

Photo Gallery

based on Chapter 24 of
Android Programming: A Big Nerd Ranch Guide (4th edition)

REST API and Retrofit

- **Please see the Retrofit App slides**
 - The information in this slide deck assumes that you have seen them

Activity and Fragment Layouts

- `activity_photo_gallery`
- `fragment_photo_gallery`

activity_photo_gallery

Simple frame layout for the Activity

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".PhotoGalleryActivity"/>
```

fragment_photo_gallery

Simple recycler view for the Fragment

```
<?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_width="match_parent"
    android:layout_height="match_parent"/>
```

Flickr API

- [flickr.com/services/api](https://www.flickr.com/services/api/)

Google 24. HTTP and Background Task Flickr Services

flickr.com/services/api/

Apps VT Canvas VT Web-CAT VT LinkedIn O'Reilly Piazza GitHub Zoom Medium Todoist Other Bookmarks

flickr You Explore Prints Get Pro

Photos, people, or groups

The App Garden

Create an App API Documentation Feeds What is the App Garden?

The Flickr API is available for non-commercial use by outside developers. Commercial use is possible by prior arrangement.

Read these first:

- Developer Guide
- Overview
- Encoding
- User Authentication
- Dates
- Tags
- URLs
- Buddyicons
- Flickr APIs Terms of Use
- API Keys**
- Developers mailing list

Photo Upload API

- Uploading Photos
- Replacing Photos
- Example Request
- Asynchronous Uploading

API Methods

activity

- flickr.activity.userComments
- flickr.activity.userPhotos

auth

- flickr.auth.checkToken
- flickr.auth.getFrob
- flickr.auth.getFullToken
- flickr.auth.getToken

auth.oauth

- flickr.auth.oauth.checkToken
- flickr.auth.oauth.getAccessToken

blogs

- flickr.blogs.getList
- flickr.blogs.getServices
- flickr.blogs.postPhoto

cameras

Sign in, or create a free account if you don't have one

Go here to get your key

Google

Networking Basics with Retrofi

Apps by Gregory_K. on Flickr

flickr.com/services/apps/by/23971394@N05

AppsVT CanvasVT Web-CATVT LinkedInO'ReillyPiazzaGitHubZoomMediumTodoistOther Bookmarks

flickr

YouExplorePrintsGet Pro

Photos, people, or groups

The App Garden

Gregory_K. / Gregory K
Apps By You | [Apps You're Using](#) | [Your Favorite Apps](#)

PhotoGallery
Key: [redacted]
Secret: [redacted]
0 authenticated users | 3 calls in the last 24 hours ([stats](#))
Private, Active

Gallery
Key: [redacted]
Secret: [redacted]
0 authenticated users | 0 calls in the last 24 hours ([stats](#))
Private, Active

DreamCatcher
Key: [redacted]
Secret: [redacted]
0 authenticated users | 0 calls in the last 24 hours ([stats](#))
Private, Active

Keep 'em coming!
Thanks for contributing to the [App Garden](#). If you have more apps in the works, we'd love to hear about them.
[Get Another Key](#) or [learn more](#).

Google | Network | services

← → flickr.com/services/api/

Apps VT Canvas VT LinkedIn Zoom Medium Todoist Other Bookmarks

Perl

- [Flickr::API](#)
- [Flickr::API2](#)
- [Flickr::Upload](#)

PHP

- [PEAR::Flickr_API](#)
- [phpFlickr](#)

PHP5

- [Phlickr](#)

Python

- [Beej's Python Flickr API](#)
- [flickr.py](#)
- [python-flickr-api](#)

REALbasic

- [REALflickr](#)

Ruby

- [flickraw](#)
- [flickr.rb](#)
- [rflickr](#)

See also: [Yahoo!'s Search APIs](#)

flickr.groups.discuss.replies.add

flickr.groups.discuss.replies.delete

flickr.groups.discuss.replies.edit

flickr.groups.discuss.replies.getInfo

flickr.groups.discuss.replies.getList

groups.discuss.topics

- [flickr.groups.discuss.topics.add](#)
- [flickr.groups.discuss.topics.getInfo](#)
- [flickr.groups.discuss.topics.getList](#)

groups.members

- [flickr.groups.members.getList](#)

groups.pools

- [flickr.groups.pools.add](#)
- [flickr.groups.pools.getContext](#)
- [flickr.groups.pools.getGroups](#)
- [flickr.groups.pools.getPhotos](#)
- [flickr.groups.pools.remove](#)

interestingness

- [flickr.interestingness.getList](#)

machinetags

- [flickr.machinetags.getNamespaces](#)
- [flickr.machinetags.getPairs](#)
- [flickr.machinetags.getPredicates](#)
- [flickr.machinetags.getRecentValues](#)
- [flickr.machinetags.getValues](#)

We're back on the original api page

We will get this list of photos

Google

Creating PhotoGallery

Flickr Services: Flickr API: flickr

flickr.com/services/api/flickr.interestingness.getList.html

AppsVT CanvasVT Web-CATVT LinkedInO'ReillyPiazzaGitHubZoomMediumTodoistOther Bookmarks

flickrYouExplorePrintsGet Pro

Photos, people, or groups

The App Garden

Create an AppAPI DocumentationFeedsWhat is the App Garden?

flickr.interestingness.getList

Returns the list of interesting photos for the most recent day or a user-specified date.

Authentication

This method does not require authentication.

Arguments

api_key (Required)
Your API application key. [See here](#) for more details.

date (Optional)
A specific date, formatted as YYYY-MM-DD, to return interesting photos for.

extras (Optional)
A comma-delimited list of extra information to fetch for each returned record. Currently supported fields are: description, license, date_upload, date_taken, owner_name, icon_server, original_format, last_update, geo, tags, machine_tags, o_dims, views, media, path_alias, url_sq, url_t, url_s, url_q, url_m, url_n, url_z, url_c, url_l, url_o

per_page (Optional)
Number of photos to return per page. If this argument is omitted, it defaults to 100. The maximum allowed value is 500.

page (Optional)
The page of results to return. If this argument is omitted, it defaults to 1.

A key is required

url_s will give us a URL to the JPG photo

Example Response

This method returns the standard XML response.

