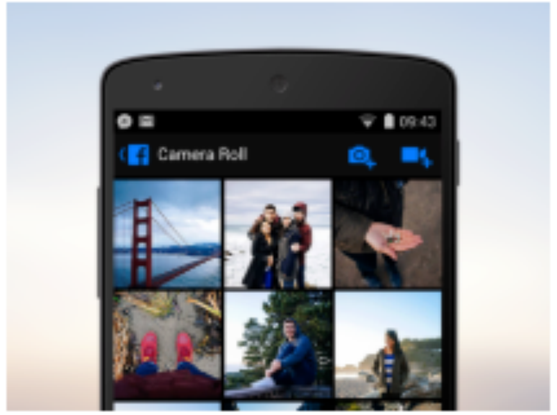
Images in Android

Images



Support selecting image from phone gallery, and then displaying them in an activity.

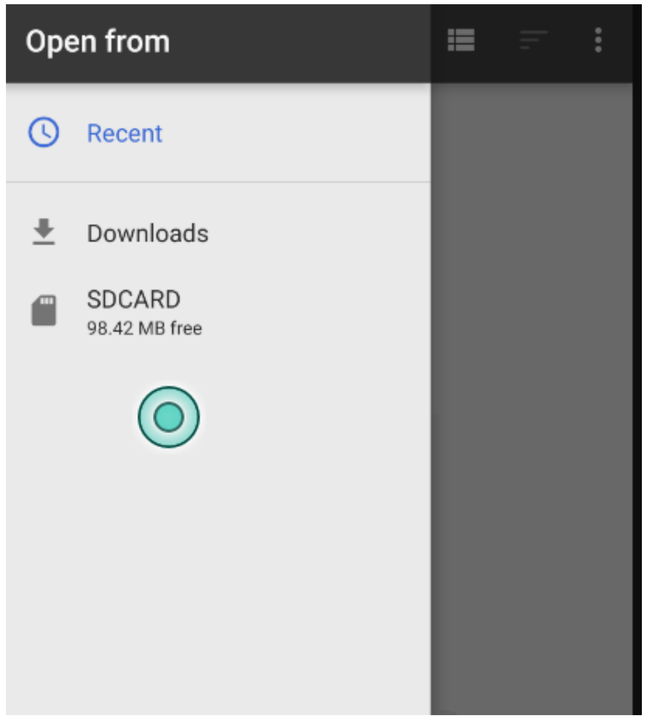
Open Files using Storage Access Framework

Android 4.4 (API level 19) introduces the Storage Access Framework (SAF). The SAF makes it simple for users to browse and open documents, images, and other files across all of their their preferred document storage providers. A standard, easy-to-use UI lets users browse files and access recents in a consistent way across apps and providers.

https://developer.android.com/guide/topics/providers/document-provider.html

SAF enable:

- browse content from all document providers, not just a single app.
- Makes it possible for your app to have long term, persistent access to documents owned by a document provider. Through this access users can add, edit, save, and delete files on the provider.
- Supports multiple user accounts and transient roots such as USB storage providers, which only appear if the drive is plugged in.



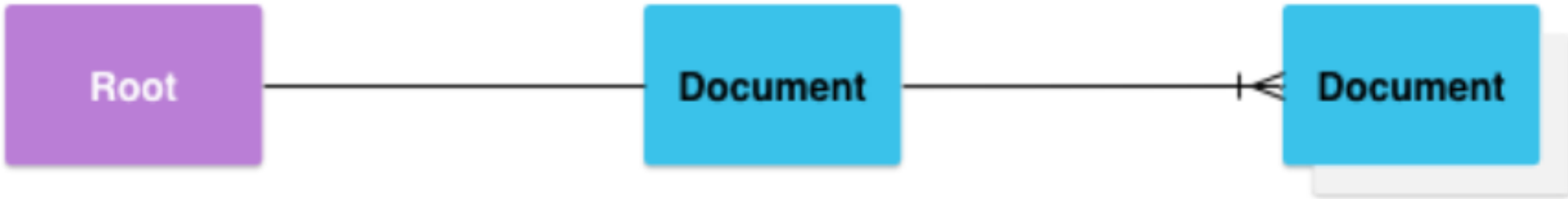
The SAF includes the following:

