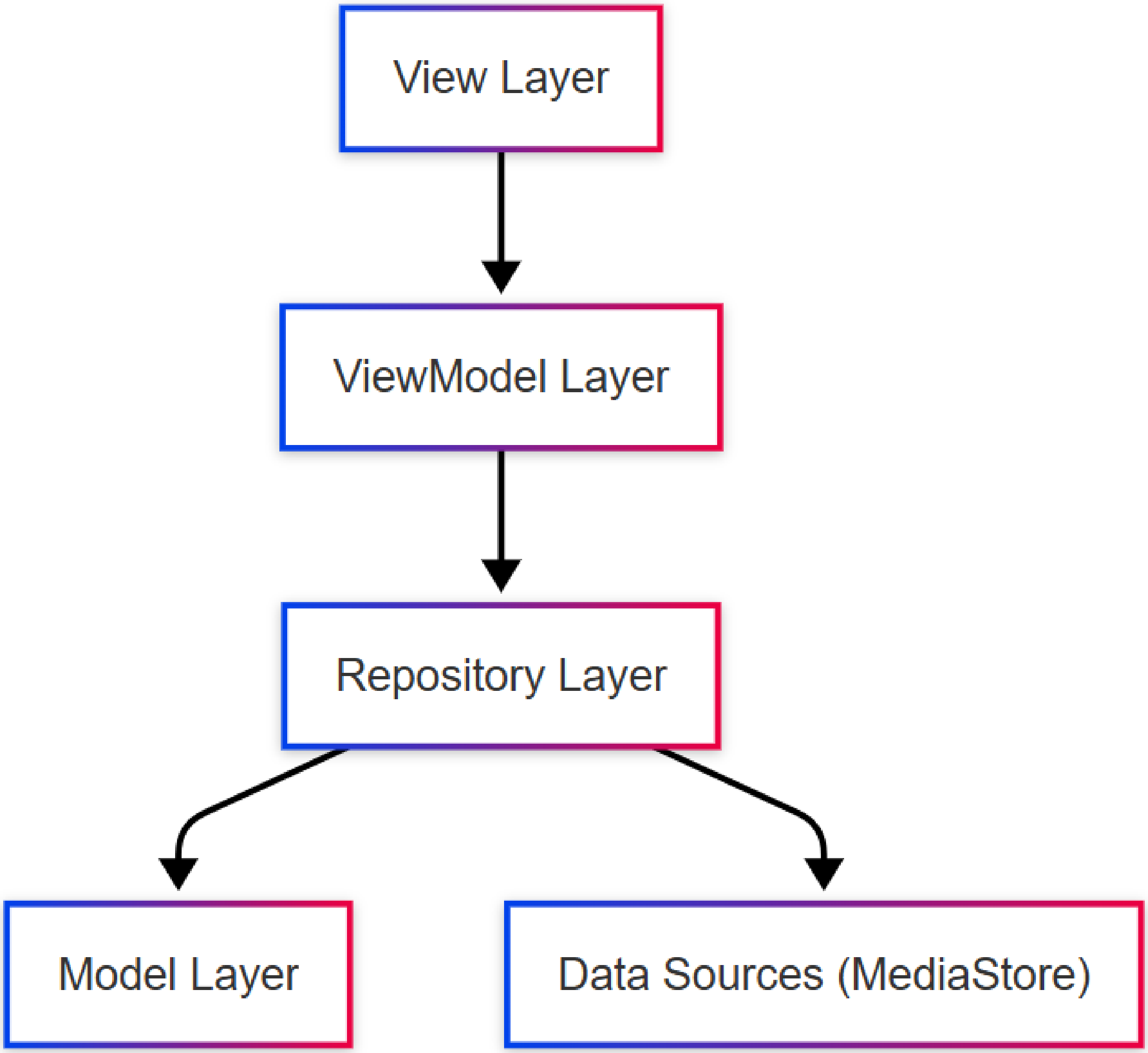
MediaVault: MediaVault-MVVM-Android-Example

