HTTP and Background Tasks-17

In this assignment, you will continue working on your Soo Greyhounds app. We will fetch and display the most recent photos from the Soo Greyhounds web service. In this lab you will get the basics of downloading and parsing JSON up and running. You will also get a RecyclerView setup that will eventually show images.

1. Create a new repository on GitHub and add your instructor as a collaborator.

2. Open your Soo Greyhounds app in Android Studio. Ensure you have submitted your previous lab. Remove the remote repository that is pointing to your previous lab and add the remote to your new repository. Perform a push to ensure everything is setup properly.

3. We are going to setup a new main screen that we will use for the next few labs. Let's create that new activity now. Right click on your main app package, put your mouse over "Now", then "Activity", and click "Empty Activity". Give the activity a name of "PhotoGalleryActivity".

4. PhotoGalleryActivity will be a SingleFragmentActivity subclass like we did with Criminal Intent (and as we learned, is a very common way to setup apps to reduce the amount of code we have to write/manage). Also, its view will be the container view (as we did with Criminal Intent) and will be defined in a layout file called activity_fragment.xml. This activity will host a fragment – in particular, an instance of PhotoGalleryFragment, which you will create shortly. Copy SingleFragmentActivity.java and activity_fragment.xml into your project from your Criminal Intent project (this is good practice as copying code from previous project is very common in the app development world).

5. Due to the fact our Soo Greyhounds app has a different package name than Criminal Intent, we will need to update our package name in SingleFragmentActivity.java. Open SingleFragmentActivity.java and change the package name (if you did not name your package com.soogreyhounds.soogreyhoundsmobile ensure you change the code below to be your package name). You may also have to update your imports if you are using the newer version of Android Studio and used the older version with Criminal Intent.

   ~~**package com.bignerdranch.android.criminalintent;**~~
   package com.soogreyhounds.soogreyhoundsmobile;

   import android.os.Bundle;

   import androidx.appcompat.app.AppCompatActivity;
   import androidx.fragment.app.Fragment;
   import androidx.fragment.app.FragmentManager;

   ```
   public abstract class SingleFragmentActivity extends AppCompatActivity {
       …
   }
   ```

6. In PhotoGalleryActivity.java, set up PhotoGalleryActivity as a SingleFragmentActivity by deleting the code that the template generated and replacing it with an implementation of createFragment(). Have createFragment() return an instance of PhotoGalleryFragment. (you will get an error, it will go away after you create the PhotoGalleryFragment class.)

```
public class PhotoGalleryActivity extends ~~Activity~~ SingleFragmentActivity {
   @Override
   protected Fragment createFragment() {
      return PhotoGalleryFragment.newInstance();
   }

   ~~@Override~~
   ~~protected void onCreate(Bundle savedInstanceState) {~~
      ~~/* Auto-generated template code... */~~
   ~~}~~
}
```

7. PhotoGallery will display its results in a RecyclerView, using the built-in GridLayoutManager to arrange the items in a grid. Add a new layout resource called “fragment_photo_gallery.xml” and add the RecyclerView to it by replacing the content of the layout file with the content below.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.recyclerview.widget.RecyclerView
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/photo_recycler_view"
   android:layout_height="match_parent"
   android:layout_width="match_parent" />
```

8. Create the PhotoGalleryFragment class. Retain the fragment, inflate the layout you just created, and initialize a member variable referencing the RecyclerView.

```
public class PhotoGalleryFragment extends Fragment {
  private RecyclerView mPhotoRecyclerView;

  public static PhotoGalleryFragment newInstance() {
    return new PhotoGalleryFragment();
  }

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
  }

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

    mPhotoRecyclerView = (RecyclerView) v.findViewById(R.id.photo_recycler_view);
    mPhotoRecyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));

    return v;
  }
}
```

9. We want to run our app to ensure there are no errors, however before we can do that we have to tell Android to run our PhotoGalleryActivity as the start up activity instead of our MainActivity. Go over to your app’s manifest and change the launcher activity.

