CS19611 - MOBILE APPLICATION DEVELOPMENT

PROJECT REPORT

MY GALLERY APPLICATION

Submitted by

MANICK VISHAL C (220701158)

in partial fulfilment for the course for the degree

Of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

RAJALAKSHMI ENGINEERING COLLEGE

THANDALAM, CHENNAI - 602105

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project report titled **"MY GALLERY APPLICATION"** is the bonafide work of **MANICK VISHAL C (220701158)**, who carried out the work under my supervision. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation based on which a degree or award was conferred earlier.

SIGNATURE

DR. P. KUMAR
Head of the Department
Computer Science and Engineering
Rajalakshmi Engineering College
Chennai - 602105

SIGNATURE

Dr. V. KARTHICK
Associate Professor
Computer Science Engineering
Rajalakshmi Engineering College
Chennai – 602105

Submitted to Project and Viva Voce Examination for the subject
CS19611 – Mobile Application Development Laboratory held on _____

Internal Examiner

External Examiner

ABSTRACT

The **"My Gallery Application"** is a user-friendly and visually appealing Android mobile application developed to display a collection of images in a dynamic and interactive grid format. This project was undertaken to explore and demonstrate the use of core Android components for creating a simple yet efficient image gallery, without relying on third-party libraries or cloud-based APIs. Designed and developed using **Android Studio** as the Integrated Development Environment (IDE), the application utilizes **Java** for backend logic and **XML** for user interface design.

The primary goal of the application is to enable users to browse through a predefined set of images arranged in a grid layout using the built-in **GridView** component. Each image is presented in a uniform, scrollable grid, offering a consistent visual structure. Upon clicking any image, a responsive feedback mechanism is triggered using a **Toast** message, indicating successful interaction and laying the groundwork for potential enhancements like image preview or magnification.

Key features of the application include dynamic image loading from local drawable resources, efficient use of Android adapters **(BaseAdapter)** for scalable UI rendering, and an intuitive layout managed using **RelativeLayout** and **GridView.** The design ensures that the app remains lightweight, performs efficiently on various screen sizes, and delivers a seamless user experience.

The project emphasizes modular code design, clean UI development, and hands-on experience with Android's activity lifecycle, layout managers, and image-handling techniques. The expected outcomes include improved understanding of Android UI development, effective implementation of adapters and views, and a foundation for more advanced gallery applications that may support image capture, storage, editing, or sharing in future versions.

Overall, the **My Gallery Application** serves as an educational and practical demonstration of building a basic gallery viewer using native Android tools, offering potential for both academic learning and further innovation.

ACKNOWLEDGEMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S. Meganathan, B.E, F.I.E.,** our Vice Chairman **Mr. Abhay Shankar Meganathan,B.E.,M.S.,** and our respected Chairperson **Dr. (Mrs.) Thangam Meganathan, Ph.D.,** for providing us with the requisite infrastructure and sincere endeavouring in educating us in their premier institution.

Our sincere thanks to **Dr. S. N. Murugesan, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to our **DR. P. Kumar** Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere thanks to our internal guide and Project Coordinator, **Dr. V. Karthick**, Rajalakshmi Engineering College for his valuable guidance throughout the course of the project.

MANICK VISHAL C (220701158)

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1

1.INTRODUCTION

In the era of mobile computing, smartphones have become central to the way users capture, store, and interact with multimedia content, particularly images. As camera technology on mobile devices continues to improve and cloud storage becomes more accessible, the demand for efficient and responsive image-viewing applications has risen significantly. Users now expect gallery apps to be fast, intuitive, and visually pleasing, allowing them to browse, organize, and interact with their images effortlessly.

Despite the availability of various third-party and built-in gallery applications such as Google Photos, many users still seek simpler, ad-free, and lightweight alternatives that focus on core functionalities. These include basic image viewing, grid-based navigation, and responsive touch interactions. Additionally, developers and learners in the Android ecosystem often require clear and customizable examples to better understand how to work with fundamental UI components and activities.

Background on Mobile Apps and Image Viewing Applications

The proliferation of Android devices globally has fueled the demand for diverse mobile applications. Among the most frequently used are image-viewing or gallery applications. From casual users who wish to revisit memories to professionals who manage large volumes of visual content, the need for reliable gallery apps is universal. Most commercial gallery apps are feature-rich but often come at the cost of resource consumption, privacy concerns, or overcomplication.

From a technical perspective, building an image gallery app serves as an ideal starting point for Android development. It introduces developers to essential concepts such as layout management, event handling, adapters, and resource optimization. **GridView,** a commonly used component in such applications, provides a flexible and scrollable grid structure for displaying images, making it a powerful tool for building custom galleries.

