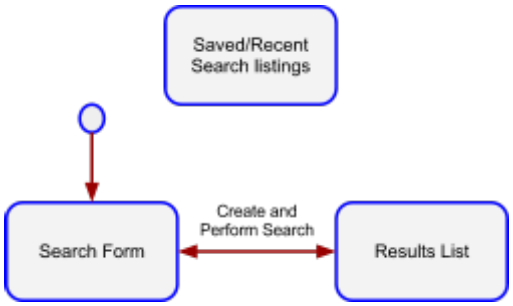
CSI319-51
Assignment 2
Due 1 October 2012

Assignment 2

Continuing from the previous assignment, you should have the following workflow implement:



We will keep the existing workflow with this assignment. As the current implementation is quite boring, we will connect to the 3taps web service to work with *real* data.

Part 1

Pretty up that UI!

Update the Search Form layout to support the following:

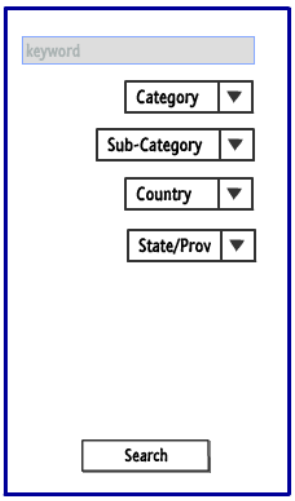
- EditText View for search ***keyword*** *(From Assignment 1)*
- Spinner Widget for ***category*** *(From Assignment 1)*
- Spinner Widget for ***sub-category***
- Spinner Widget for ***Country***
- Spinner Widget for ***State/Providence***
- Button for submitting the form and performing a search *(From Assignment 1)*

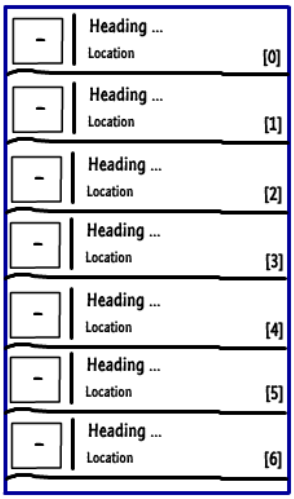
Update the Item layout used by your Adapter in ResultsListFragment to support the following:

- ImageView for a thumbnail (72dp x 72dp)
- A separator (vertical line)
- TextView for heading
- TextView for location
- TextView for an index to the position in the result list

Note: If you did not extend an Adapter in Assignment 1, you will have to for this assignment.

Rough concepts:





Be sure to update your SearchCriteria and SearchResult classes to support the following:

SearchCriteria
- keyword
- category
- subCategory
- country
- state

SearchResult
- heading
- location
- imageUrl

For now, these can all be Strings.

Part 2

To keep this simple, the Spinners (category, subCategory, country, and state) will be populated with the HashMaps provided (Locations.java and Categories.java). Place these classes in a new sub-package called *content*.

*edu.champlain.csi319.findstuff.***content**
Categories.java
Locations.java
*edu.champlain.csi319.findstuff.***model**
SearchCriteria.java
SearchResult.java
*edu.champlain.csi319.findstuff.***ui**
*edu.champlain.csi319.findstuff.***searchform**
SearchFormActivity.java
SearchFormFragment.java
*edu.champlain.csi319.findstuff.***resultslist**
ResultsListActivity.java
ResultsListFragment.java
MainActivity.java

The *subCategory* and *state* Spinners must be dynamic. The items populated in these Spinners will depend on *category* and *country* respectively. For example, when selecting category “For Sale”, the subCategory Spinner should display “Electronics & Photo” and “Furniture”.