Project Overview

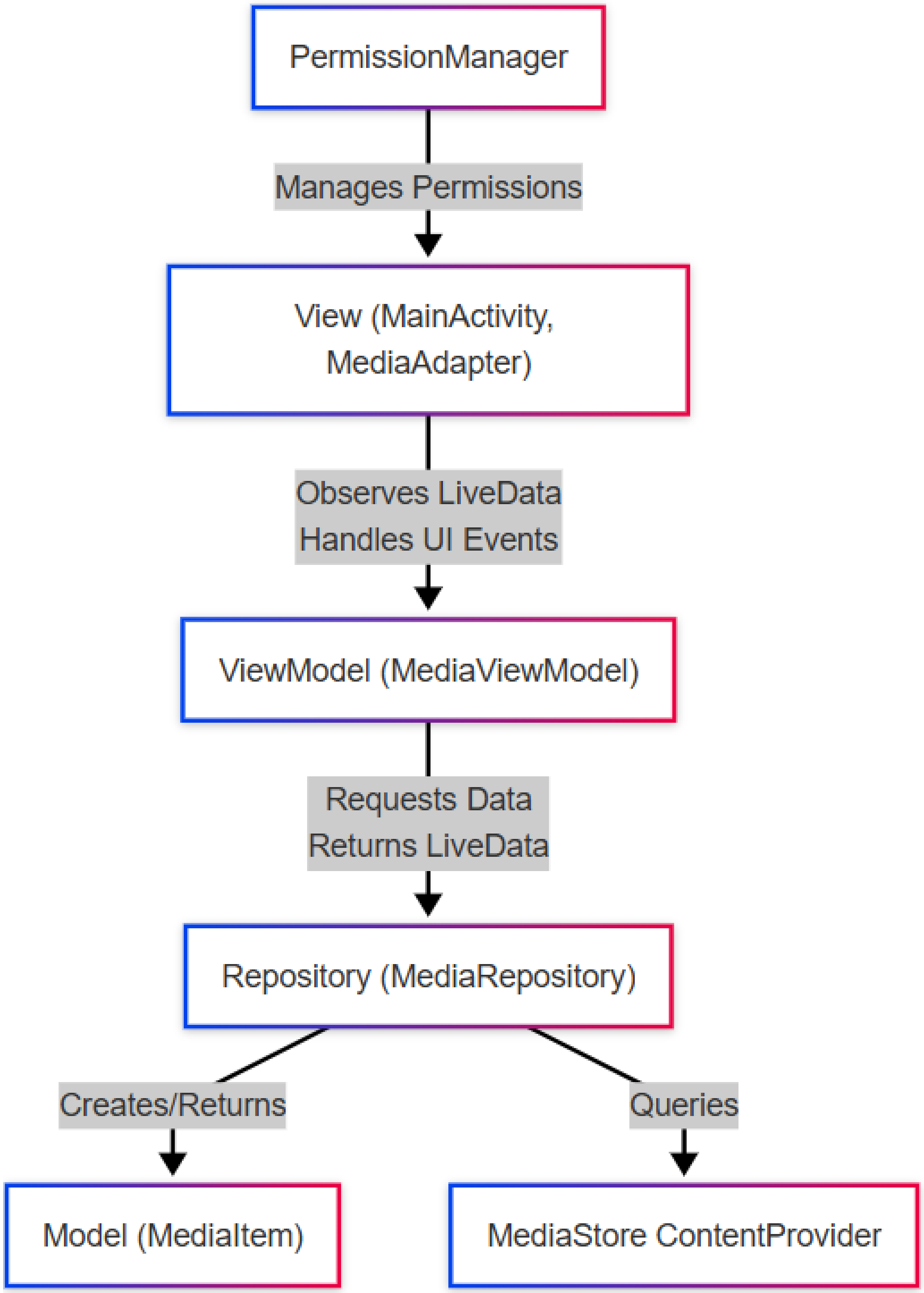
MediaVault is an Android application that allows users to browse photos and videos stored on their device. The app follows the MVVM (Model-View-ViewModel) architecture pattern and implements modern Android development practices including Kotlin, Coroutines, and Koin for dependency injection.