```
<activity android:name=".PhotoGalleryActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>

<activity android:name=".MainActivity">
  ~~<intent-filter>~~
    ~~<action android:name="android.intent.action.MAIN" />~~
    ~~<category android:name="android.intent.category.LAUNCHER" />~~
  ~~</intent-filter>~~
</activity>
```

10. Run Soo Greyhounds Mobile to make sure everything is wired up correctly before moving on. If all is well, you will see a blank screen without any crashing or errors.

11. You are going to have one class handle the networking in Soo Greyhounds Mobile. Create a new Java class called SooGreyhoundsAPI.

12. SooGreyhoundsAPI will start off small with only two methods: getUrlBytes(String) and getUrlString(String). The getUrlBytes(String) method fetches raw data from a URL and returns it as an array of bytes. The getUrlString(String) method converts the result from getUrlBytes(String) to a String. In SooGreyhoundsAPI.java, add implementations for getUrlBytes(String) and getUrlString(String)

```
public class SooGreyhoundsAPI {
  public byte[] getUrlBytes(String urlSpec) throws IOException {
    URL url = new URL(urlSpec);
    HttpURLConnection connection = (HttpURLConnection)url.openConnection();

    try {
      ByteArrayOutputStream out = new ByteArrayOutputStream();
      InputStream in = connection.getInputStream();

      if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
        throw new IOException(connection.getResponseMessage() +
            ": with " +
            urlSpec);
      }

      int bytesRead = 0;
      byte[] buffer = new byte[1024];
      while ((bytesRead = in.read(buffer)) > 0) {
        out.write(buffer, 0, bytesRead);
      }
      out.close();
      return out.toByteArray();
    } finally {
      connection.disconnect();
    }
  }

  public String getUrlString(String urlSpec) throws IOException {
    return new String(getUrlBytes(urlSpec));
  }
}
```

13. In PhotoGalleryFragment.java, add a new inner class called FetchItemsTask at the bottom of PhotoGalleryFragment. Override AsyncTask.doInBackground(…) to get data from a website and log it.

```
public class PhotoGalleryFragment extends Fragment {
   private static final String TAG = "PhotoGalleryFragment";

