Gallery intent android example

This lesson teaches you how to capture a photo by delegating work to another camera app on your device. (If you prefer to create your own camera functionality, see Camera Control.) Suppose you're implementing a crowdsourced weather service that makes a global weather map by mixing together photos of the sky taken by devices running your client app. Photo integration is only a small part of the app. You want to take photos with minimal fuss, not reinvent the camera. Fortunately, most Android devices already have at least one camera app installed. In this lesson, you will learn how to make this take a picture for you. Request camera function If an important feature of your app is shooting, then limit its visibility on Google Play on devices that have a camera. To advertise that your app depends on having a camera, place the tag in the manifest file: qlt;manifest ...-gt; zlt;feature android:uses.android.hardware.camera android: required If your app uses but doesn't require a camera to function, instead install android: you need a false one. At the same time, Google Play will allow devices without a camera to download the app. It is then your responsibility to check the presence of the camera while running on the phone hasSystemFeature (PackageManager.FEATURE_CAMERA_ANY). If the camera isn't available, you should turn off the camera. Take a photo with the camera app The way you delegate action to other apps is to trigger the intention by describing what you want to do. This process involves three parts: the intention itself, the call to start external activity, and some code for processing image data when the focus returns to your activity. Here is a feature that triggers the intention to capture a photo. val REQUEST_IMAGE_CAPTURE - 1 private fun dispatchtakeTakePictureIntent () - val takePictureIntent - Intention (MediaStore.ACTION_IMAGE_CAPTURE) attempt - startActivityForResult (takePictureIntent, REQUEST_IMAGE_CAPTURE) REQUEST_IMAGE_CAPTURE - catch (e: ActivityNotFoundException) private void dispatchingtakeTakeTakePictent () - Intention takePicIntureIn MediaStore.ACTION_IMAGE_CAPTURE ture Try to startActivityForResult (takePictureIntent, REQUEST_IMAGE_CAPTURE); Catch (ActivityNotFoundException e) // State of the display error to the user - If you don't want to show in the app that the camera is unavailable, the other option is to add the following to your manifest: qlt'zgt; qlt;action android:android.media.action.IMAGE_CAPTURE/action Your app's ambitions, then you probably want to get the image back out of the camera app and do something with it. Android The app encodes the photo in Return Intention delivered to OnActivityResult () as a small Bitmap in extras, under key data. The next code gets this image and displays it in ImageView. redefine pleasure onActivityResult (requestCode: Int, resultCode: Int, Data: Intention?) - if (requestCode - REQUEST_IMAGE_CAPTURE - resultCode - RESULT_OK) - val imageBitmap - data.extras.get (data) as Bitmap imageView.setImimageBit @Override map int resultCode, Intent Data) - if (requestCode - REQUEST_IMAGE_CAPTURE - resultCode - RESULT_OK) - Supplemental Services and data.getExtras (); Bitmap imageBitmap (Bitmap) extras.get imageView.setImageBitmap (imageBitmap); Note: This sketch image from the data may be good for the icon, but not much more. Working with a full-size image takes a little more work. Save the full photo Android Camera app saves full-size photos if you give it a file to save in. You have to provide a fully qualified file name where the camera app should save the photo. Typically, any photos that a user captures with the device's camera must be stored on the device in a public external store so that they are available to all apps. The proper catalog for shared photos is provided by getExternalStoragePublicDirectory, with DIRECTORY_PICTURES argument. The catalog provided by this method is common to all applications. On Android 9 (API level 28) and below, reading and writing in this catalog requires READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE permissions, respectively: zlt;manifest android.permission.WRITE_EXTERNAL_STORAGE android.permission.READ_EXTERNAL_STORAGE ... On Android 10 (API level 29) and above, the correct catalog for photo sharing is the mediaStore.Images table. You don't need to announce any storage permissions if your app only needs access to photos taken by the user using the app. However, if you want your photos to remain private only for your app, you can use the catalog provided by getExternalFilesDir. On Android 4.3 and below, writing this catalog also requires WRITE_EXTERNAL_STORAGE permission. Starting with Android 4.4, resolution is no longer required because the catalog is not available to other apps, so you can announce that permission should only be requested on the lower versions of Android, adding the attribute maxsdkVersion: qlt;manifest android.permission.WRITE_EXTERNAL_STORAGE ... ... Note: The files you save in the directories provided by getExternalFilesDir () or getFilesDir are deleted when the user deletes your app. Once you decide to catalog for the file, you need to create a create file name. You might also want to save the path in the member variable for longer use. Here's an example of a solution in a method that returns a unique file name for a new photo using a date of time stamp: lateinit var currentPhotoPath: String @Throws (IOException::class) private fun createImageFile (): File / File : Create the name of the image file of the image of timeStamp: String and SimpleDateFormat (yyyyMMdd_HHmmss). ExternalFilesDir (Environment.DIRECTORY_PICTURES) return File.createTempFile (JPEG_ JPEG_ $timeStamp) / prefix.jpg/.jpg, // Suffix th/storageDir /) catalog No / // Save file: way to use with