- **Document provider**—A content provider that allows a storage service (such as Google Drive) to reveal the files it manages. A document provider is implemented as a subclass of the `DocumentsProvider` class. The document-provider schema is based on a traditional file hierarchy, though how your document provider physically stores data is up to you. The Android platform includes several built-in document providers, such as Downloads, Images, and Videos.

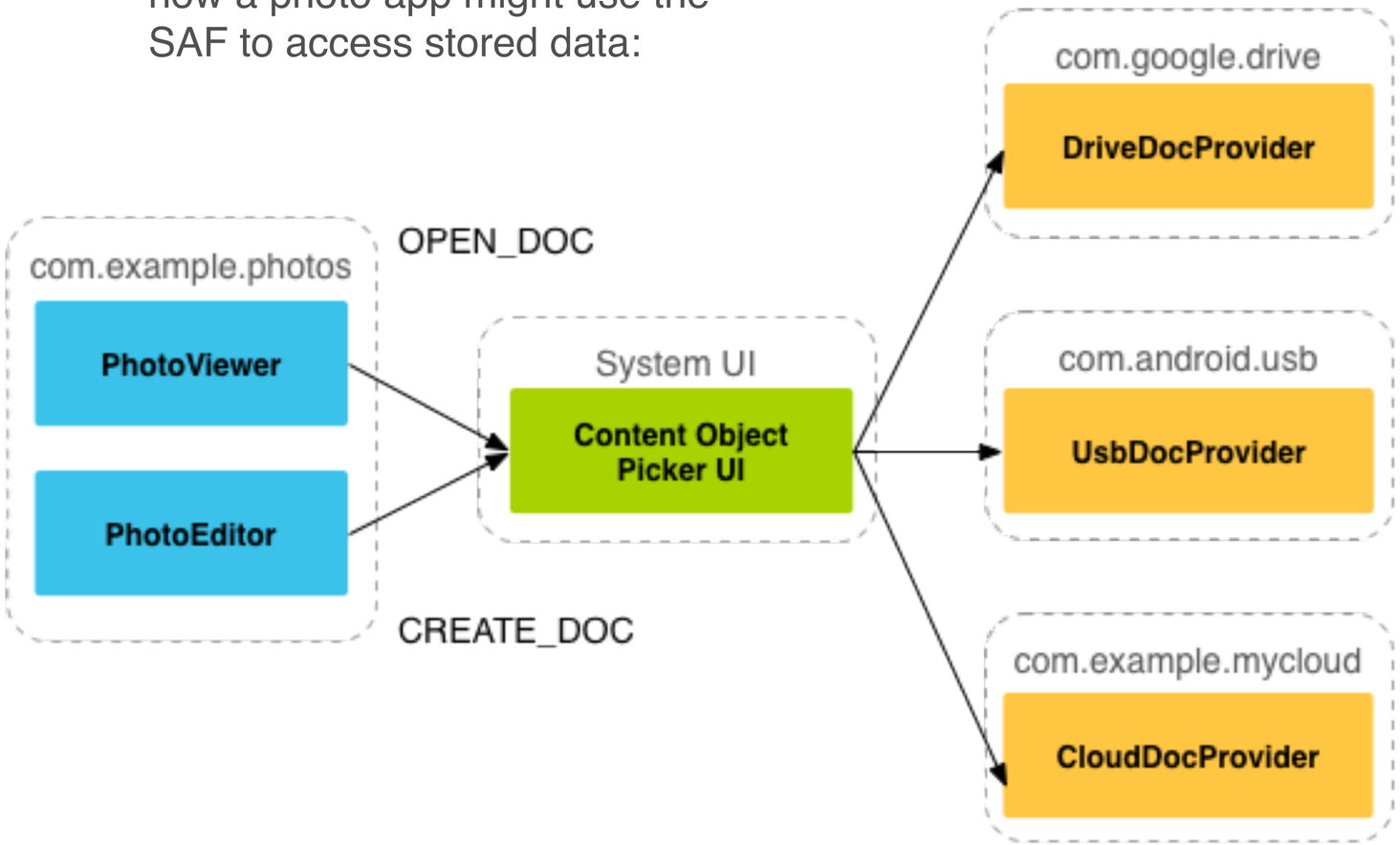
- **Client app**—A custom app that invokes the `ACTION_OPEN_DOCUMENT` and/or `ACTION_CREATE_DOCUMENT` intent and receives the files returned by document providers.

