MOBILE PROGRAMING

Midterm project – Firebase Image Storage

Date | time 15/4/2024

# Members

Lâm Nguyễn Huy Hoàng – 21110028

Nguyễn Đăng Khoa – 21110045

# Table of Contents

[Members 1](#_Toc164039883)

[Table of Contents 1](#_Toc164039884)

[Project Description 2](#_Toc164039885)

[Project target 2](#_Toc164039886)

[Uploading Mechanic 2](#_Toc164039887)

[1. Dependencies: 2](#_Toc164039888)

[2. Activity Setup: 2](#_Toc164039889)

[3. File Chooser: 4](#_Toc164039890)

[4. Image Selection: 5](#_Toc164039891)

[5. Image Upload: 5](#_Toc164039892)

[6. Upload Success/Failure: 5](#_Toc164039893)

[7. View Images: 6](#_Toc164039894)

[8. Firebase Integration: 6](#_Toc164039895)

[View Images mechanic 6](#_Toc164039896)

[1. Activity Setup: 6](#_Toc164039897)

[2. Loading Images: 8](#_Toc164039898)

[3. ImageAdapter: 8](#_Toc164039899)

[4. Placeholder: 9](#_Toc164039900)

[5. Image Loading with Glide: 9](#_Toc164039901)

[6. Full Screen Image: 9](#_Toc164039902)

[Full Screen Viewer 10](#_Toc164039903)

[1. Activity Setup: 10](#_Toc164039904)

[2. Image Loading: 11](#_Toc164039905)

[3. Click Listener: 11](#_Toc164039906)

[4. Download Popup: 11](#_Toc164039907)

[5. Download Button: 11](#_Toc164039908)

[6. Download Service: 12](#_Toc164039909)

[Download Service 12](#_Toc164039910)

[1. Service Setup: 12](#_Toc164039911)

[2. Service Execution: 12](#_Toc164039912)

[3. Image Download: 12](#_Toc164039913)

[4. Glide Target: 13](#_Toc164039914)

[5. Bitmap Compression and Saving: 13](#_Toc164039915)

[6. Inserting File into MediaStore: 13](#_Toc164039916)

[7. Writing Bitmap to File: 13](#_Toc164039917)

[8. File Name Generation: 14](#_Toc164039918)

[Conclusion 14](#_Toc164039919)

# Project Description

Applying Firebase Service, especially the Storage Service for uploading images from Mobile device and view the uploaded images from server with smoothy, fast loading.

# Project target

Loading images from Firebase server with:

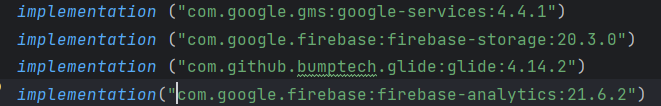
- Atleast 500 images

- Total size atleast 1GB

# Uploading Mechanic

## Dependencies:

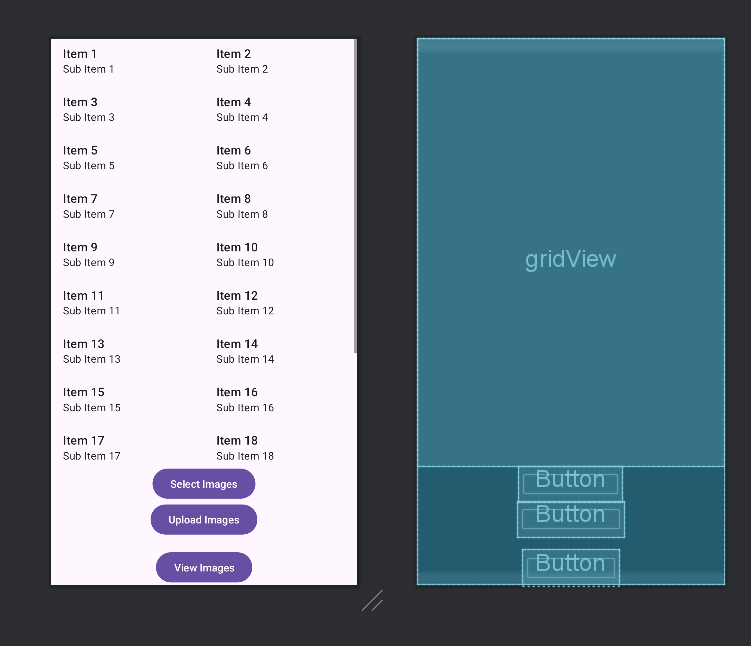
The code imports necessary libraries and classes, such as `FirebaseStorage` and `StorageReference` from the Firebase SDK, and `Glide` for image loading.