Importance of User-Friendly Gallery Apps

In today's fast-paced digital environment, user experience (UX) plays a pivotal role in application success. An image gallery app must not only load content efficiently but also offer a clean, responsive, and intuitive interface. Users expect consistent performance across devices, seamless navigation, and immediate feedback on interactions such as clicks or long-presses.

User-friendly gallery apps eliminate unnecessary complexity, reduce cognitive load, and allow users to focus solely on their visual content. They also ensure accessibility for users across different age groups and technical backgrounds. By developing a simplified gallery app using core Android technologies, the aim is to strike a balance between functionality and simplicity, offering a focused and efficient user experience.

Motivation for Developing this Project

The motivation behind the development of **My Gallery Application** was twofold. First, it serves as a practical exercise in understanding Android application development fundamentals, particularly in working with layout components, adapters, and image rendering. Second, it addresses the need for a clean and minimal gallery solution that avoids the pitfalls of bloated or overly complex alternatives.

As a student or aspiring developer, building a functional image gallery app offers firsthand experience with Android Studio, Java programming, and XML layout design. It provides insight into the complete app development lifecycle — from layout design and resource management to user event handling and interface testing. The experience gained from such a project can be extended to more advanced applications that incorporate animations, filters, cloud integration, and media editing.

Problem Statement and Solution Offered by the App

While many pre-installed and downloadable gallery applications are available on Android, they often come with several limitations such as unnecessary features, performance lags, or aggressive data synchronization that compromises privacy. Users looking for basic functionality — simply viewing images stored on their device — may find these apps overengineered for their needs.

This project attempts to solve the problem by providing a **lightweight, efficient, and customizable gallery application** built from scratch using only the essential components. The app loads images from the app's drawable resources and displays them in a structured grid. Upon tapping an image, a visual confirmation (via **Toast**) is shown, proving the app's responsiveness and scope for interaction. Although the current version does not include advanced functionality like full-screen preview or cloud sync, it provides a solid foundation for such enhancements in the future.

Overview of the Development Process

The development of **My Gallery Application** followed a structured process using **Android Studio**, with a clear division between design (XML layouts) and logic (Java code). The user interface was defined using **RelativeLayout** and **GridView** to ensure images are aligned properly and the UI scales well across screen sizes.

The Java class **MainActivity** was implemented to manage the application's main behavior. It initializes the **GridView** and sets up an **ImageAdapter,** a custom subclass of **BaseAdapter,** to populate image views dynamically. The adapter ensures efficient recycling of views to optimize memory usage and maintain performance. Event handling was introduced via **setOnItemClickListener()** to capture user interactions, allowing the app to respond in real-time when an image is clicked.

Resource images were stored in the drawable folder and referenced using **R.drawable.*** identifiers, making them easily accessible programmatically. This approach ensures that the app does not rely on external internet connectivity, making it fully functional offline.

In summary, the Introduction provides the rationale behind the project, sets the context of its relevance, outlines the development process, and presents the motivations and objectives clearly. **My Gallery Application** is not just a simple tool for viewing images, but also a practical implementation of core Android concepts that serve as the building blocks for more complex applications in the mobile domain.

CHAPTER 2

2.LITERATURE SURVEY

Numerous mobile gallery applications dominate the Android ecosystem, ranging from feature-rich commercial apps like **Google Photos** to open-source alternatives such as **Simple Gallery**. While these applications serve the basic purpose of image organization and viewing, they vary significantly in terms of functionality, user interface design, performance, and privacy control. A critical examination of existing gallery applications and related academic studies highlights several opportunities for improvement, especially in creating lightweight, customizable apps suitable for learning and development.

2.1 Review of Existing Gallery Applications

Google Photos, developed by Google, is one of the most widely used photo management apps. It offers advanced features such as automatic backup, facial recognition, and AI-driven categorization. However, these features often come with trade-offs in terms of resource usage, internet dependency, and user privacy. The app frequently runs background processes to sync data with cloud servers, consuming device storage and network bandwidth. Additionally, its closed-source nature limits its adaptability for educational purposes or niche use cases.

Simple Gallery is an open-source alternative known for its offline functionality and user-friendly design. It allows users to view, organize, and edit photos without requiring internet access. Its minimalist interface and support for multiple image formats make it a strong candidate for users who value performance and privacy. However, even Simple Gallery includes numerous features like folders, recycle bins, and image editing, which might be overwhelming or unnecessary for users who seek a basic image viewer.

Other alternatives like A+ Gallery, Piktures, and F-Stop Gallery provide varying levels of performance and complexity. While some of these apps support cloud integration, others focus on high customization, metadata viewing, or slideshow functionality.