- **Picker**—A system UI that lets users access documents from all document providers that satisfy the client app's search criteria.

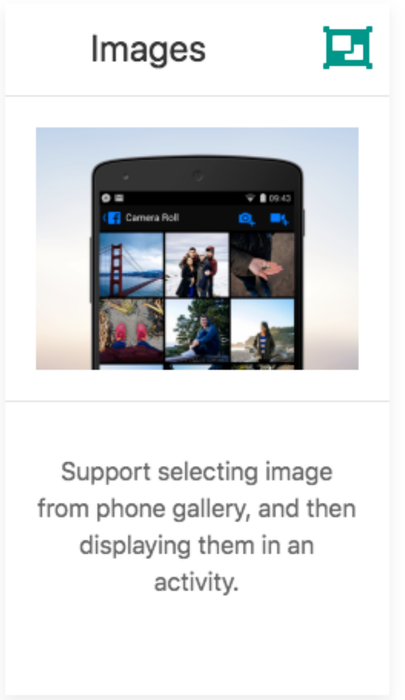
The SAF centers around a content provider that is a subclass of the DocumentsProvider class. Within a document provider, data is structured as a traditional file hierarchy:



how a photo app might use the SAF to access stored data:



Images in Placemark



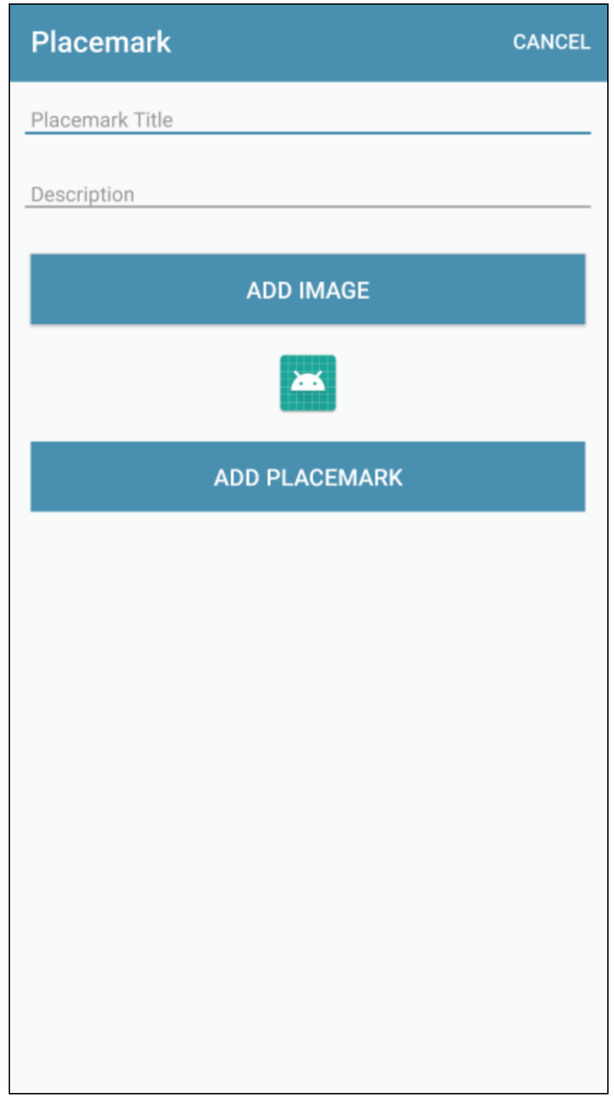
1. Include ImageViewer in Layout
2. Incorporate Helper function: showImagePicker()
3. Introduce event handler for ‘Add Image’ button + show picker
4. Return image file name to PlacemarkActivity
5. Extend PlacemarkModel to store image name
6. Incorporate additional helper to read image from picker
7. In PlacemarkActivity - display the selected image in the layout
8. Incorporate additional helper to load image given its file name
9. When PlacemarkActivity loaded for existing model - load image from image path