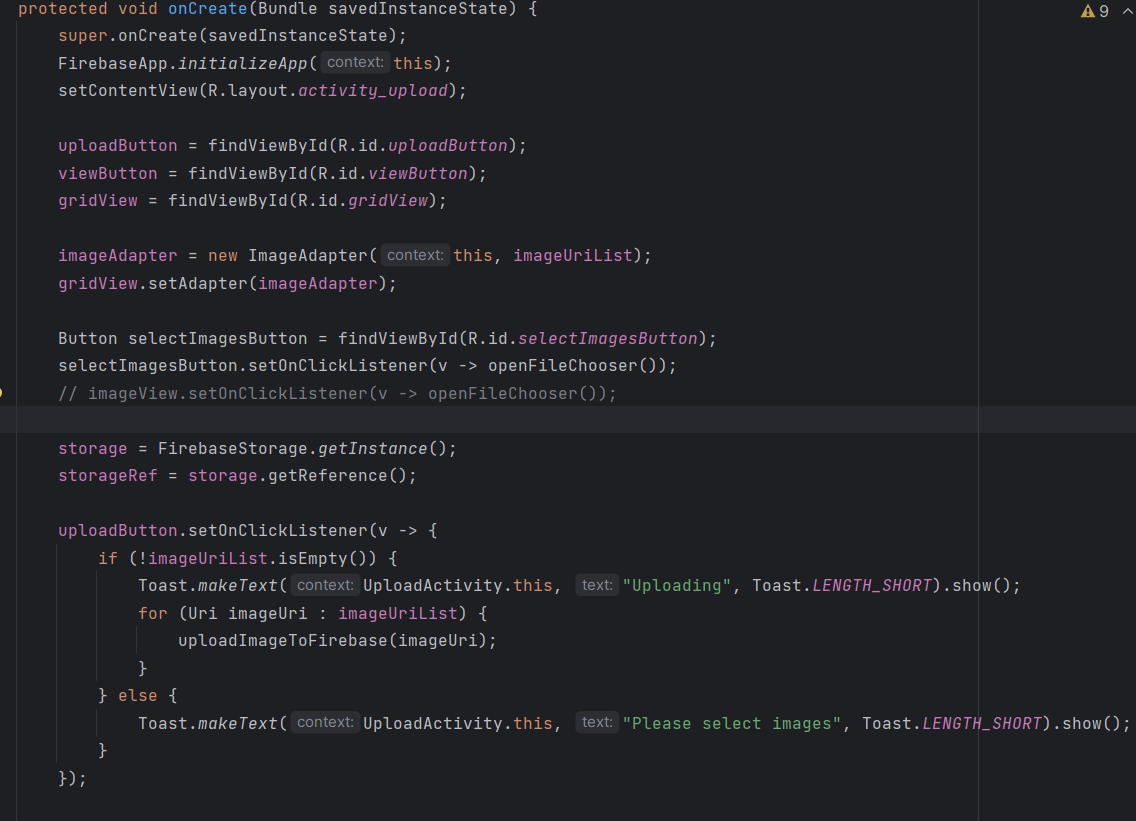
*Some implementations for the gradle build*

## Activity Setup:

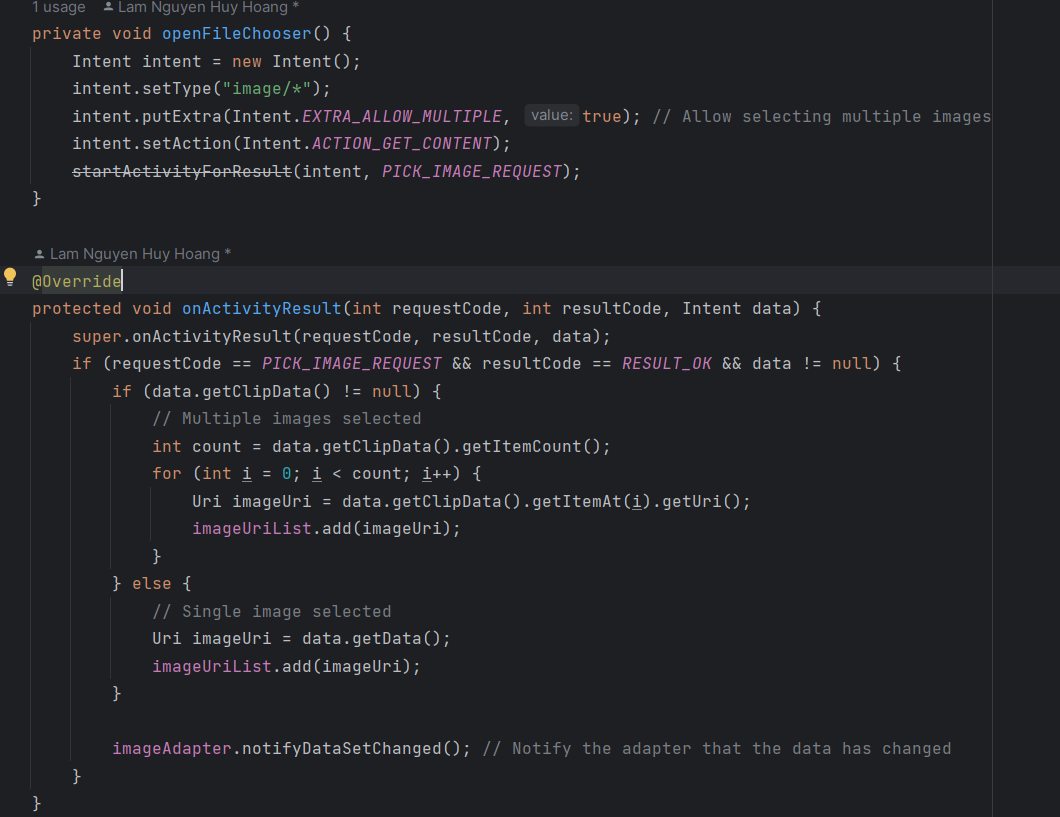
The `UploadActivity` class extends `AppCompatActivity` and initializes the views (ImageView, Buttons) and Firebase Storage objects in the `onCreate` method.