```
<photos page="2" pages="89" perpage="10" total="881">
  <photo id="2636" owner="47058503995@N01"
    secret="a123456" server="2" title="test_04"
    ispublic="1" isfriend="0" isfamily="0" />
</photos>
```

flickr

YouExplorePrintsGet Pro

Q

Photos, people, or groups

The App Garden

Create an AppAPI DocumentationFeedsWhat is the App Garden?

JSON Response Format

JSON, or JavaScript Object Notation, is a simple machine-readable data-interchange format, which makes constructing API applications in JavaScript easy (though it can be used from other languages too!). For more information about JSON, visit json.org.

To return an API response in JSON format, send a parameter "format" in the request with a value of "json".

Object Representation

Some simple rules are used when converting flickr REST XML into JSON objects. Some examples illustrate this best. A single tag will be translated to JSON as follows:

Failure Responses

Failure responses also call the `jsonFlickrApi()` method, but with a different JSON object. The object is not structured like the REST failure responses - instead it's simplified for JSON. For example:

```
jsonFlickrApi({
  "stat"      : "fail",
  "code"      : "97",
  "message"   : "Missing signature"
})
```

From JavaScript, you can check `rsp.stat` for failure, and then read the error from `rsp.code` and `rsp.message`.

Callback Function

If you just want the raw JSON, with no function wrapper, add the parameter `nojsoncallback` with a value of 1 to your request.

To define your own callback function name, add the parameter `jsoncallback` with your desired name as the value.