   private RecyclerView mPhotoRecyclerView;
   ...
   private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
      @Override
      protected Void doInBackground(Void... params) {
         try {
            String result = new SooGreyhoundsAPI()
                  .getUrlString("https://www.bignerdranch.com");
            Log.i(TAG, "Fetched contents of URL: " + result);
         } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch URL: ", ioe);
         }
         return null;
      }
   }
}
```

14. Now, in PhotoGalleryFragment.onCreate(…), call execute() on a new instance of FetchItemsTask.

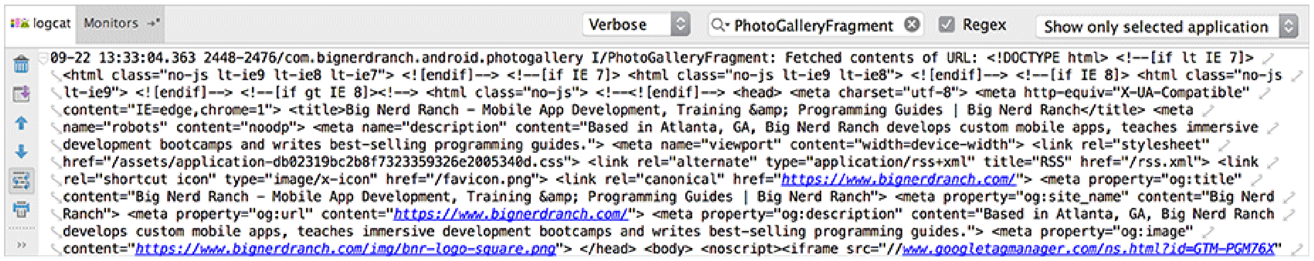
```
public class PhotoGalleryFragment extends Fragment {
   private static final String TAG = "PhotoGalleryFragment";
   private RecyclerView mPhotoRecyclerView;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setRetainInstance(true);
      new FetchItemsTask().execute();
   }
   ...
}
```

15. Tell Android Studio you will need permission to use the internet by adding a uses-permission tag in AndroidManifest.xml.

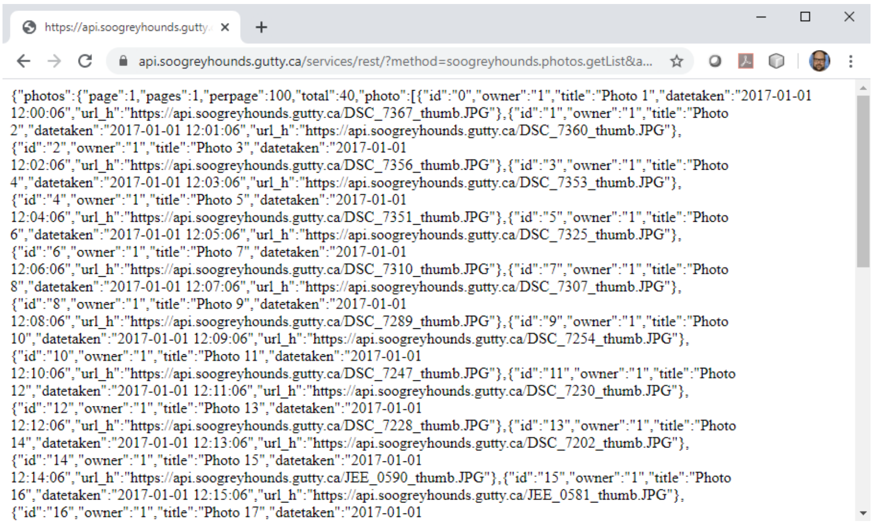
```
<manifest …
  <uses-permission android:name="android.permission.INTERNET" />
  …
</manifest>
```

16. The call to execute() will start your AsyncTask, which will then fire up its background thread and call doInBackground(…). Run your code and you should see the Big Nerd Ranch home page HTML pop up in Logcat. Finding your log statements within the Logcat window can be tricky. It helps to search for something specific. In this case, enter “PhotoGalleryFragment” into the Logcat search box, as shown.



```
09-22 13:33:04.363 2448-2476/com.bignerdranch.android.photogallery I/PhotoGalleryFragment: Fetched contents of URL: <!DOCTYPE html> <!--[if lt IE 7]>
<html class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]--> <!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8"> <![endif]--> <!--[if IE 8]> <html class="no-js
lt-ie9"> <![endif]--> <!--[if gt IE 8]><!--> <html class="no-js"> <!--<![endif]--> <head> <meta charset="utf-8"> <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1"> <title>Big Nerd Ranch - Mobile App Development, Training &amp; Programming Guides | Big Nerd Ranch</title> <meta
name="robots" content="noodp"> <meta name="description" content="Based in Atlanta, GA, Big Nerd Ranch develops custom mobile apps, teaches immersive
development bootcamps and writes best-selling programming guides."> <meta name="viewport" content="width=device-width"> <link rel="stylesheet"
href="/assets/application-db02319bc2b8f7323359326e2005340d.css"> <link rel="alternate" type="application/rss+xml" title="RSS" href="/rss.xml"> <link
rel="shortcut icon" type="image/x-icon" href="/favicon.png"> <link rel="canonical" href="https://www.bignerdranch.com/"> <meta property="og:title"
content="Big Nerd Ranch - Mobile App Development, Training &amp; Programming Guides | Big Nerd Ranch"> <meta property="og:site_name" content="Big Nerd
Ranch"> <meta property="og:url" content="https://www.bignerdranch.com/"> <meta property="og:description" content="Based in Atlanta, GA, Big Nerd Ranch
develops custom mobile apps, teaches immersive development bootcamps and writes best-selling programming guides."> <meta property="og:image"
content="https://www.bignerdranch.com/img/bnr-logo-square.png"> </head> <body> <noscript><iframe src="//www.googletagmanager.com/ns.html?id=GTM-PGM76X"
```

17. Next we will make a request to the Soo Greyhounds web service. Your GET request URL will look something like this: https://api.soogreyhounds.gutty.ca/services/rest/?method=soogreyhounds.photos.getList&api_key=a6d819499131071f158fd740860a5a88&extras=url_h,date_taken&format=json&nojsoncallback=1. Copy this URL into your browser. This will allow you to see an example of what the response data will look like. This is sometimes helpful when you are troubleshooting or learning about a given web service.


https://api.soogreyhounds.gutty.
api.soogreyhounds.gutty.ca/services/rest/?method=soogreyhounds.photos.getList&a...