**

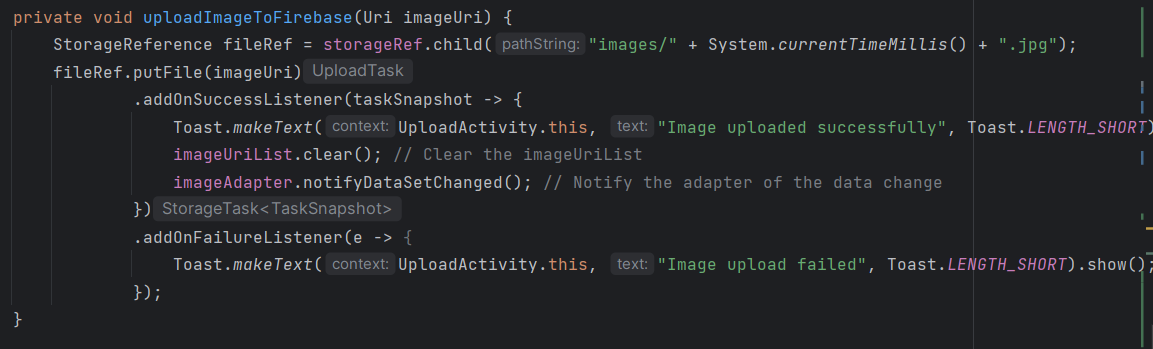
*Designed UI*



*Initial functions*

**

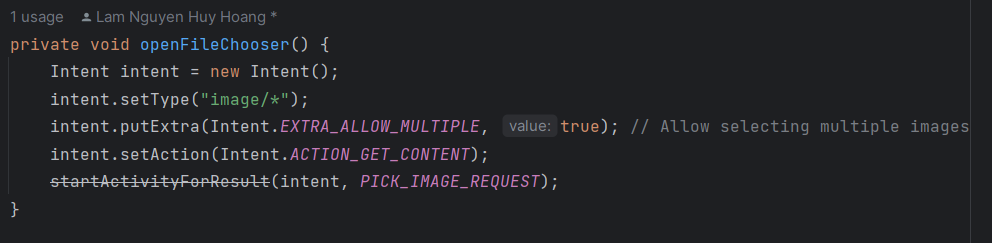
*Utilities functions*



*Firebase Handler function*

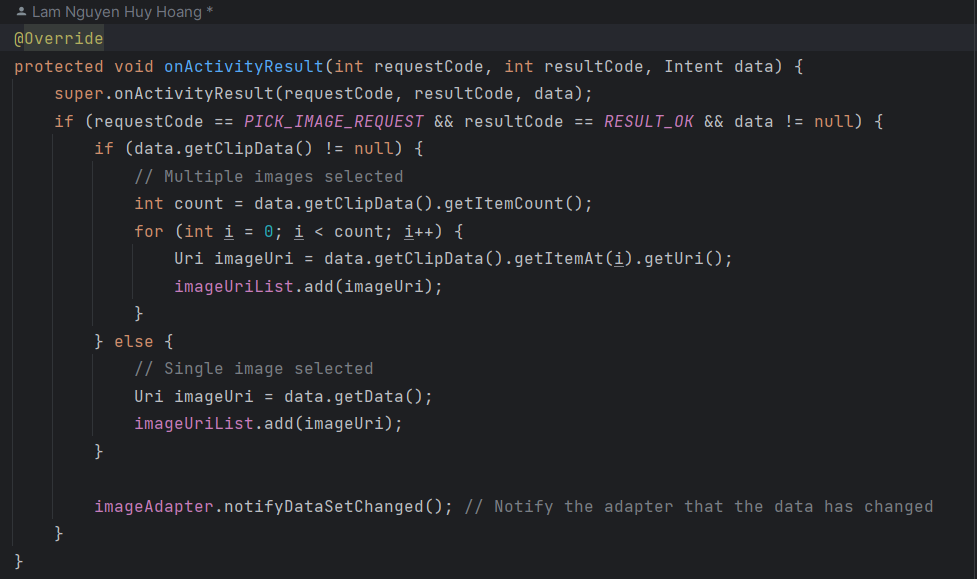
## File Chooser:

When the ImageView is clicked, the `openFileChooser` method is called, which launches an Intent to allow the user to select images from their device's file system.



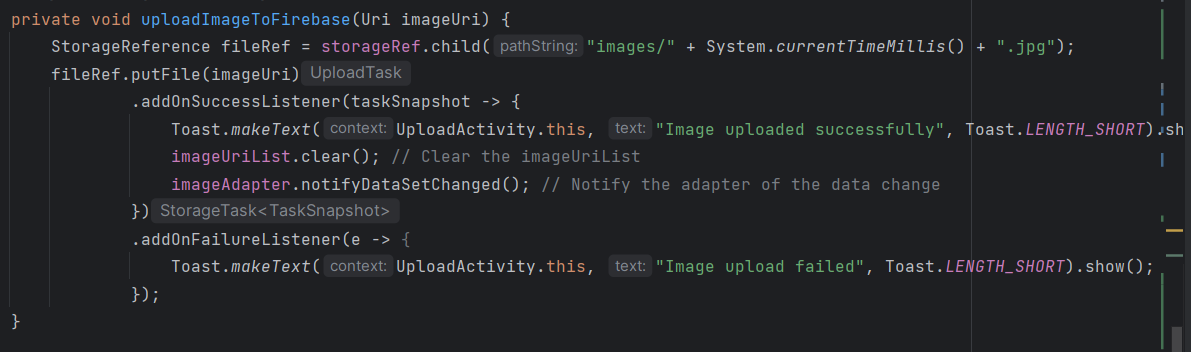
## Image Selection:

After the user selects images, the `onActivityResult` method is called with the selected image's URI. The image is then loaded into the ImageView using the Glide library.