1. Include ImageView in Layout

```
<string name="button_addImage"> Add Image </string>
```

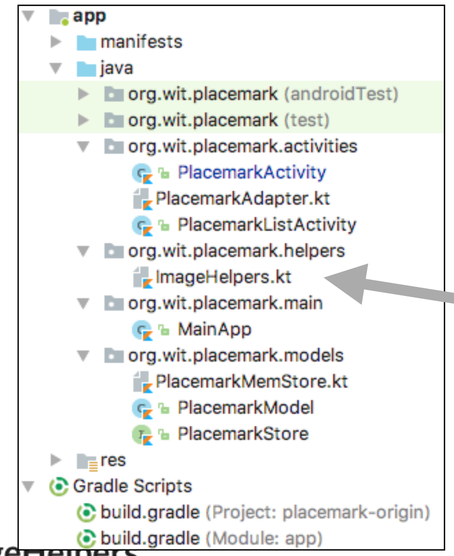
```
<Button
    android:id="@+id/chooseImage"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:background="@color/colorAccent"
    android:paddingBottom="8dp"
    android:paddingTop="8dp"
    android:text="@string/button_addImage"
    android:textColor="@color/colorPrimary"
    android:textSize="16sp"/>

<ImageView
    android:id="@+id/placemarkImage"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:srcCompat="@mipmap/ic_launcher"/>
```

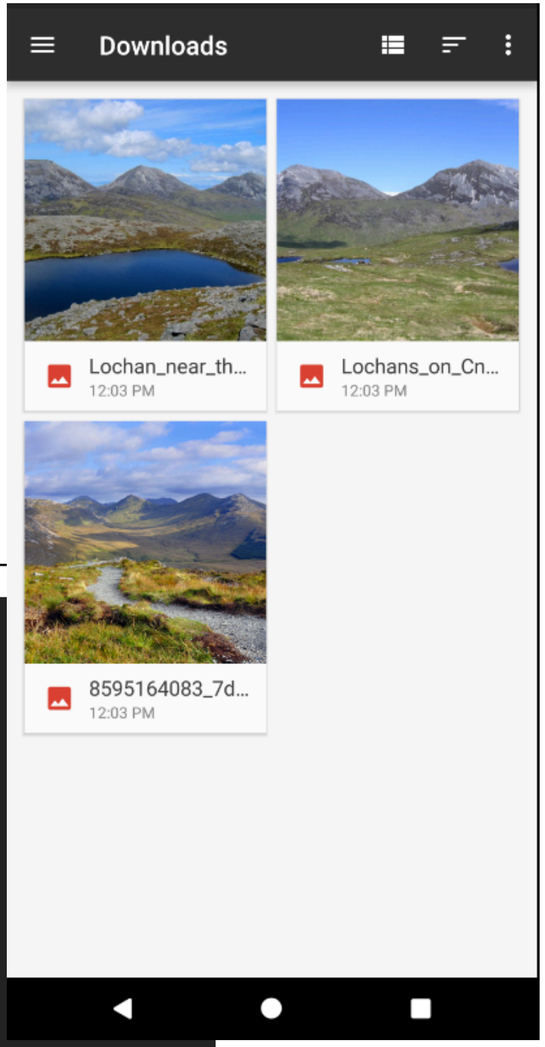


PlacemarkActivity

```
    chooseImage.setOnClickListener {
      info ("Select image")
    }
```