```
{"photos":{"page":1,"pages":1,"perpage":100,"total":40,"photo":[{"id":"0","owner":"1","title":"Photo 1","datetaken":"2017-01-01
12:00:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7367_thumb.JPG"},{"id":"1","owner":"1","title":"Photo
2","datetaken":"2017-01-01 12:01:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7360_thumb.JPG"},
{"id":"2","owner":"1","title":"Photo 3","datetaken":"2017-01-01
12:02:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7356_thumb.JPG"},{"id":"3","owner":"1","title":"Photo
4","datetaken":"2017-01-01 12:03:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7353_thumb.JPG"},
{"id":"4","owner":"1","title":"Photo 5","datetaken":"2017-01-01
12:04:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7351_thumb.JPG"},{"id":"5","owner":"1","title":"Photo
6","datetaken":"2017-01-01 12:05:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7325_thumb.JPG"},
{"id":"6","owner":"1","title":"Photo 7","datetaken":"2017-01-01
12:06:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7310_thumb.JPG"},{"id":"7","owner":"1","title":"Photo
8","datetaken":"2017-01-01 12:07:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7307_thumb.JPG"},
{"id":"8","owner":"1","title":"Photo 9","datetaken":"2017-01-01
12:08:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7289_thumb.JPG"},{"id":"9","owner":"1","title":"Photo
10","datetaken":"2017-01-01 12:09:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7254_thumb.JPG"},
{"id":"10","owner":"1","title":"Photo 11","datetaken":"2017-01-01
12:10:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7247_thumb.JPG"},{"id":"11","owner":"1","title":"Photo
12","datetaken":"2017-01-01 12:11:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7230_thumb.JPG"},
{"id":"12","owner":"1","title":"Photo 13","datetaken":"2017-01-01
12:12:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7228_thumb.JPG"},{"id":"13","owner":"1","title":"Photo
14","datetaken":"2017-01-01 12:13:06","url_h":"https://api.soogreyhounds.gutty.ca/DSC_7202_thumb.JPG"},
{"id":"14","owner":"1","title":"Photo 15","datetaken":"2017-01-01
12:14:06","url_h":"https://api.soogreyhounds.gutty.ca/JEE_0590_thumb.JPG"},{"id":"15","owner":"1","title":"Photo
16","datetaken":"2017-01-01 12:15:06","url_h":"https://api.soogreyhounds.gutty.ca/JEE_0581_thumb.JPG"},
{"id":"16","owner":"1","title":"Photo 17","datetaken":"2017-01-01
```

18. Time to start coding. First, add some constants to SooGreyhoundsAPI.