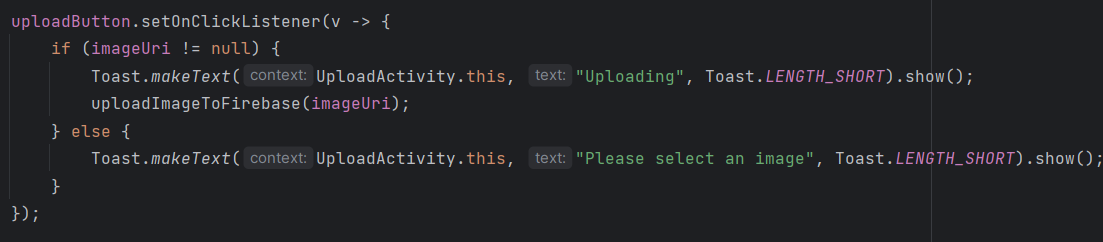
## Image Upload:

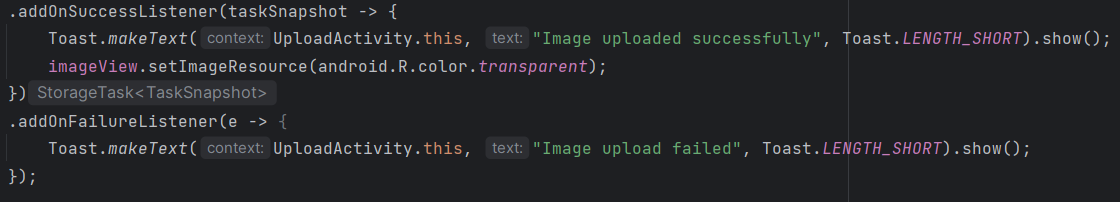
When the "Upload" button is clicked, the `uploadImageToFirebase` method is called. It creates a unique filename based on the current timestamp and uploads the selected image to the "images" directory in Firebase Storage.



## Upload Success/Failure:

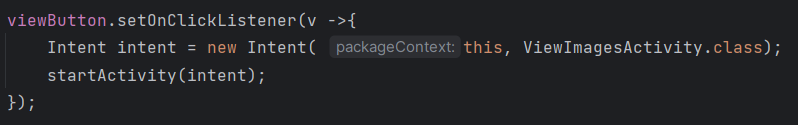
After the upload is complete, a Toast message is displayed to indicate whether the upload was successful or failed. If successful, the ImageView is cleared by setting its resource to `android.R.color.transparent`.





## View Images:

When the "View" button is clicked, a new `ViewImagesActivity` is started. This activity is likely responsible for displaying the uploaded images, although its implementation is not provided in the given code.



## Firebase Integration:

The code initializes the Firebase app and Firebase Storage instance using the `FirebaseApp.initializeApp` and `FirebaseStorage.getInstance` methods, respectively.

# View Images mechanic

**Mechanic:** Using Placeholder

**Detail:**

The use of a placeholder image is a common technique in Android development to provide a better user experience. When loading remote images, there is often a delay before the actual image data is retrieved and displayed. Without a placeholder, the user would see a blank space or a broken image icon, which can be confusing or visually unappealing. By setting a placeholder image, the user sees a temporary image while the actual image is being loaded, providing a smoother and more polished experience.

In the grid view of images, we reduce the resolution of loaded image for faster load, we can called it as the lazy view, and when click into image to view it on fullscreen, we will able to view it with the original resolution.

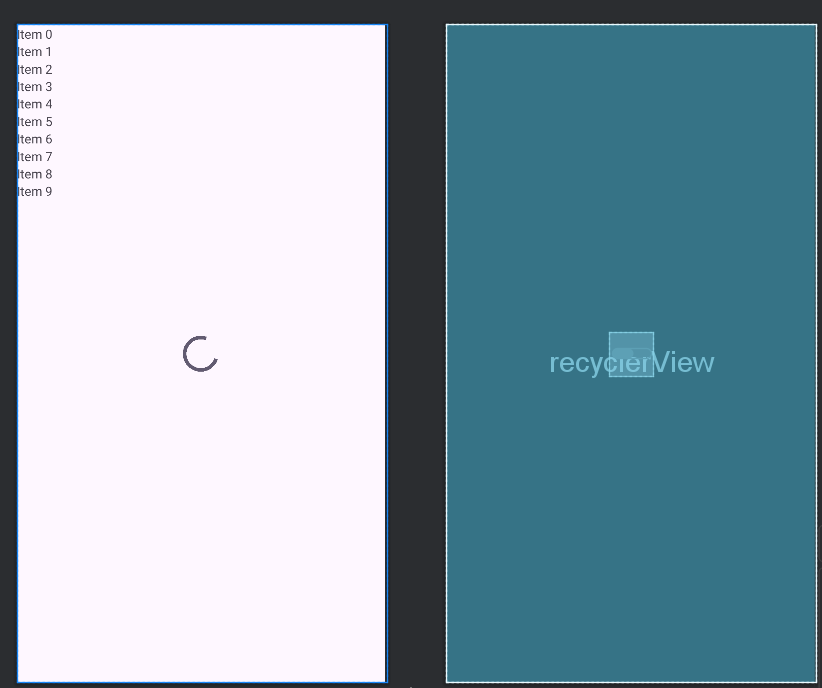
The placeholder image is set both when creating the **ImageView** in the **onCreateViewHolder** method and when loading the actual image using Glide in the **onBindViewHolder** method. This ensures that the placeholder is always displayed initially, and then replaced by the actual image once it is loaded from Firebase Storage.