The Maps provide the following keys:

Category	Sub-Categories
For Sale	• Electronics & Photo • Furniture
Real Estate	• Housing For Sale • Housing For Rent

Country	State/Province
USA	*(All 50 States)*
Canada	*(All 10 Provinces)*

Upon form submission, you must bind the values associated to the keys selected to your SearchCriteria object.
For example, if you were to search for Pink Sofas for sale in Vermont, the SearchCriteria fields would be as follows:

SearchCriteria

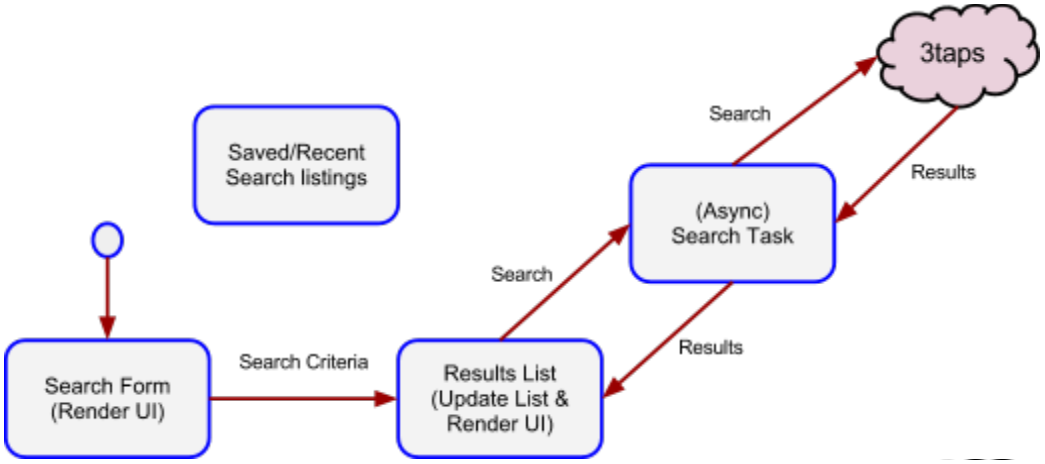
- *keyword* = “Pink Sofa”
- *category* = “S”
- *subCategory* = “FUR”
- *country* = “USA”
- *state* = “VT”

Part 3

Visit 3taps and register as a developer to get an API key: http://register.3taps.com/member/register

Test your API key in a browser: http://3taps.net/search?authID=**KEY**&source=CRAIG&category=SFUR
You should see a JSON string consisting of some search results.

Don’t block the UI!!!
The idea is that when the user submits a search form with some criteria, that criteria is passed to the ResultsListFragment which starts an asynchronous task to query 3taps for 10 results at a time. The newly found results would then be appended to the UI’s list.



Update the adapter used by ResultsListFragment to support the UI changes in **Part 1**. For this part, you can ignore setting the ImageView for the thumbnail, and instead leave it as a placeholder in the UI.

Provided is a class which performs the search: **SearchTask.java**. Place this class under the *content* sub-package.
Be sure to update the *AUTH_ID* field with your 3taps API key.

Leverage the LoaderManager.LoaderCallbacks interface to manage your asynchronous search task by implementing `LoaderManager.LoaderCallbacks<List<SearchResult>>` in ResultsListFragment. This interface takes a generic type as the loader data. In our case, we want our loader to provide a List of SearchResults, hence the `List<SearchResult>`.

The ResultsListFragment will initialize the loader. When the loader is created, it will instantiate the SearchTask. When the SearchTask has completed (onLoadFinished), the UI will need to be updated to display the new results in the list. To keep this simple, cap the search results at 100.

If no results are found, you should display a “No results found” TextView in the UI.
Also make sure progress indicators are shown when appropriate (like when first loading!).