```
public class SooGreyhoundsAPI {

    private static final String TAG = "SooGreyhoundsAPI";

    private static final String API_KEY = "a6d819499131071f158fd740860a5a88";
    …
 }
```

19. Now use the constants to write a method that builds an appropriate request URL and fetches its contents.

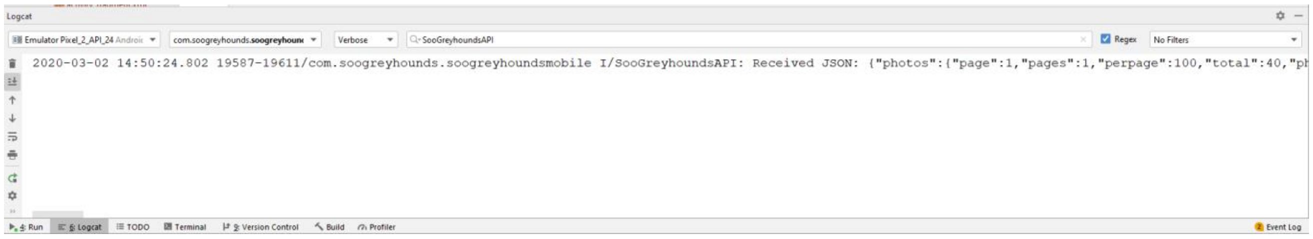
```
public class SooGreyhoundsAPI {
  ...
  public String getUrlString(String urlSpec) throws IOException {
    return new String(getUrlBytes(urlSpec));
  }
  public void fetchItems() {
    try {
      String url = Uri.parse("https://api.soogreyhounds.gutty.ca/services/rest/")
          .buildUpon()
          .appendQueryParameter("method", "soogreyhounds.photos.getList")
          .appendQueryParameter("api_key", API_KEY)
          .appendQueryParameter("format", "json")
          .appendQueryParameter("nojsoncallback", "1")
          .appendQueryParameter("extras", "url_h")
          .build().toString();

      String jsonString = getUrlString(url);
      Log.i(TAG, "Received JSON: " + jsonString);
    } catch (IOException ioe) {
      Log.e(TAG, "Failed to fetch items", ioe);
    }
  }
}
```

20. Modify the AsyncTask in PhotoGalleryFragment to call the new fetchItems() method.

```
public class PhotoGalleryFragment extends Fragment {
  ...
  private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
      @Override
      protected Void doInBackground(Void... params) {
        ~~try {~~
          ~~String result = new SooGreyhoundsAPI()~~
              ~~.getUrlString("https://www.bignerdranch.com");~~
          ~~Log.i(TAG, "Fetched contents of URL: " + result);~~
        ~~} catch (IOException ioe) {~~
          ~~Log.e(TAG, "Failed to fetch URL: ", ioe);~~
        ~~}~~
        new SooGreyhoundsAPI().fetchItems();
        return null;
    }
  }
}
```

21. Run Soo Greyhounds Mobile and you should see JSON in Logcat. (It will help to search for "SooGreyhoundsAPI" in the Logcat search box.)



```
2020-03-02 14:50:24.802 19587-19611/com.soogreyhounds.soogreyhoundsmobile I/SooGreyhoundsAPI: Received JSON: {"photos":{"page":1,"pages":1,"perpage":100,"total":40,"ph
```

22. Create a new class called GalleryItem and add the following code

```
public class GalleryItem {
   private String mCaption;
   private String mId;
   private String mUrl;

   @Override
   public String toString() {
      return mCaption;
   }
}
```

23. Have Android Studio generate getters and setters for mCaption, mId, and mUrl (right click on the class and select generate, then setters and getters).

24. Update fetchItems() in SooGreyhoundsAPI.java to parse the JSON text into corresponding Java objects using the JSONObject(String) constructor.

```
public class SooGreyhoundsAPI {

   private static final String TAG = "SooGreyhoundsAPI";
   ...
   public void fetchItems() {
      try {
         ...
         Log.i(TAG, "Received JSON: " + jsonString);
         JSONObject jsonBody = new JSONObject(jsonString);
      } catch (IOException ioe) {
         Log.e(TAG, "Failed to fetch items", ioe);
      } catch (JSONException je){
         Log.e(TAG, "Failed to parse JSON", je);
      }
   }
}
```

25. Write a method that pulls out information for each photo. Make a GalleryItem for each photo and add it to a List.