## Activity Setup:

The activity sets up the `RecyclerView` and initializes the `ImageAdapter`. It also gets a reference to the Firebase Storage and the "images" directory.



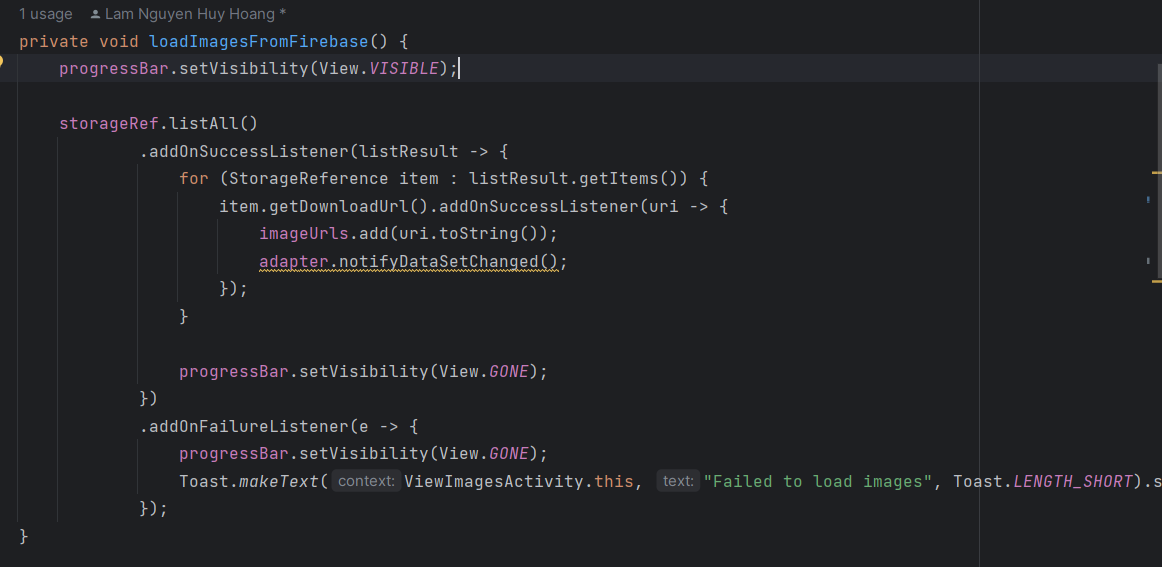
*Initial function*



*Designed UI*

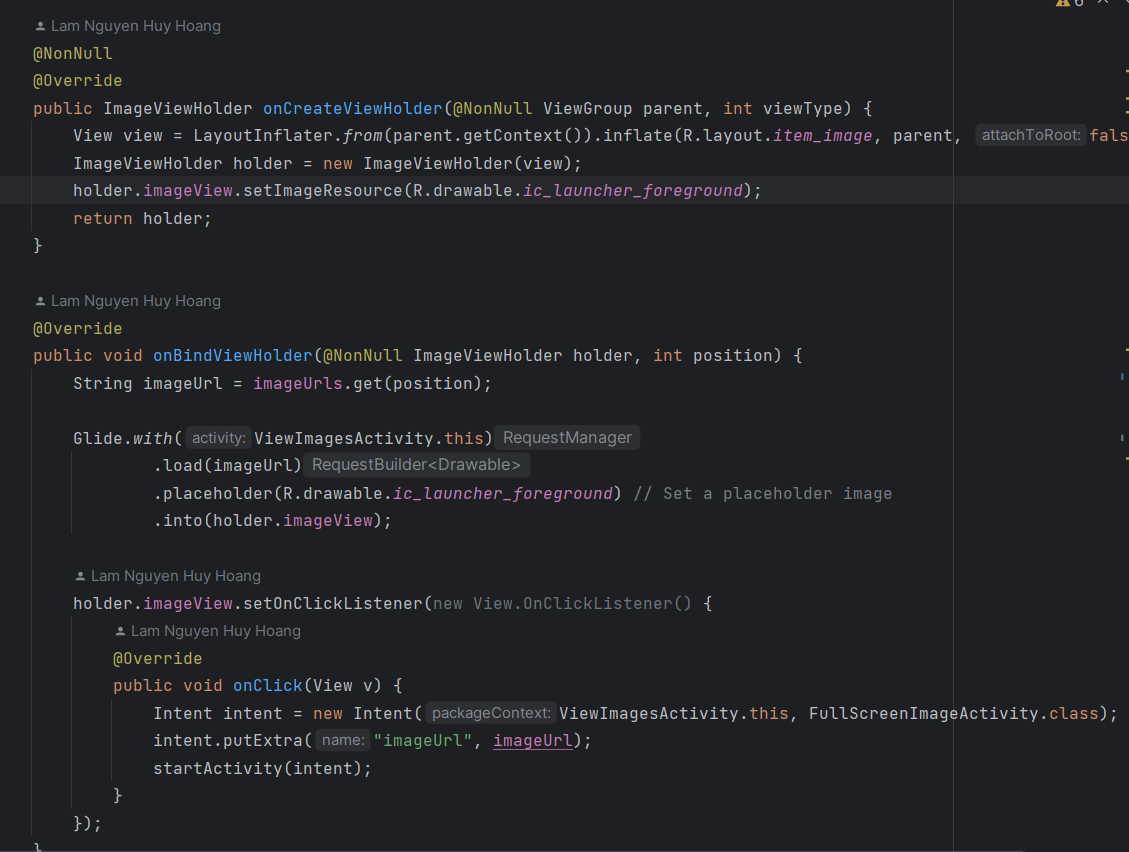
## Loading Images:

The `loadImagesFromFirebase` method is called to retrieve the list of image URLs from Firebase Storage. This is done by calling `storageRef.listAll()` and iterating over the resulting list of `StorageReference` objects. For each item, the `getDownloadUrl` method is called to get the download URL of the image, which is then added to the `imageUrls` list.



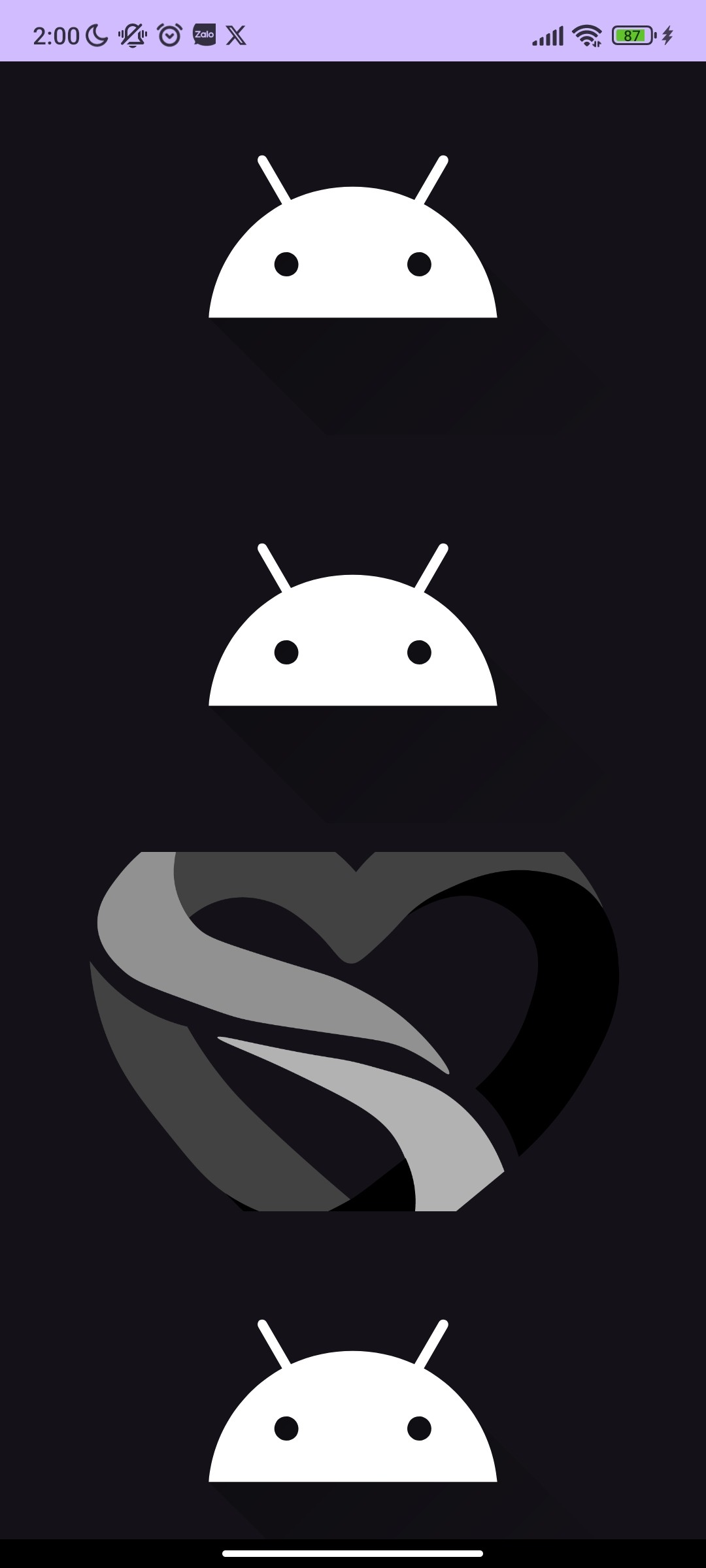
## ImageAdapter:

The `ImageAdapter` is a custom `RecyclerView.Adapter` that handles the creation and binding of the image views in the `RecyclerView`.