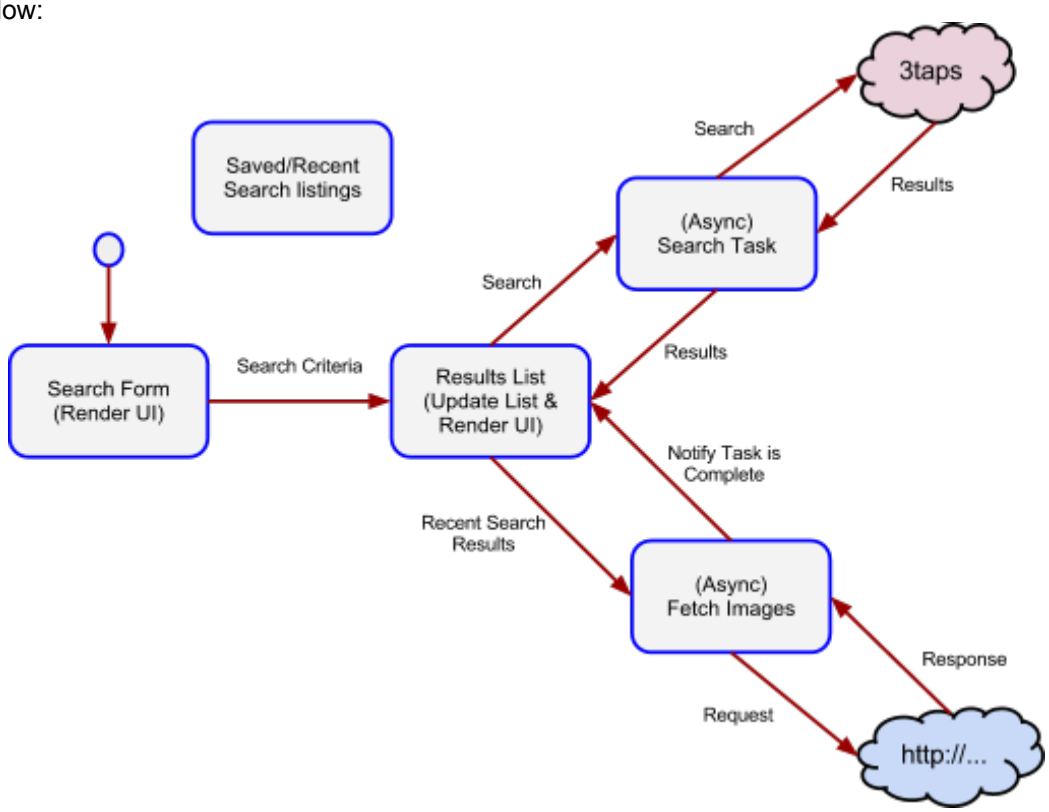
Part 4

Time to get a thumbnail in the ResultsList's ImageView. When querying 3taps for search results, image urls are returned in the JSON string. You need to create an asynchronous task to fetch and decode an image from a url. This image will be resized to fit our 72dpx72dp ImageView.

As the fetching of the image is occurring on a separate thread, you need a way to indicate to the user that an image is being loaded. To do this, provide a ProgressBar in the UI that will overlay the ImageView. The ProgressBar should be visible until an image is fetched.

But what if there is no image url? If no images exist for a result, you can leave it empty and just make the ProgressBar invisible.

High level workflow:



Now... how to approach this:
1. Start by adding the provided **HttpImageLoader.java** to a sub-package called *util*. This contains the static methods needed to decode an http input stream to an appropriately sized Bitmap.

2. Add a new field to the SearchResult class so that you can store the thumbnail.
This should be of type Bitmap.

SearchResult
- heading
- location
- imageUrl
- thumbnail

3. Create a new class to asynchronously load images from url's. This class should be placed in the *content* sub-package and should extend AsyncTask. This class should expect *Params* of type SearchResult. This way the background task can simply update the SearchResult with the thumbnail:

```
result.setThumbnail(HttpImageLoader.decodeUrl(urlString, width, height));
```

4. Modify ResultsListFragment to pass an array of the recently added results to a new instance of your asynchronous image loader and invoke it with parallel execution enabled.

By now, your project should look something like this:

edu.champlain.csi319.findstuff.**content**
- **Categories.java**
- **Locations.java**
- **LoadImageUrlTask.java** *(or whatever you named it)*
- **SearchTask.java**

edu.champlain.csi319.findstuff.**model**
- **SearchCriteria.java**
- **SearchResult.java**

edu.champlain.csi319.findstuff.**ui**
- *edu.champlain.csi319.findstuff*.**searchform**
  - **SearchFormActivity.java**
  - **SearchFormFragment.java**
- *edu.champlain.csi319.findstuff*.**resultslist**
  - **ResultsListActivity.java**
  - **ResultsListFragment.java**
  - **ResultsListItemAdapter.java** *(or whatever you named it)*
- **MainActivity.java**

edu.champlain.csi319.findstuff.**util**
- **HttpImageLoader.java**

Assignment Submission

Submit an exported archive (zip or tar) of your project in Angel. Please name the archive as follows: ***lastname*_assignment2.tar** or ***lastname*_assignment2.zip**

Do not hesitate to ask questions!