```
public class SooGreyhoundsAPI {

  private static final String TAG = "SooGreyhoundsAPI";
  ...
  public void fetchItems() {
    ...
  }

  private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
      throws IOException, JSONException {

    JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
    JSONArray photoJsonArray = photosJsonObject.getJSONArray("photo");

    for (int i = 0; i < photoJsonArray.length(); i++) {
      JSONObject photoJsonObject = photoJsonArray.getJSONObject(i);

      GalleryItem item = new GalleryItem();
      item.setId(photoJsonObject.getString("id"));
      item.setCaption(photoJsonObject.getString("title"));

      if (!photoJsonObject.has("url_h")) {
        continue;
      }

      item.setUrl(photoJsonObject.getString("url_h"));
      items.add(item);
    }
  }
}
```

26. The parseItems(…) method needs a List and JSONObject. Update fetchItems() to call parseItems(…) and return a List of GalleryItems.

```
public ~~void~~ List<GalleryItem> fetchItems() {

    List<GalleryItem> items = new ArrayList<>();

    try {
        String url = ...;
        String jsonString = getUrlString(url);
        Log.i(TAG, "Received JSON: " + jsonString);
        JSONObject jsonBody = new JSONObject(jsonString);
        parseItems(items, jsonBody);
    } catch (JSONException je) {
            Log.e(TAG, "Failed to parse JSON", je);
    } catch (IOException ioe) {
        Log.e(TAG, "Failed to fetch items", ioe);
    }

    return items;
}
```

27. Run Soo Greyhounds Mobile to test your JSON parsing code. Soo Greyhounds Mobile has no way of reporting the contents of your List right now, so you will need to set a breakpoint and use the debugger if you want to make sure everything worked correctly.

28. Let's switch to the view layer and get PhotoGalleryFragment's RecyclerView to display some captions. First define a ViewHolder as an inner class.

```
public class PhotoGalleryFragment extends Fragment {
  ...
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    ...
  }

  private class PhotoHolder extends RecyclerView.ViewHolder {
    private TextView mTitleTextView;

    public PhotoHolder(View itemView) {
      super(itemView);

      mTitleTextView = (TextView) itemView;
    }

    public void bindGalleryItem(GalleryItem item) {
      mTitleTextView.setText(item.toString());
    }
  }

  private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
    ...
  }
}
```

29. Next, add a RecyclerView.Adapter to provide PhotoHolders as needed based on a list of GalleryItems.

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";
    ...
    private class PhotoHolder extends RecyclerView.ViewHolder {
        ...
    }

    private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {

        private List<GalleryItem> mGalleryItems;

        public PhotoAdapter(List<GalleryItem> galleryItems) {
            mGalleryItems = galleryItems;
        }

        @Override
        public PhotoHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
            TextView textView = new TextView(getActivity());
            return new PhotoHolder(textView);
        }

        @Override
        public void onBindViewHolder(PhotoHolder photoHolder, int position) {
            GalleryItem galleryItem = mGalleryItems.get(position);
            photoHolder.bindGalleryItem(galleryItem);
        }

        @Override
        public int getItemCount() {
            return mGalleryItems.size();
        }
    }
    ...
}
```

30. Now that you have the appropriate nuts and bolts in place for RecyclerView, add code to set up and attach an adapter when appropriate.

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;
    private List<GalleryItem> mItems = new ArrayList<>();
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

        mPhotoRecyclerView = (RecyclerView) v.findViewById(R.id.photo_recycler_view);
        mPhotoRecyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));

        setupAdapter();