****

## Placeholder:

In the `onCreateViewHolder` method of the `ImageAdapter`, a placeholder image is set for the `ImageView` using `holder.imageView.setImageResource(R.drawable.ic\_launcher\_foreground)`. This placeholder is displayed before the actual image is loaded from Firebase Storage.



## Image Loading with Glide:

In the `onBindViewHolder` method, the `Glide` library is used to load the image URL into the `ImageView`. The `placeholder` method is called and passed the same placeholder image resource (`R.drawable.ic\_launcher\_foreground`) to display this placeholder while the actual image is being loaded from Firebase Storage.

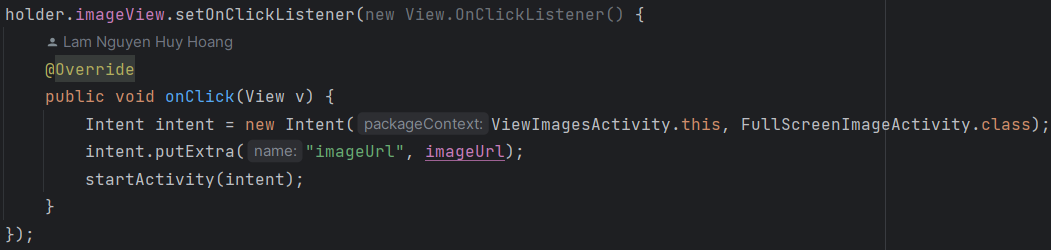
## Lazy view Image:

Showing image in Lazy mode in the Grid view for faster load



## Full Screen Image:

When an image in the `RecyclerView` is clicked, an `Intent` is started to launch the `FullScreenImageActivity`, passing the image URL as an extra.



# Full Screen Viewer

The mechanics of this code involve displaying an image in full-screen mode and providing a way for the user to download the image. When the user clicks on the full-screen image, a popup dialog appears with a button to initiate the download process. Clicking the download button starts a service (ImageDownloadService) that handles the actual downloading of the image from the provided URL.

The code does not include the implementation of the ImageDownloadService, so it's not clear how the actual download process is handled. However, it's common to use services for long-running tasks like downloading files, as services can run in the background without being tied to the activity's lifecycle.

## Activity Setup:

The activity sets up the ImageView (fullScreenImageView) and retrieves the image URL from the Intent extras. The image URL is passed from the ViewImagesActivity when an image is clicked.

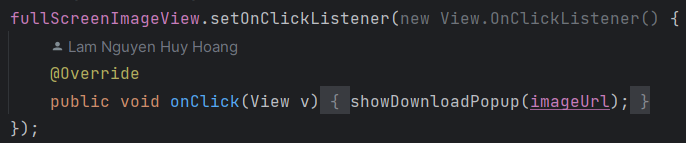


## Image Loading:

The Glide library is used to load the image from the provided URL into the fullScreenImageView.

## Click Listener:

A click listener is set on the fullScreenImageView. When the image is clicked, the showDownloadPopup method is called, which displays a popup dialog for downloading the image.



## Download Popup:

The showDownloadPopup method creates a new Dialog and sets its layout to R.layout.popup\_download\_image. This layout likely contains a button or other UI elements for initiating the download.



## Download Button:

Inside the popup dialog, the code finds the Button with the ID downloadButton and sets a click listener on it.

## Download Service:

When the download button is clicked, a new Intent is created to start the ImageDownloadService. The image URL is passed as an extra to the service using serviceIntent.putExtra("imageUrl", imageUrl). The startService method is called to start the service, which is likely responsible for downloading the image from the provided URL.

# Download Service

The mechanics of this code involve downloading an image from a remote URL using the Glide library, compressing the downloaded bitmap, and saving it to the device's external storage by inserting a new entry into the MediaStore. The IntentService is used to perform this long-running task in the background, without blocking the main UI thread.

By saving the image to the external storage, the user can access the downloaded image through the device's gallery or file manager app. The use of the MediaStore ensures that the downloaded image is properly registered with the device's media database and can be easily managed and accessed by other applications.

## Service Setup:

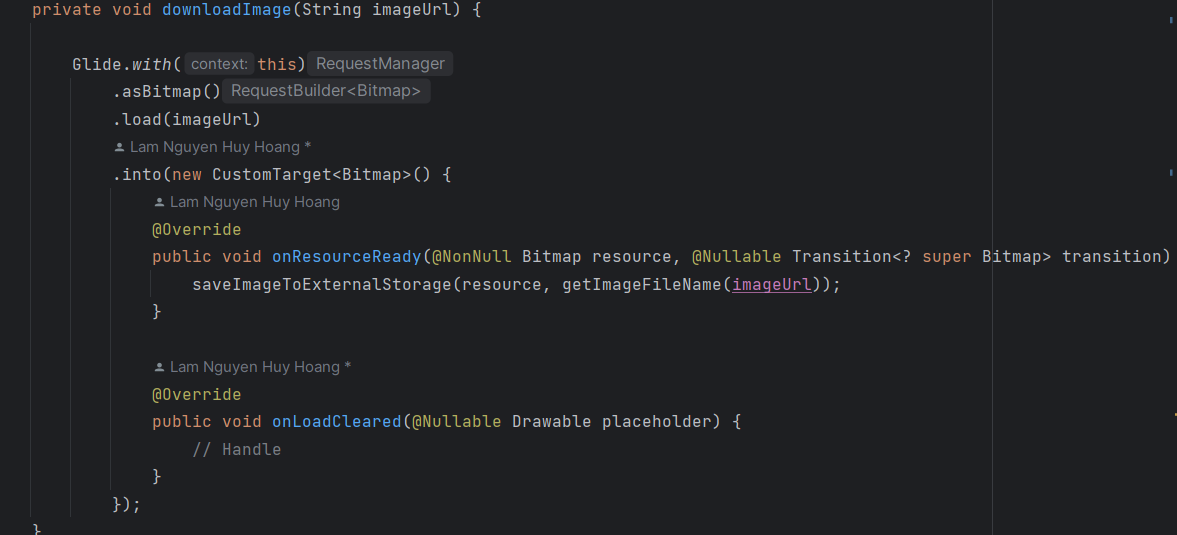
The ImageDownloadService extends the IntentService class, which is a base class for services that handle asynchronous requests on a single worker thread. The service is named "ImageDownloadService".