2. Incorporate Helper function: showImagePicker()



ImageHelpers

```
package org.wit.placemark.helpers

import android.app.Activity
import android.content.Intent
import org.wit.placemark.R

fun showImagePicker(parent: Activity, id: Int) {
  val intent = Intent()
  intent.type = "image/*"
  intent.action = Intent.ACTION_OPEN_DOCUMENT
  intent.addCategory(Intent.CATEGORY_OPENABLE)
  val chooser = Intent.createChooser(intent, R.string.select_placemark_image.toString())
  parent.startActivityForResult(chooser, id)
}
```

strings.xml

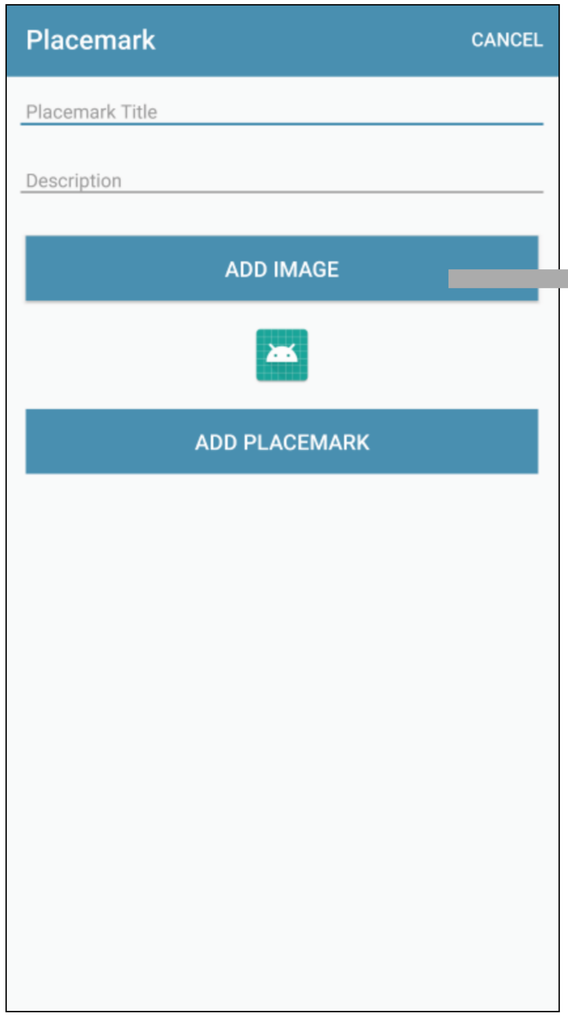
```
<string name="select_placemark_image">Select placemark image</string>
```