```
nojsoncallback=1      -> {...}
jsoncallback=wooYay   -> wooYay({...});
```

Examples

You can see a successful json response [here](#).

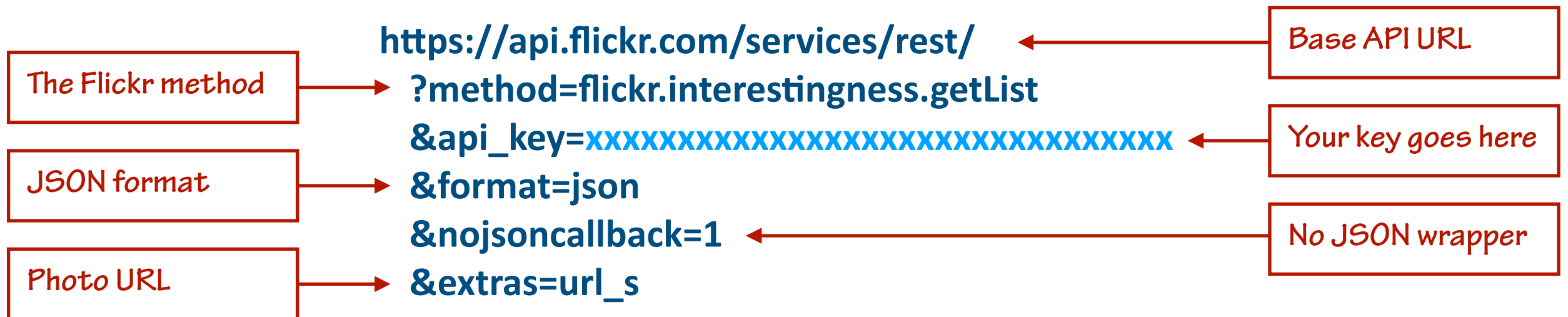
You can see a json failure response [here](#).

We will specify JSON format

We want raw JSON with no wrapper

The string to access
the JSON we want

This is all one line;
it's broken up for
clarity



We can type this in
the address bar and
see the JSON result

api.flickr.com/services/rest/?method= X

https://api.flickr.com/services/rest/?method=flickr.interestingness.getList&api

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

photos:

- page: 1
- pages: 5
- perpage: 100
- total: 500
- photo:
 - 0:
 - id: "49771750293"
 - owner: "12192905@N00"
 - secret: "535a56f382"
 - server: "65535"
 - farm: 66
 - title: "RIPPLES IN VARANASI..."
 - ispublic: 1
 - isfriend: 0
 - isfamily: 0
 - url_s: "https://live.staticflickr.com/65535/49771750293_535a56f382_m.jpg"
 - height_s: 159
 - width_s: 240
 - 1:
 - id: "49772340786"
 - owner: "66041906@N05"
 - secret: "1ac517168e"
 - server: "65535"
 - farm: 66
 - title: "By the Sea (Explored)"
 - ispublic: 1
 - isfriend: 0
 - isfamily: 0
 - url_s: "https://live.staticflickr.com/65535/49772340786_1ac517168e_m.jpg"
 - height_s: 160
 - width_s: 240
 - 2:
 - id: "49772241498"
 - owner: "15145314@N06"
 - secret: "0541d94590"
 - server: "65535"
 - farm: 66
 - title: "Pastel Serenity"
 - ispublic: 1
 - isfriend: 0
 - isfamily: 0
 - url_s: "https://live.staticflickr.com/65535/49772241498_0541d94590_m.jpg"
 - height_s: 153
 - width_s: 240

URL string from previous slide

JSON formatted nicely by Firefox

URLs to photos

Model and API Objects

- GalleryItem
- PhotoResponse
- FlickrResponse
- FlickrApi

▼ photos:
page: 1
pages: 5
perpage: 100
total: 500

This is our model class

GalleryItem