Features

1. **Media Gallery**
2. Browse photos from device storage
3. Browse videos from device storage
4. Grid layout display with thumbnails
5. **Tab Navigation**
6. Switch between Photos and Videos tabs
7. Persistent state when switching tabs
8. **Permission Handling**
9. Runtime permission requests
10. Permission rationale dialogs
11. Settings redirection for permanently denied permissions
12. SDK version-specific permission handling
13. **UI Components**
14. Responsive grid layout
15. Loading indicators
16. Empty state handling
17. Video indicators on thumbnails

Architecture: MVVM Implementation



Folder Structure

```
app/src/main/java/com/example/mediavault/
├── di/
│   └── AppModule.kt                 # Dependency injection module
├── model/
│   └── MediaItem.kt                 # Data models
├── repository/
│   └── MediaRepository.kt           # Repository layer
├── utils/
│   └── PermissionManager.kt         # Utility classes
├── view/
│   ├── MainActivity.kt              # UI components
│   └── MediaAdapter.kt              # RecyclerView adapter with DiffUtil
├── viewmodel/
│   └── MediaViewModel.kt            # ViewModels
└── MediaVaultApplication.kt         # Application class
```

Implementation Details

1. Dependency Injection with Koin

Koin is used for dependency injection, providing a lightweight alternative to Dagger/Hilt:

```kotlin
// AppModule.kt
val appModule = module {
    // Single instance of MediaRepository
    single { MediaRepository(androidContext()) }

    // ViewModel
    viewModel { MediaViewModel(get()) }
}

// MediaVaultApplication.kt
startKoin {
    androidLogger(Level.ERROR)
    androidContext(this@MediaVaultApplication)
    modules(appModule)
}
```

2. Media Loading with MediaStore API

The app uses the MediaStore ContentProvider to access media files:

```kotlin
// MediaRepository.kt
val collection = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
    MediaStore.Images.Media.getContentUri(MediaStore.VOLUME_EXTERNAL)
} else {
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI
}
```

```
context.contentResolver.query(
    collection,
    projection,
    null,
    null,
    sortOrder
)?.use { cursor ->
    // Process cursor and create MediaItem objects
}
```

3. Permission Handling

The app implements a custom PermissionManager to handle runtime permissions:

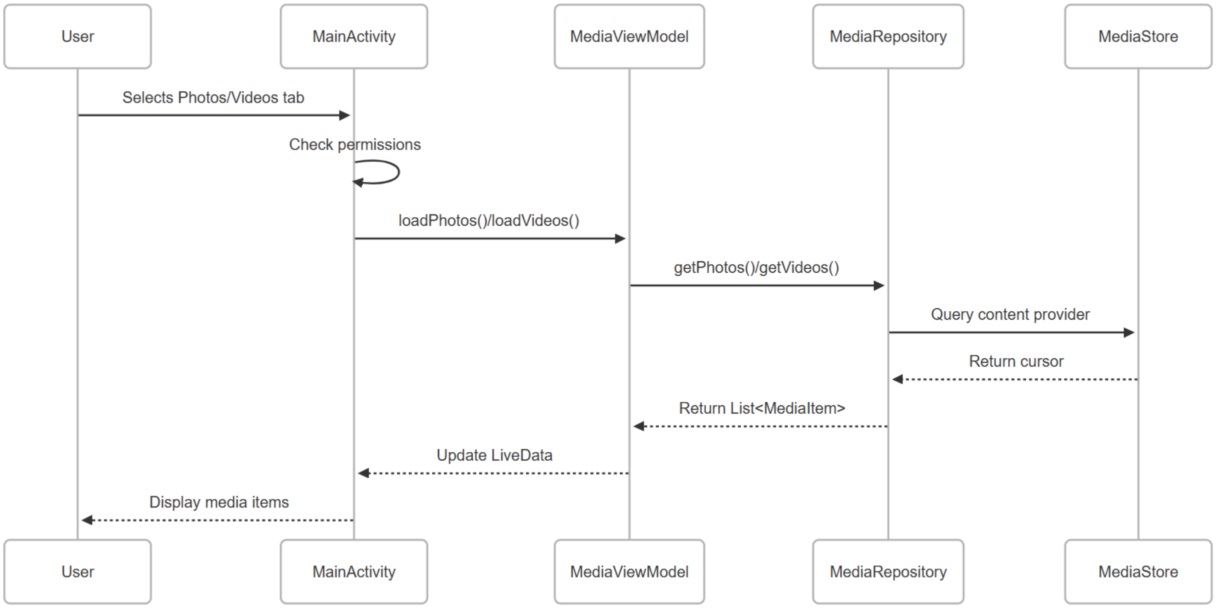
```
// PermissionManager.kt
fun requestPermission(
    permission: String,
    onGranted: () -> Unit,
    onDenied: () -> Unit,
    onPermanentlyDenied: () -> Unit
) {
    // Check if permission is already granted
    if (ContextCompat.checkSelfPermission(activity, permission) ==
            PackageManager.PERMISSION_GRANTED) {
        onGranted()
        return
    }

    // Request the permission...
}
```