## Service Execution:

The onHandleIntent method is called when the service is started. It retrieves the image URL from the Intent extras passed by the calling activity (FullScreenImageActivity).

## Image Download:

The downloadImage method is called with the image URL. It uses the Glide library to download the image asynchronously.

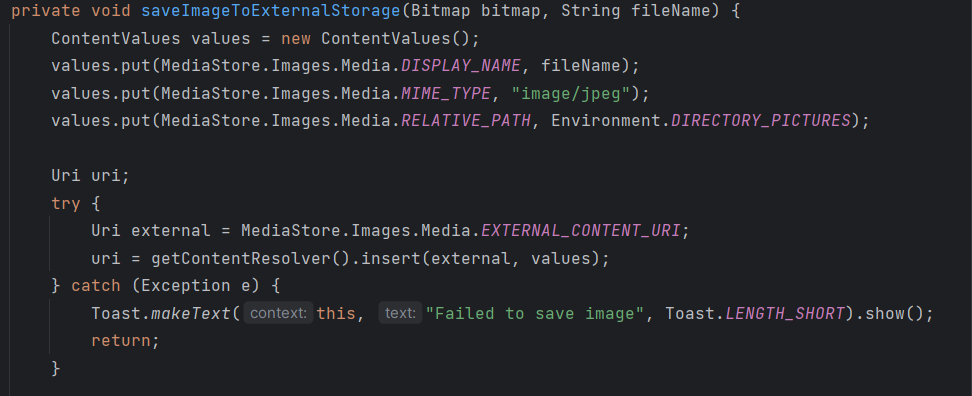


## Glide Target:

A custom CustomTarget is used with Glide to handle the downloaded image. When the image is ready, the onResourceReady callback is invoked with the downloaded bitmap. The saveImageToExternalStorage method is then called to save the bitmap to the device's external storage.

## Bitmap Compression and Saving:

The saveImageToExternalStorage method first creates a ContentValues object to store the metadata for the image file. It sets the display name (file name), MIME type, and relative path (Pictures directory) for the image.



## Inserting File into MediaStore:

The code then inserts a new entry into the MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI to create a new file in the external storage. The insert method returns a Uri representing the location of the new file.

## Writing Bitmap to File:

If the Uri is not null, an OutputStream is obtained using getContentResolver().openOutputStream(uri). The bitmap is then compressed and written to the output stream using bitmap.compress(Bitmap.CompressFormat.JPEG, 100, outputStream). Finally, the output stream is flushed and closed.



## File Name Generation:

The getImageFileName method generates a unique file name for the downloaded image by appending the current timestamp to the string "image\_" and adding the ".jpg" extension.

# Conclusion

Demonstrates a complete workflow for uploading images to Firebase Storage, displaying the uploaded images, allowing users to view images in full-screen mode, and providing the ability to download images to the device's external storage.

The project consists of four main components:

1. **UploadActivity:** This activity allows users to select an image from their device's file system and upload it to Firebase Storage. It uses the Firebase Storage SDK to handle the upload process and provides feedback to the user through Toast messages.
2. **ViewImagesActivity:** This activity displays the images uploaded to Firebase Storage in a `RecyclerView`. It retrieves the list of image URLs from Firebase Storage and uses the Glide library to load and display the images efficiently. The activity also implements a placeholder image to provide a better user experience while the images are being loaded.
3. **FullScreenImageActivity:** When an image is clicked in the `ViewImagesActivity`, this activity is launched to display the image in full-screen mode. It also provides a popup dialog with a button to initiate the download process for the displayed image.
4. **ImageDownloadService:** This `IntentService` is responsible for downloading the image from the provided URL and saving it to the device's external storage. It uses the Glide library to download the image asynchronously and writes the downloaded bitmap to a new file in the device's Pictures directory. The `MediaStore` is used to ensure that the downloaded image is properly registered with the device's media database.

The project demonstrates several key Android development concepts and techniques, including:

- Integration with Firebase Storage for file uploads and downloads

- Use of `RecyclerView` and adapters for displaying lists of data

- Efficient image loading and caching with the Glide library

- Implementation of placeholders for better user experience

- Handling of user interactions and events

- Passing data between activities using intents

- Use of dialogs for user input and confirmation

- Background task execution with `IntentService`

- Saving files to external storage and working with the `MediaStore`