ACTION_VIEW intentions currentPhotoPath private file createImageFile yyyyMMdd_HHmmss () casts IOException // Create the title of the image of the image of the time String imageFileName - JPEG_ - timeStamp - I; Dir file storage - getExternalFilesDir (Environment.DIRECTORY_PICTURES); File image - File.createTempFile (imageFileName, /) prefix No/.jpg // Suffix Save file: way to use with ACTION_VIEW intentions currentPhotoPath and image.getAbsolutePath (); Return the image With this method available to create a file for a photo, you can now create and trigger an intention like this: val REQUEST_TAKE_PHOTO and 1 private fun dispatchTakePictureIntent () - Intention (MediaStore.ACTION_IMAGE_CAPTURE qgt;)... Also, / Create a file where the photo should go val photoFile: File? null / Continue only if the file has been successfully created photoFile?. in addition, val photoURI: Uri - FileProvider.getUriForFile (this, com.example.android.android.fileprovider, it) takePictureIntent.putExtra (MediaStore.EXTRA_OUTPUT, photoURI) startActivityForResult (takePictureIntent, MediaStore.ACTION_IMAGE_CAPTURE REQUEST_TAKE_PHOTO REQUEST_TAKE_PHOTO) private void dispatch control roomTakeTakeTakepictentInture Make sure there is an activity camera to handle the intention if (takePictureIntent.resolveActivity (getPackageManager)) ! try photoPhile and createImageFile (); catch (IOException ex) / A mistake occurred when creating a file... / Continue only in the event if the file was successfully created, if (photoFile !) - Uri photoURI and FileProvider.getUriForFile (it, com.android.android.fileprovider) takePicIntent.putExtra (MediaStore.EXTRA_OUTPUT, photoURI); startActivityForultRes (takePictureIn REQUEST_TAKE_PHOTO tent) content:// URI. For later apps focused on Android 7.0 (API level 24) and above, the passage of the URI file:// across the package boundary is caused by FileUriExposedException. So now we're introducing a more general way of storing images with FileProvider. Now you need to set up FileProvider. In the app's manifesto, add a vendor to the app: ... <provider android:name=androidx.core.content.FileProvider android:authorities=com.example.android.fileprovider android:exported=false android:granturipermissions=true> <meta-data android:name=android.support.FILE_PROVIDER_PATHS android:resource=@xml/file_paths></meta-data> </provider> ... Make sure the power line matches the second argument to getUriForFile (Context, Line, File). In the vendor definition metadata section, you can see that the vendor expects the appropriate paths to be configured in the dedicated res/xml/file_paths.xml resource file. Here's the content required for this particular example: ?xml version?1.0 encoding?utf-8?'lt; my_images path/data/data.example.issue.name/files/photos/external-files-path;the path component corresponds to the path that getExternalFilesDir returns when Environment.DIRECTORY_PICTURES. Make sure you com.example.package.name on the actual name of your app's package. Also, check fileProvider documentation for an extensive description of the path listed that you can use other than the outside path. Add a photo to the gallery When you create a photo through intent, you need to know where your image is because you said where to save it first. For everyone else, perhaps the easiest way to make your photo available is to make it available from the media system provider. Note: If you have stored your photo in a catalog provided by getExternalFilesDir, the media scanner cannot access the files because they are closed to your app. The following example demonstrates how to call a media scanner system to add a photo to a media provider's database, make it available in the Android Gallery app and other apps. Private Fun GalleryAddPic (Intent.ACTION_MEDIA_SCANNER_SCAN_FILE) - Intention (Intent.ACTION_MEDIA_SCANNER_SCAN_FILE zgt;)... File f - new file (currentPhotoPath); Uri contentUri and Uri.fromFile (f); mediaScanIntent.setData (contentUri); this.sendBroadcast (mediaScanIntent); Decoding a scalable image, managing multiple full-size images, can be difficult with limited memory. If you find that your app is running out of memoryBy displaying just a few images, you can significantly reduce the amount of dynamic heap used by expanding JPEG into a memory array that is already scaling to match the size of the destination view. This method demonstrates the following example. Private Fun SetPic ( ) / Get the dimensions of View val targetW: Int - imageView.width val targetH: Int - imageView.height val bmOptions - BitmapFactory.Options .). Apply / Get bit card sizes inJustDecodeBounds - true BitmapFactory.decodeFile (currentPhotoPath). bmOptions) val photoW: Int - outWidth val photoH: Int - outHeight / Identify, how much you need to shorten the image of val scaleFactor: Int - Math.max (1, Math.min (photoW/targetW, photoH/targetH)) // Decode the image file in Bitmap size to fill the view inJustDecodeBounds - false inSampleSize - scaleFactor inPurgeable - true - BitmapFactory.deFilcode also - bitmap - imageView.setImageBitmap (bitmap) - private void setPic () / Get the dimensions of the target kind intW - imageView.getWidth (); int targetH - imageView.getHeight(); Get bitmapFactory.Options bmOptions - new BitmapFactory.Options bmOptions.inJustDecodeBounds - truth; BitmapFactory.deCodeFile (currentPhotoPath, bmOptions); int photoW - bmOptions.outWidth; int photoH and bmOptions.outHeight; Determine how much to scale int scaleFactor and Math.max (1, Math.min (photoW/targetW, photoH/targetH)); Decode the image file in Bitmap the size of View bmOptions.inJustDecodeBounds is false; bmOptions.inSampleSize - scaleFactor; bmOptions.inPurgeable - truth; Bitmap bitmap - BitmapFactory.decodeFile imageView.setImageBitmap (bit map); } } camera and gallery intent android example