Lab: PhotoBucket (Firebase)

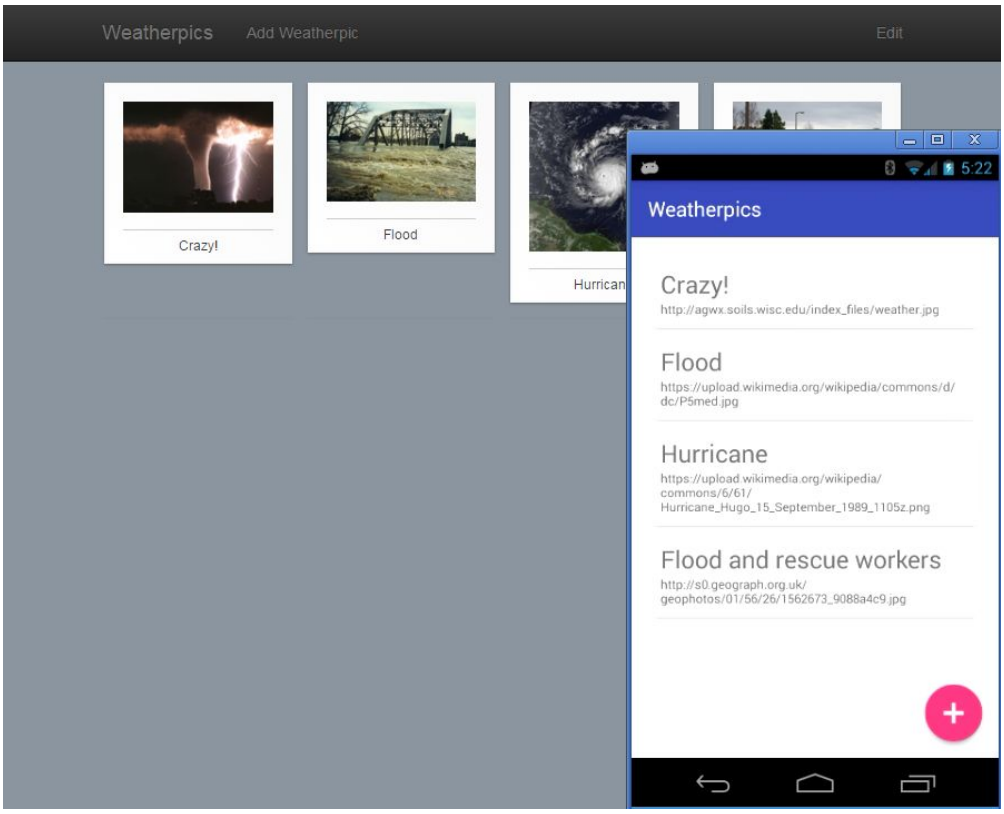
Goal

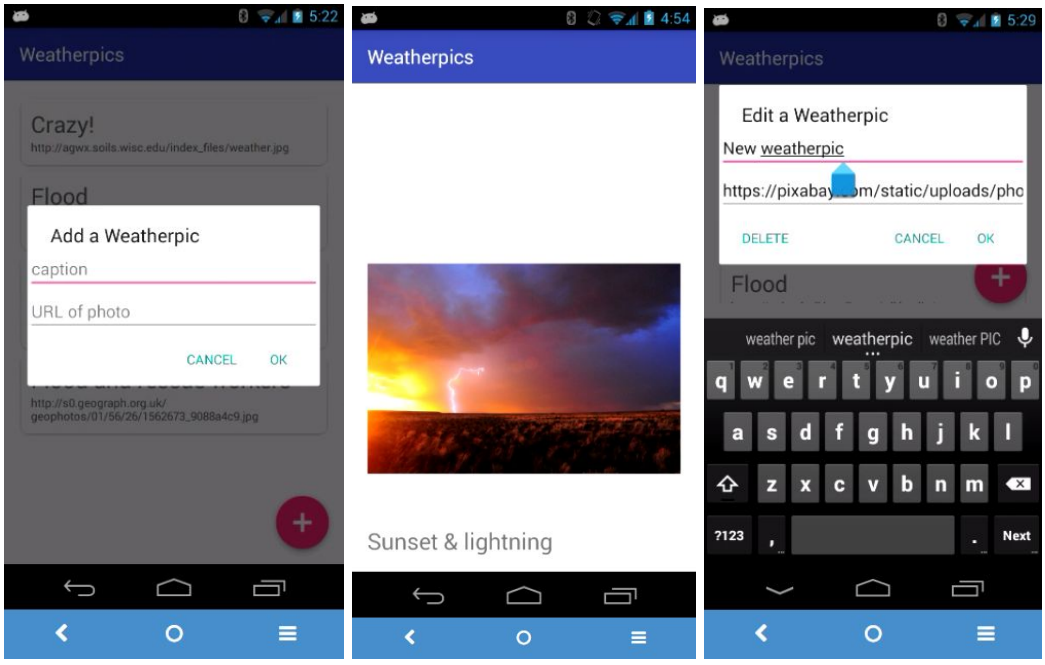
The goal for this lab is to practice using Firebase for cloud storage with a new app. We'll create a PhotoBucket app that uses Firebase for CRUD. Use the Firebase console to see your data. Then use the console to create, update, and delete data and see those changes happening in your app. Note: this lab was formerly known as Weatherpics, since my solution app held photos (I call them pics, short for pictures) of severe weather, inspired the Terre Haute FIRST Lego League team's idea for a a TornadoSafeZone app. If you see references to weather pics, that is why. However, you can choose to include any themed photos (like dogs, cars, etc.) in your app if you are more interested in that theme - hence the new name, *PhotoBucket*.