4. RecyclerView with DiffUtil

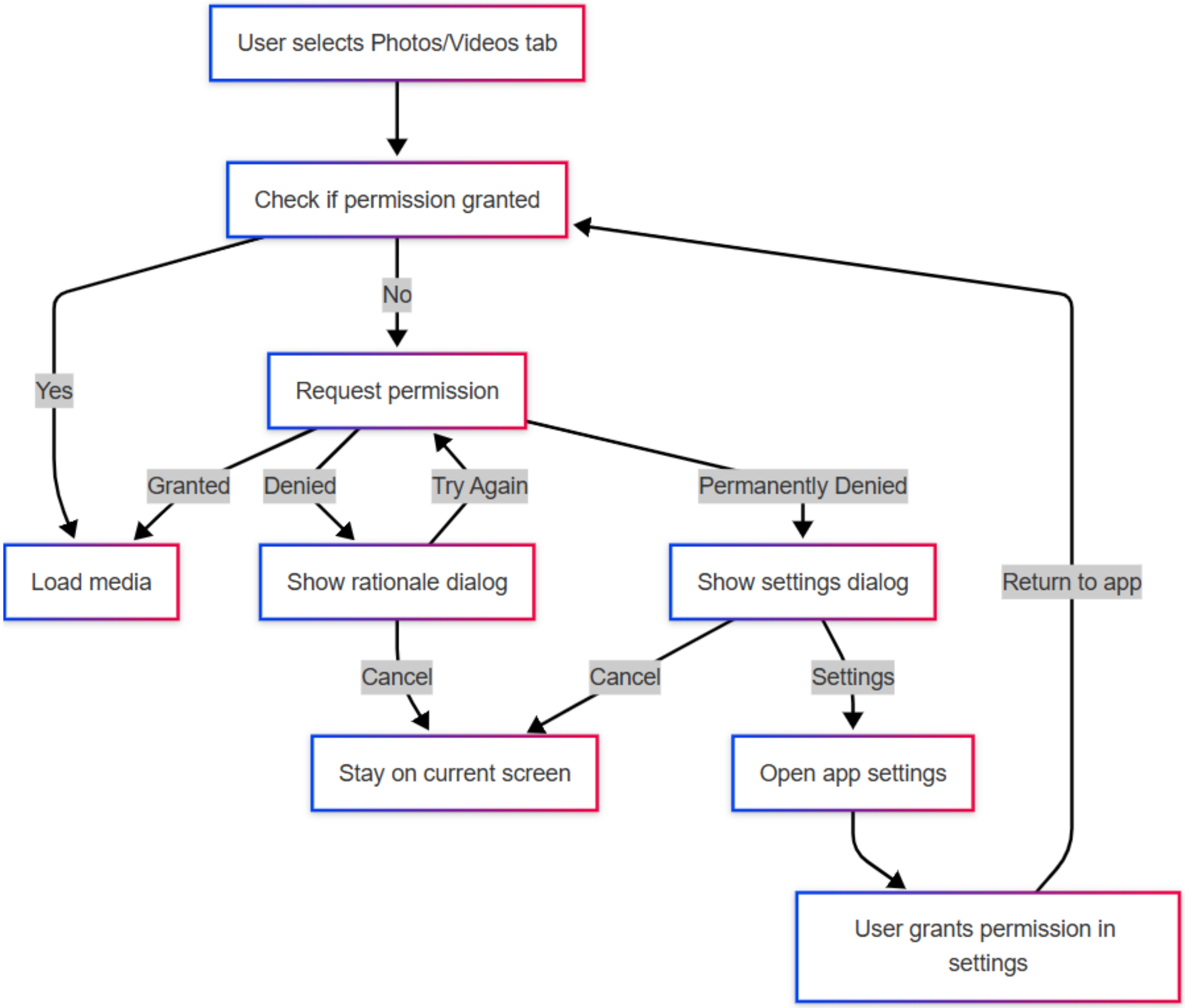
The app uses ListAdapter with DiffUtil for efficient list updates:

```
// MediaAdapter.kt
class MediaAdapter(private val isVideo: Boolean)
    : ListAdapter<MediaItem,
MediaAdapter.MediaViewHolder>(MediaDiffCallback()) {
    // ...
}

private class MediaDiffCallback : DiffUtil.ItemCallback<MediaItem>() {
    override fun areItemsTheSame(oldItem: MediaItem, newItem: MediaItem):
Boolean {
        return oldItem.id == newItem.id
    }

    override fun areContentsTheSame(oldItem: MediaItem, newItem: MediaItem):
Boolean {
        return oldItem == newItem
    }
}
```