```
data class GalleryItem(  
    var title: String = "",  
    var id: String = "",  
    @SerializedName("url_s") var url: String = ""  
)
```

Property name in JSON

▼ photo:

▼ 0:

id: "49771750293"
owner: "12192905@N00"
secret: "535a56f382"
server: "65535"
farm: 66
title: "RIPPLES IN VARANASI..."
ispublic: 1
isfriend: 0
isfamily: 0
▼ url_s: "https://live.staticflickr.com/65535/49771750293_535a56f382_m.jpg"
height_s: 159
width_s: 240

▼ 1:

id: "49772340786"
owner: "66041906@N05"
secret: "1ac517168e"
server: "65535"
farm: 66
title: "By the Sea (Explored)"
ispublic: 1
isfriend: 0
isfamily: 0
▼ url_s: "https://live.staticflickr.com/65535/49772340786_1ac517168e_m.jpg"
height_s: 160
width_s: 240

```

▼ photos:
  page: 1
  pages: 5
  perpage: 100
  total: 500
  ▼ photo:

```

Main response type.
Main **request** is type:
`Call<FlickrResponse>`

FlickrResponse

```

class FlickrResponse {
    lateinit var photos: PhotoResponse
}

```

```

  id: "49771750293"
  owner: "12192905@N00"
  secret: "535a56f382"
  server: "65535"
  farm: 66
  title: "RIPPLES IN VARANASI..."
  ispublic: 1
  isfriend: 0
  isfamily: 0
  ▼ url_s: "https://live.staticflickr.com/65535/49771750293_535a56f382_m.jpg"
  height_s: 159
  width_s: 240
  ▼ 1:
    id: "49772340786"
    owner: "66041906@N05"
    secret: "1ac517168e"
    server: "65535"
    farm: 66
    title: "By the Sea (Explored)"
    ispublic: 1
    isfriend: 0
    isfamily: 0
    ▼ url_s: "https://live.staticflickr.com/65535/49772340786_1ac517168e_m.jpg"
    height_s: 160
    width_s: 240

```

PhotoResponse

```

class PhotoResponse {
    @SerializedName("photo")
    lateinit var galleryItems: List<GalleryItem>
}

```

Retrofit translates each
item in JSON photo
array into a *GalleryItem*

FlickrApi

```
interface FlickrApi {  
    @GET(  
        "services/rest/?method=flickr.interestingness.getList" +  
        "&api_key=12345678901234567890123456789012" +  
        "&format=json" +  
        "&nojsoncallback=1" +  
        "&extras=url_s"  
    )  
    fun fetchPhotos(): Call<FlickrResponse>  
}
```

Request string
sent to Flickr API

Your key
goes here

Returns the main **request**

FlickrFetchr

- FlickrFetcher


FlickrFetchr

```
class FlickrFetchr {  
  
    private val flickrApi: FlickrApi  
  
    init {  
        val retrofit: Retrofit = Retrofit.Builder()  
            .baseUrl("https://api.flickr.com/")  
            .addConverterFactory(GsonConverterFactory.create())  
            .build()  
  
        flickrApi = retrofit.create(FlickrApi::class.java)  
    }  
  
    fun fetchPhotos(): LiveData<List<GalleryItem>> { ... }  
}
```

Retrofit
implementation of
FlickrApi interface



Calls through to FlickrApi
fetchPhotos, but wraps the
response in a LiveData object,
which is observed by the
Fragment



FlickrFetchr ⇨ fetchPhotos