Although diverse, most of these gallery applications are developed for end-users rather than learners or developers. They typically abstract core functionality behind complex interfaces, making them unsuitable for beginners aiming to understand basic Android components like **GridView, Adapters,** and event handling.

2.2 Comparison of UI, Performance, and Features

Application	UI Design	Performance	Key Features	Support	Customizability
Google Photos	Polished & modern	Moderate to High	Cloud backup, face grouping, search	Partial	Low
Simple Gallery	Minimalist	High	Offline access, sorting, recycle bin	Full	Moderate
A+ Gallery	Feature-rich	Moderate	Cloud sync, color themes	Partial	Low
My Gallery App *(This Project)*	Basic but clean	High	GridView display, click interaction	Full	High *(code-level)*

As illustrated in the comparison, while existing apps excel in specific areas, few offer a balance of simplicity, performance, and customizability that is ideal for educational projects or lightweight use cases. This gap is what the **My Gallery Application** aims to fill.

2.3 Review of Related Studies and Academic Work

Several academic studies have explored the development and optimization of mobile user interfaces, especially within the context of image rendering and interactive elements. One study by A. Sharma et al. (2020), titled ″Design and Evaluation of Mobile UIs for Image-Based Applications″, published in IEEE Access, emphasized the significance of touch responsiveness and UI clarity in improving user engagement. The paper also stressed the importance of efficient memory usage in image display applications.

Another paper, *"Adaptive Layouts for Mobile Applications Using Android Framework"* by R. Singh et al. (2021), explored the use of Android's built-in layout managers to design responsive and scalable user interfaces. The study found that components such as **GridView** and **RecyclerView** could be leveraged to develop high-performance galleries without relying on external dependencies. These findings support the design choices made in the **My Gallery Application** project.

Further research on mobile computing, such as "Optimizing Image Loading and Rendering on Android Devices" (IEEE, 2019), provided valuable insights into resource-efficient image handling. It recommended using drawable resources for offline image access and emphasized the use of adapter patterns to minimize memory overhead.

2.4 Identified Gaps and Justification for the Proposed Application

From both commercial and academic perspectives, several gaps and limitations have been identified in current gallery apps that justify the need for a new, simplified application:

- **Complexity Overload**: Most gallery apps are overloaded with features that are not always necessary for all users, especially beginners or developers.
- **Cloud Dependency**: Applications like Google Photos rely heavily on internet connectivity and cloud storage, which compromises offline usability and privacy.
- **Limited Educational Value**: Commercial apps are not intended for code inspection or modification, limiting their utility for developers and students.
- **Performance Concerns**: Feature-rich applications often result in performance lag on low-end devices or older Android versions.
- **Lack of Customizability**: End-users have limited control over the app's functionality or appearance unless the app is open source and customizable.

The **My Gallery Application** addresses these gaps by offering a basic, offline gallery built using core Android elements. It demonstrates the use of native UI components like **GridView,** interaction handlers such as **OnItemClickListener,** and efficient image loading using local resources. Its simplicity makes it ideal not only for lightweight use but also as a learning tool for aspiring Android developers.

CHAPTER 3

3.PROPOSED SYSTEM

The proposed system is a standalone Android application named **My Gallery Application**, developed to display a collection of images in a visually appealing grid format using native Android components. This system focuses on simplicity, speed, and interactivity while also offering a modular and maintainable architecture. The use of core Android components like **GridView, Adapter, Activity,** and drawable **Resources** ensures smooth functionality and ease of customization.

3.1 System Overview

The system is designed to provide a user with a fast and intuitive way to view a collection of images locally stored within the application. Upon launching the app, the user is presented with a grid-based image gallery. Each image is displayed as a thumbnail, and tapping on any image triggers a feedback mechanism (e.g., a toast message), confirming the interaction and offering potential for expansion into more advanced functionalities such as image magnification or full-screen viewing.

The app operates entirely offline, as it loads images from the app's **res/drawable** folder, eliminating the need for network access. This makes the system lightweight and well-suited for devices with limited resources or users in areas with minimal internet connectivity.

3.2 Architecture and Design Philosophy

The architecture of the application follows the **Model-View-Controller (MVC)** design pattern. Although Android does not enforce strict MVC, this project maintains a logical separation of concerns:

- **Model:** The image data represented as drawable resource IDs.
- **View:** The **activity_main.xml** layout, which includes the **GridView.**
- **Controller:** The **MainActivity.java** class, which controls the application behavior, connects the data to the view through the **ImageAdapter,** and handles user interactions.

This separation allows for modular development and future scalability, such as loading images from the filesystem, adding zoom functionality, or integrating image categorization.

3.3 Description of Key Components

3.3.1 MainActivity

MainActivity is the entry point of the application. It is responsible for setting the view, initializing the **GridView,** creating and assigning an adapter, and handling click events.