Data Flow


User
MainActivity
MediaViewModel
MediaRepository
MediaStore
Selects Photos/Videos tab
Check permissions
loadPhotos()/loadVideos()
getPhotos()/getVideos()
Query content provider
Return cursor
Return List<MediaItem>
Update LiveData
Display media items
User
MainActivity
MediaViewModel
MediaRepository
MediaStore

Permission Handling Flow



Step-by-Step Implementation Guide

Step 1: Project Setup

1. Create a new Android project with Kotlin support
2. Add dependencies for:
3. AndroidX (AppCompat, ConstraintLayout)
4. ViewModel and LiveData
5. Coroutines
6. Koin for dependency injection
7. Glide for image loading

Step 2: Define Data Model

1. Create the MediaItem data class to represent media files
2. Include properties for id, uri, name, date, size, and mimeType

Step 3: Implement Repository Layer

1. Create MediaRepository to handle data operations
2. Implement methods to query MediaStore for photos and videos
3. Handle Android version differences for MediaStore access

Step 4: Create ViewModel

1. Create MediaViewModel to manage UI-related data
2. Implement LiveData for photos, videos, and loading state
3. Create methods to load photos and videos from the repository

Step 5: Implement Permission Manager

1. Create PermissionManager to handle runtime permissions
2. Implement logic for permission requests, denials, and permanent denials
3. Track permission denial count to detect "Don't ask again" selections

Step 6: Create UI Components

1. Design activity_main.xml with TabLayout and RecyclerView
2. Create item_media.xml for grid items
3. Implement MediaAdapter with DiffUtil for efficient updates

Step 7: Implement MainActivity

1. Set up TabLayout for Photos/Videos switching
2. Initialize adapters and observe ViewModel LiveData
3. Implement permission checking and handling
4. Create dialogs for permission rationale and settings redirection