```
fun fetchPhotos(): LiveData<List<GalleryItem>> {  
    val responseLiveData: MutableLiveData<List<GalleryItem>> = MutableLiveData()  
    val flickrRequest: Call<FlickrResponse> = flickrApi.fetchPhotos()  
  
    flickrRequest.enqueue(object : Callback<FlickrResponse> {  
  
        override fun onFailure(call: Call<FlickrResponse>, t: Throwable) {  
            Log.e(TAG, "Failed to fetch photos", t)  
        }  
  
        override fun onResponse(  
            call: Call<FlickrResponse>,  
            response: Response<FlickrResponse>  
        ) {  
            Log.d(TAG, "Response received")  
            val flickrResponse: FlickrResponse? = response.body()  
            val photoResponse: PhotoResponse? = flickrResponse?.photos  
            var galleryItems: List<GalleryItem> = photoResponse?.galleryItems  
            ?: mutableListOf()  
            galleryItems = galleryItems.filterNot {  
                it.url.isBlank()  
            }  
            responseLiveData.value = galleryItems  
        }  
    })  
  
    return responseLiveData  
}
```

Asynchronous call
on separate thread

Main response

Main response
photos

Main response
photo items

If list is null,
return empty list

Filter out photos with
no valid url_s property

Observers are notified

PhotoGallery

- **PhotoGalleryActivity**
- **PhotoGalleryFragment**
- **PhotoGalleryViewModel**

The Activity holds
the Fragment

Photo Gallery
Activity

Photo Gallery
Fragment

The Fragment observes
LiveData changes in the
ViewModel

The ViewModel calls
the repository

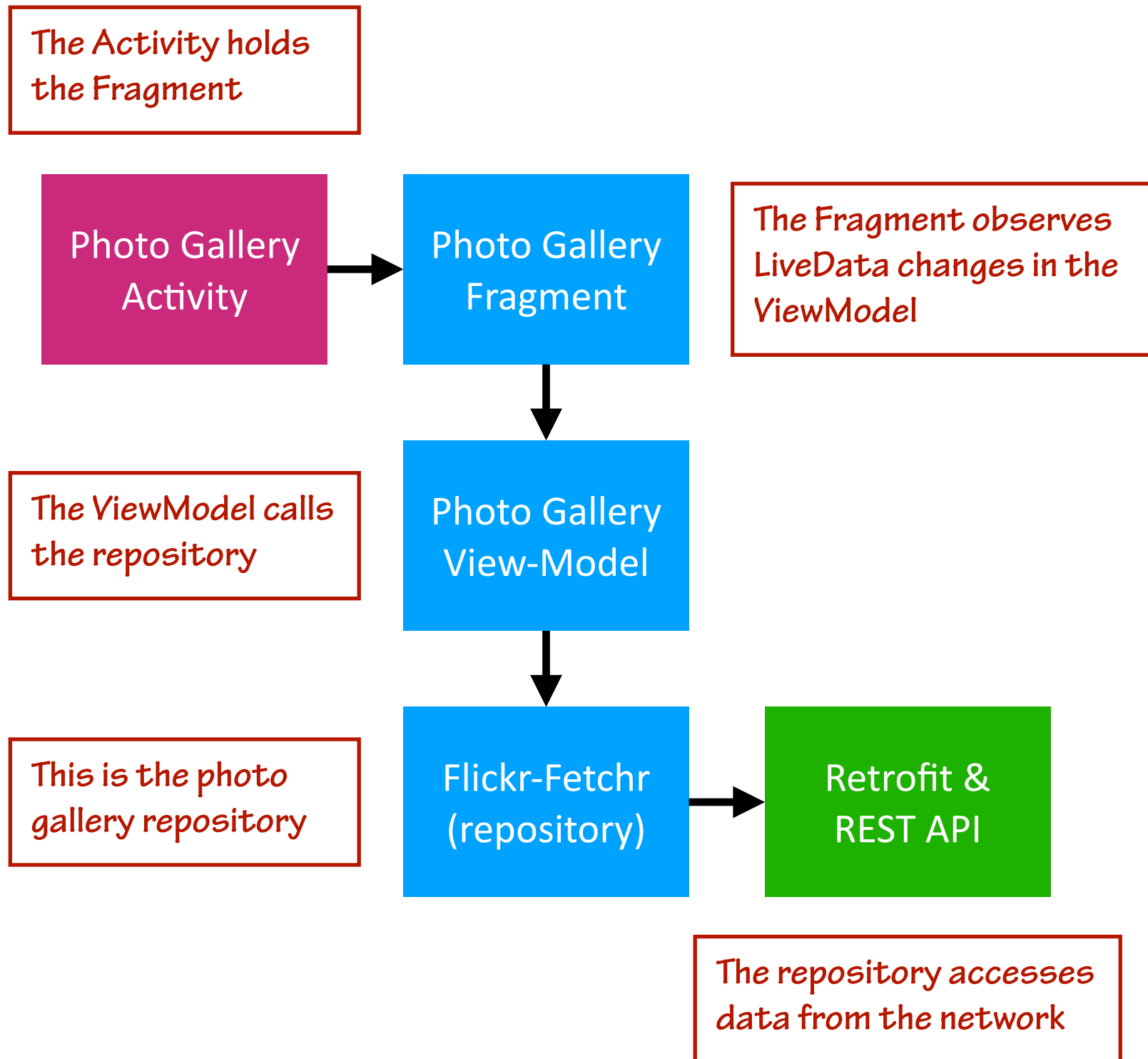
Photo Gallery
View-Model

This is the photo
gallery repository

Flickr-Fetchr
(repository)

Retrofit &
REST API

The repository accesses
data from the network



PhotoGalleryViewModel