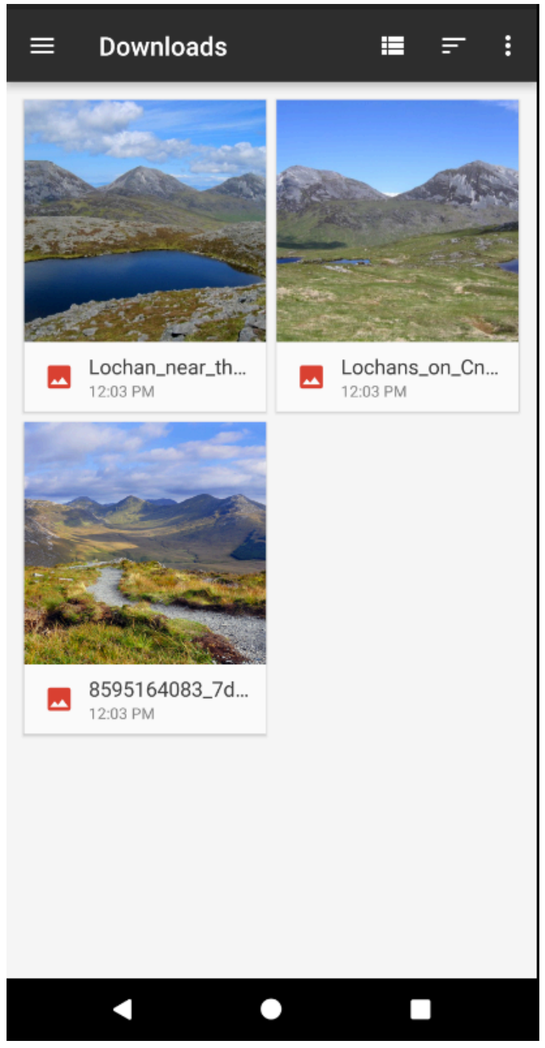
PlacemarkActivity

3. Introduce event handler for ‘Add Image’ button + show picker

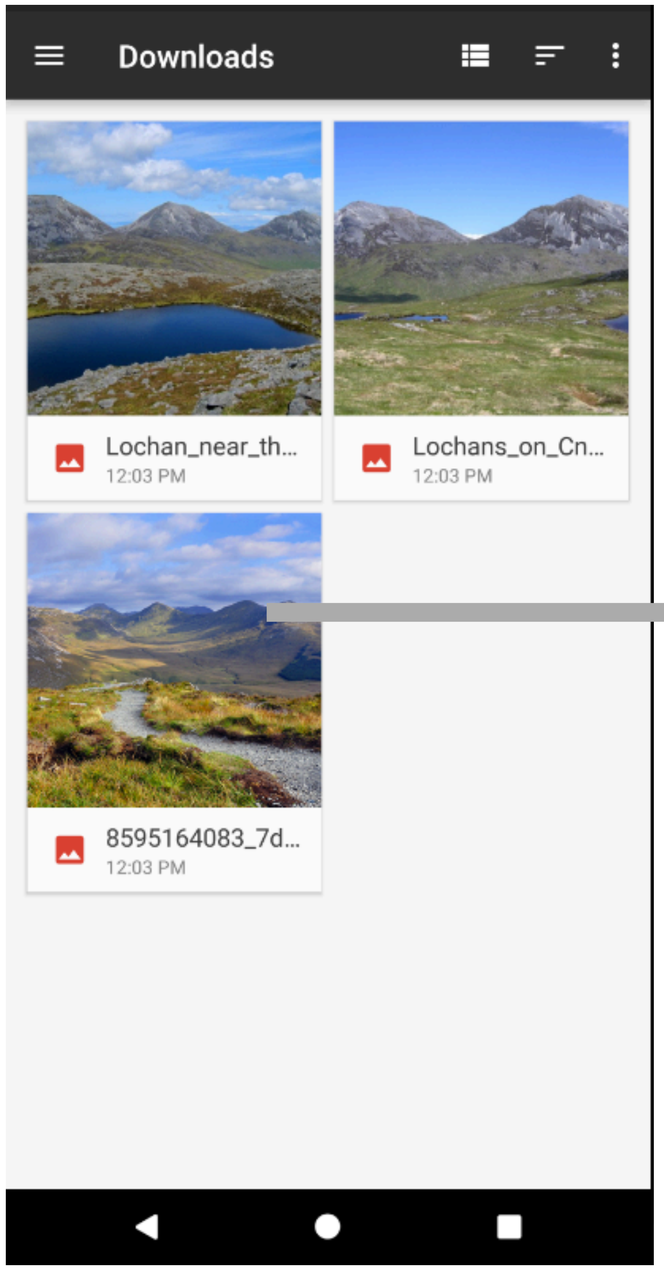
```
val IMAGE_REQUEST = 1
```

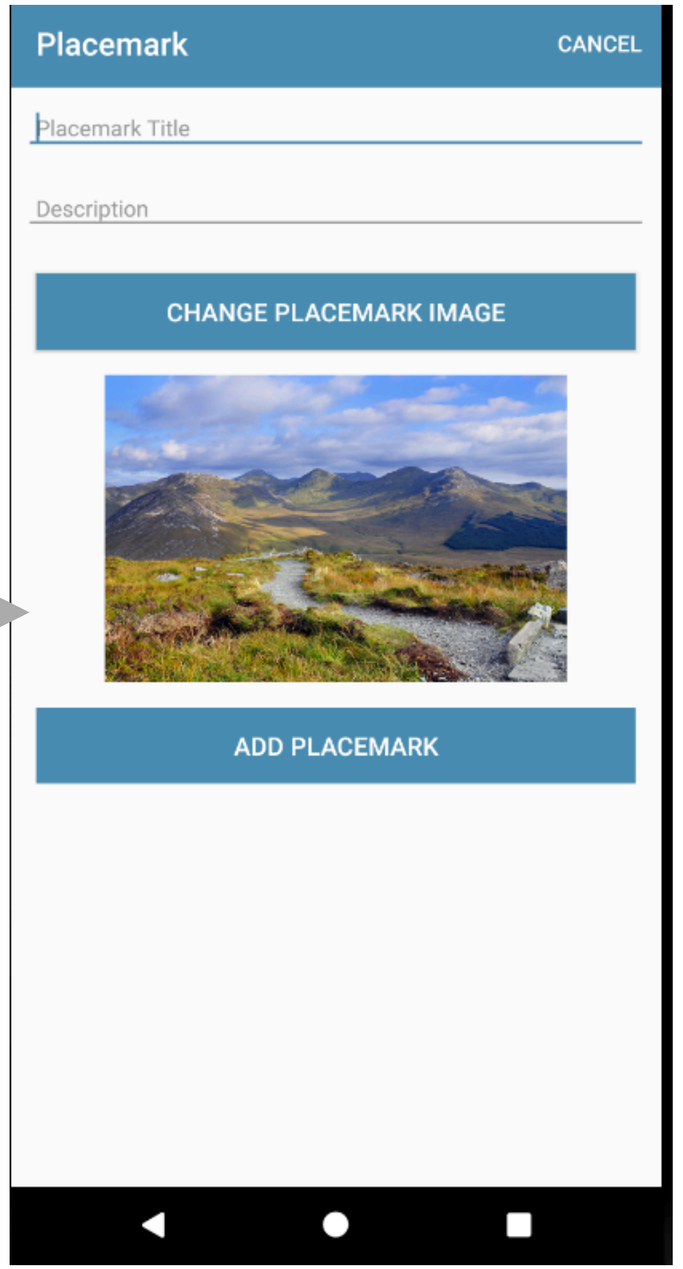
```
chooseImage.setOnClickListener {
  showImagePicker(this, IMAGE_REQUEST)
}
```





4. Return image file name to PlacemarkActivity





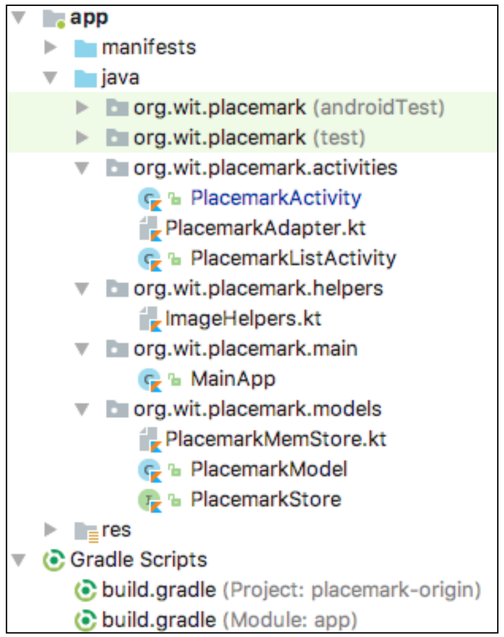
4.Return image file name to PlacemarkActivity