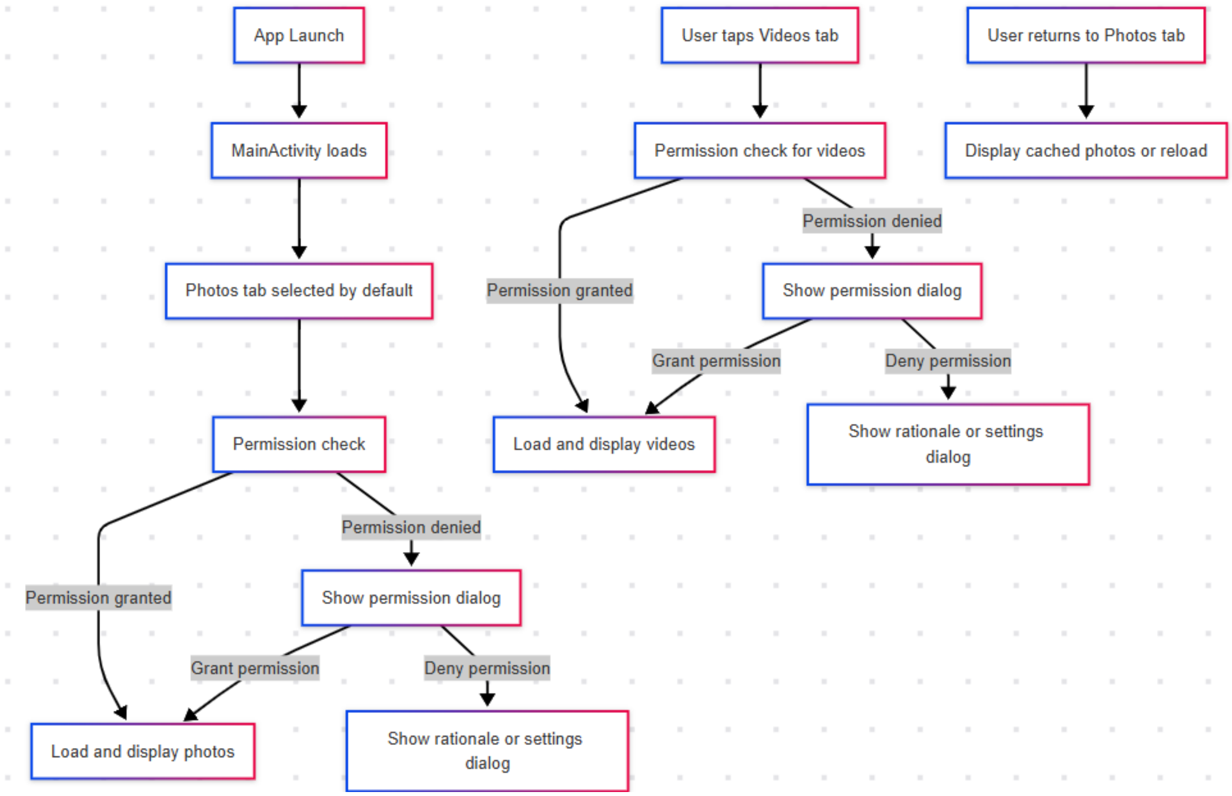
Step 8: Set Up Dependency Injection

1. Create AppModule to provide dependencies
2. Configure Koin in the Application class

Step 9: Configure AndroidManifest

1. Declare required permissions based on Android versions
2. Set up the application and activity declarations

User Flow



Key Components and Responsibilities

1. MainActivity

- Manages UI state and user interactions
- Handles tab selection and adapter switching
- Observes ViewModel LiveData
- Coordinates permission requests

2. MediaViewModel

- Exposes LiveData for UI consumption
- Manages loading state
- Coordinates data loading from repository
- Handles background processing with Coroutines

3. MediaRepository

- Provides data access methods
- Queries MediaStore ContentProvider
- Handles Android version-specific implementations
- Maps raw data to MediaItem objects

4. PermissionManager

- Encapsulates permission request logic
- Tracks permission denial count
- Provides callbacks for different permission states
- Handles permission result processing

5. MediaAdapter

- Displays media items in a grid
- Uses DiffUtil for efficient updates
- Handles different display for photos vs videos
- Loads thumbnails using Glide

Conclusion

MediaVault demonstrates a well-structured MVVM architecture with clean separation of concerns. The app handles runtime permissions properly, efficiently displays media content, and provides a smooth user experience with modern Android development practices.

The implementation showcases:

- Proper architecture with MVVM pattern
- Efficient list handling with RecyclerView and DiffUtil
- Robust permission handling with appropriate user guidance
- Dependency injection with Koin
- Asynchronous operations with Coroutines
- Responsive UI with proper loading and empty states