```
class PhotoGalleryViewModel : ViewModel() {  
    val galleryItemLiveData = FlickrFetchr().fetchPhotos()  
}
```

*Simple call-through obtains LiveData
from the repository (FlickrFetchr)*

```

class PhotoGalleryFragment : Fragment() {
    private var _binding: FragmentPhotoGalleryBinding? = null
    private val binding get() = _binding!!

    private val viewModel: PhotoGalleryViewModel by viewModels()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentPhotoGalleryBinding.inflate(inflater, container, false)
        val view = binding.root
        binding.photoRecyclerView.layoutManager = GridLayoutManager(context, 3)
        return view
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        viewModel.galleryItemLiveData.observe(
            viewLifecycleOwner
        ) { galleryItems ->
            binding.photoRecyclerView.adapter = PhotoAdapter(galleryItems)
        }
    }
}

```

Declare binding for
fragment

Declare and init
view-model (KTX)

Init binding and
use grid layout for
recycler

Observe LiveData
from ViewModel

Use gallery items
in rv-adapter

PhotoGalleryFragment ⇔ Holder / Adapter / companion object

Lambda expression
syntax

```
private class PhotoHolder(itemTextView: TextView)
    : RecyclerView.ViewHolder(itemTextView) {
    val bindTitle: (CharSequence) -> Unit = itemTextView::setText
}
private class PhotoAdapter(private val galleryItems: List<GalleryItem>)
    : RecyclerView.Adapter<PhotoHolder>() {
    override fun onCreateViewHolder(
        parent: ViewGroup,
        viewType: Int
    ): PhotoHolder {
        val textView = TextView(parent.context)
        return PhotoHolder(textView)
    }
    override fun getItemCount(): Int = galleryItems.size
    override fun onBindViewHolder(holder: PhotoHolder, position: Int) {
        val galleryItem = galleryItems[position]
        holder.bindTitle(galleryItem.title)
    }
}
companion object {
    fun newInstance() = PhotoGalleryFragment()
}
}
```

newInstance used by
main Activity

PhotoGalleryActivity

```
class PhotoGalleryActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_photo_gallery)  
  
        if (savedInstanceState == null) {  
            supportFragmentManager  
                .beginTransaction()  
                .add(R.id.fragment_container, PhotoGalleryFragment.newInstance())  
                .commit()  
        }  
    }  
}
```

Different approach than `CriminalIntent` to checking if a fragment is already hosted: checks to see if bundle is null.

If null => this is a fresh launch of the activity

If not null => activity is being reconstructed after a system-initiated destruction (such as rotation or process death)