Requirements

Your app must satisfy the following requirements:

1. Use a Pic class for a model, with caption and URLs.
2. Upon startup, a list of caption and URLs are displayed using a Recycler/CardView in a ListFragment. The list matches those in the Firebase.
3. Pressing the FAB launches a dialog so the user can enter a caption and URL for a pic, This new pic is added both locally and to the Firebase.
4. Pressing an item in the list swaps in a new DetailFragment that displays the name and image for that pic. Pressing back from the detail fragment goes back to the list.
5. Long-pressing an item in the list allows the user the opportunity to edit the name and URL, or to delete the pic. Edits/deletions are local and on the Firebase.
6. Any changes (add, remove, or edit) made remotely on the Firebase console are updated on the app.
7. (**I didn't demo this in video**) The title of the app is stored in Firebase. Changing it in the console will change the title on your app.





(a) Top: Web and Android clients in sync initially.
(b) Adding a pic.
(c) DetailView: I chose a caption of “new” and no URL so it gave me a random one and I edited the title to match.
(d) Edit/delete: I implemented the edit or delete option by re-using my add dialog and adding a delete “neutral” button and changing the title dynamically.

Hints

1. Notice that due to the detail fragment, this lab combines elements from HistoricalDocs and MovieQuotes. See how much of the fragment stuff you can do without copy-paste, to help internalize it in your mind.
2. I plan to have you extend this app in the next lab about Firebase Auth, so you'll do yourself a favor if you understand well what you are writing.
3. The ListFragment has a FAB, while the DetailFragment doesn't. You have a couple ways to handle this:
    - Have the MainActivity have the FAB, but set its visibility appropriately in each fragment: fab.setVisible(View.VISIBLE) or View.GONE. (Use GONE since you want to remove it from the layout altogether; View.INVISIBLE still makes room for it in the layout.) This is probably easier.
    - Move all the CoordinatorLayout stuff to the fragment_pic_list.xml so the FAB and Toolbar (which we don't use) are local to that fragment's layout. Then activity_min.xml is just the FrameLayout that is a container for the fragments.
4. Use interface callbacks when formalizing interactions between the fragment and the calling activity. But for simple things like when the fragment needs a resource, calling getContext() is fine to do.
5. Don't forget to add INTERNET permission in the manifest.
6. Don't forget the empty constructor in your model class.
7. It is painful to type long URLs in your phone. So I am providing you with a list of fixed URLs to use for testing. If the user doesn't enter a URL, just call this method to get a random one from this list. Feel free to change this list to your liking.

```
public class Util {
   public static String randomImageUrl() {
       String[] urls = new String[]{
               "https://upload.wikimedia.org/wikipedia/commons/1/1a/Dszpics1.jpg",
               "https://pixabay.com/static/uploads/photo/2015/09/14/15/52/lightning-939737_960_720.jpg",
               "http://s0.geograph.org.uk/geophotos/01/56/26/1562673_9088a4c9.jpg",

"https://upload.wikimedia.org/wikipedia/commons/6/61/Hurricane_Hugo_15_September_1989_1105z.png",
               "https://upload.wikimedia.org/wikipedia/commons/d/dc/P5med.jpg"
       };
       return urls[(int) (Math.random() * urls.length)];
   }
}
```

8. For the long-press to edit/remove, another option is to use a floating context menu, like we did in the Point of Sale app.
9. You are storing the app title in Firebase. First, you'll want to put it as a sibling to the list of pics. Second, you'll need to set up a listener to listen for changes: hint: it's the only setup function we haven't used (addValueEventListener) - but it works very similar to the addListenerForSingleValueEvent, which we did in FavoriteThings. Third, you can look up how to change the app title programmatically once you pull it from the dataSnapshot.

10. Refer back to ComicViewer lab and the video linked to it for a reminder on how to set up the AsyncTask to load images from a URL into an ImageView.
11. For the ImageView, sometimes getting the sizing/scaling right is hard. You can either fix the size of the image, allow scroll/zoom like ComicViewer, or use another sizing option. I found the SquareImageView suggested in this post to be helpful: http://stackoverflow.com/questions/16506275/imageview-be-a-square-with-dynamic-width

Other cool stuff you could do if you had time (all completely **optional,** but I will give **+2 bonus points** on this lab for each optional feature you do):

- Implement caching of images so refreshing isn't slow.
- Implement browsing for images so you don't have to enter URLs manually.

Show all of this to an instructor or assistant, via video or live demo.