        return v;
    }

    private void setupAdapter() {
        if (isAdded()) {
            mPhotoRecyclerView.setAdapter(new PhotoAdapter(mItems));
        }
    }
    ...
}
```

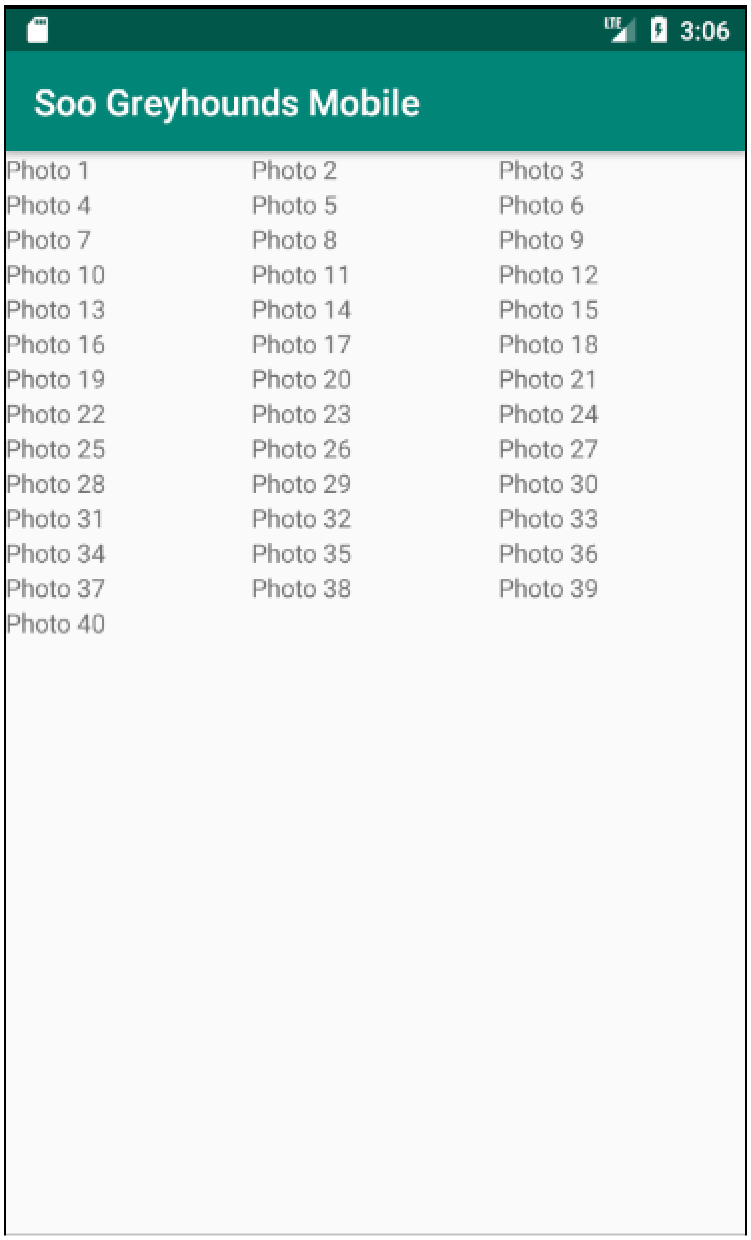
31. Modify FetchItemsTask to update mItems and call setupAdapter() after fetching your photos to update the RecyclerView’s data source.

```
private class FetchItemsTask extends AsyncTask<Void,Void,~~Void~~ List<GalleryItem>> {
    @Override
    protected ~~Void~~ List<GalleryItem> doInBackground(Void... params) {
        return new SooGreyhoundsAPI().fetchItems();
~~        return null;~~
    }

    @Override
    protected void onPostExecute(List<GalleryItem> items) {
        mItems = items;
        setupAdapter();
    }
}
```

32. Run Soo Greyhounds Mobile, and you should see text displayed for each GalleryItem you downloaded.

Final App



```
Photo 1        Photo 2        Photo 3
Photo 4        Photo 5        Photo 6
Photo 7        Photo 8        Photo 9
Photo 10       Photo 11       Photo 12
Photo 13       Photo 14       Photo 15
Photo 16       Photo 17       Photo 18
Photo 19       Photo 20       Photo 21
Photo 22       Photo 23       Photo 24
Photo 25       Photo 26       Photo 27
Photo 28       Photo 29       Photo 30
Photo 31       Photo 32       Photo 33
Photo 34       Photo 35       Photo 36
Photo 37       Photo 38       Photo 39
Photo 40
```