PlacemarkActivity

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
  super.onActivityResult(requestCode, resultCode, data)
  when (requestCode) {
    IMAGE_REQUEST -> {
      if (data != null) {
        placemark.image = data.getData().toString()
      }
    }
  }
}
```

5. Extend PlacemarkModel to store image name

```
@Parcelize
data class PlacemarkModel(var id: Long = 0,
                          var title: String = "",
                          var description: String = "",
                          var image: String = "") : Parcelable
```



6. Incorporate additional helper to read image from picker

Imagehelpers

```
fun readImage(activity: Activity, resultCode: Int, data: Intent?): Bitmap? {
  var bitmap: Bitmap? = null
  if (resultCode == Activity.RESULT_OK && data != null && data.data != null) {
    try {
      bitmap = MediaStore.Images.Media.getBitmap(activity.contentResolver, data.data)
    } catch (e: IOException) {
      e.printStackTrace()
    }
  }
  return bitmap
}
```

7. In PlacemarkActivity - display the selected image in the layout

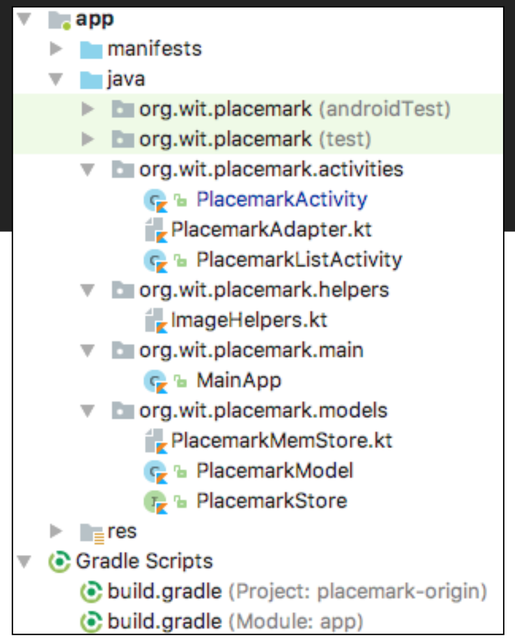
```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
  super.onActivityResult(requestCode, resultCode, data)
  when (requestCode) {
    IMAGE_REQUEST -> {
      if (data != null) {
        placemark.image = data.getData().toString()
        placemarkImage.setImageBitmap(readImage(this, resultCode, data))
        chooseImage.setText(R.string.change_placemark_image)
      }
    }
  }
}
```

```
<ImageView
    android:id="@+id/placemarkImage"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:srcCompat="@mipmap/ic_launcher"/>
```

8. Incorporate additional helper to load image given its file name

ImageHelpers

```
fun readImageFromPath(context: Context, path : String) : Bitmap? {
  var bitmap : Bitmap? = null
  val uri = Uri.parse(path)
  if (uri != null) {
    try {
      val parcelFileDescriptor = context.getContentResolver().openFileDescriptor(uri, "r")
      val fileDescriptor = parcelFileDescriptor.getFileDescriptor()
      bitmap = BitmapFactory.decodeFileDescriptor(fileDescriptor)
      parcelFileDescriptor.close()
    } catch (e: Exception) {
    }
  }
  return bitmap
}
```



9. When PlacemarkActivity loaded for existing model - load image from image path

PlacemarkActivity

```
if (intent.hasExtra("placemark_edit")) {
  //... as before
  placemarkImage.setImageBitmap(readImageFromPath(this, placemark.image))
}
```