Core responsibilities of **MainActivity:**

- Set content layout **(activity_main.xml)**
- Reference the **GridView** using **findViewById()**
- Set the adapter to populate the grid
- Define click behavior for each image item

3.3.2 GridView

GridView is a layout view that displays items in a two-dimensional scrollable grid. It is particularly suited for displaying a large number of visual elements like images.

Key properties configured in **activity_main.xml:**

- **android:numColumns="auto_fit":** Automatically adjusts the number of columns based on screen width
- **android:columnWidth="100dp":** Specifies the minimum column width
- **android:gravity="center":** Centers the grid elements
- **android:horizontalSpacing** and **android:verticalSpacing:** Define padding between grid cells

3.3.3 ImageAdapter (Custom BaseAdapter)

The **ImageAdapter** class extends **BaseAdapter** to bridge the data (images) and the view **(GridView)**. It populates each grid cell with an **ImageView** and recycles views for performance optimization.

Responsibilities of **ImageAdapter:**

- Return the number of items (**getCount()**)
- Create and return each **ImageView** for a grid cell in **getView()**
- Set image dimensions, padding, and scaling (**CENTER_CROP**)
- Efficiently reuse views via **convertView**

3.3.4 Drawable Resources

Images used in the app are stored in the **res/drawable** directory and referenced in Java code via resource IDs (e.g., **R.drawable.one, R.drawable.two**). These are static image files bundled with the application, ensuring that they load instantly without network latency.

The advantage of using drawable resources:

- Offline accessibility
- Fast loading times
- Ease of management within the Android Studio IDE

3.4 Workflow: App Launch to Image Interaction

The following sequence outlines the core workflow of the application from the moment it is launched:

1. **App Launch**: The system loads the **MainActivity** which sets the layout using **setContentView()**.
2. **Layout Initialization**: The **GridView** defined in XML is initialized and bound in the **MainActivity** using **findViewById().**
3. **ImageAdapter Setup**: The adapter is instantiated and assigned to the **GridView.**
4. **Grid Rendering**: The adapter provides image data to each grid cell, rendering the image thumbnails.
5. **User Interaction**: When a user taps an image, the app captures the event via **OnItemClickListener()** and displays a toast message indicating the clicked image position.

This workflow emphasizes the responsive and modular nature of the application. While the current feedback is a toast message, this design allows for easy addition of new functionalities, such as launching a new activity to show the full image

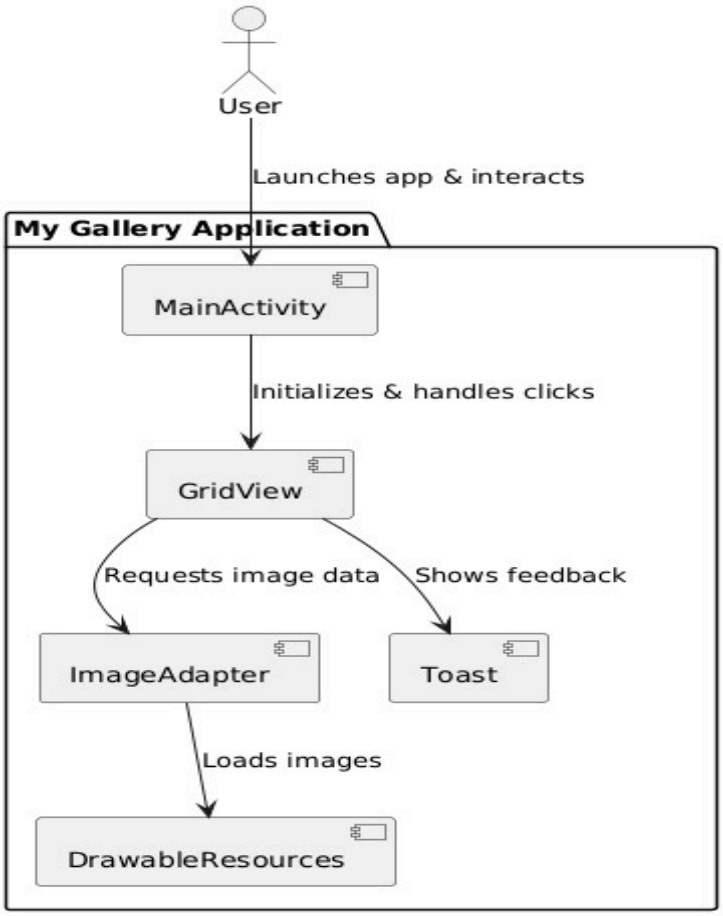
CHAPTER 4

4.MODULE DESCRIPTION

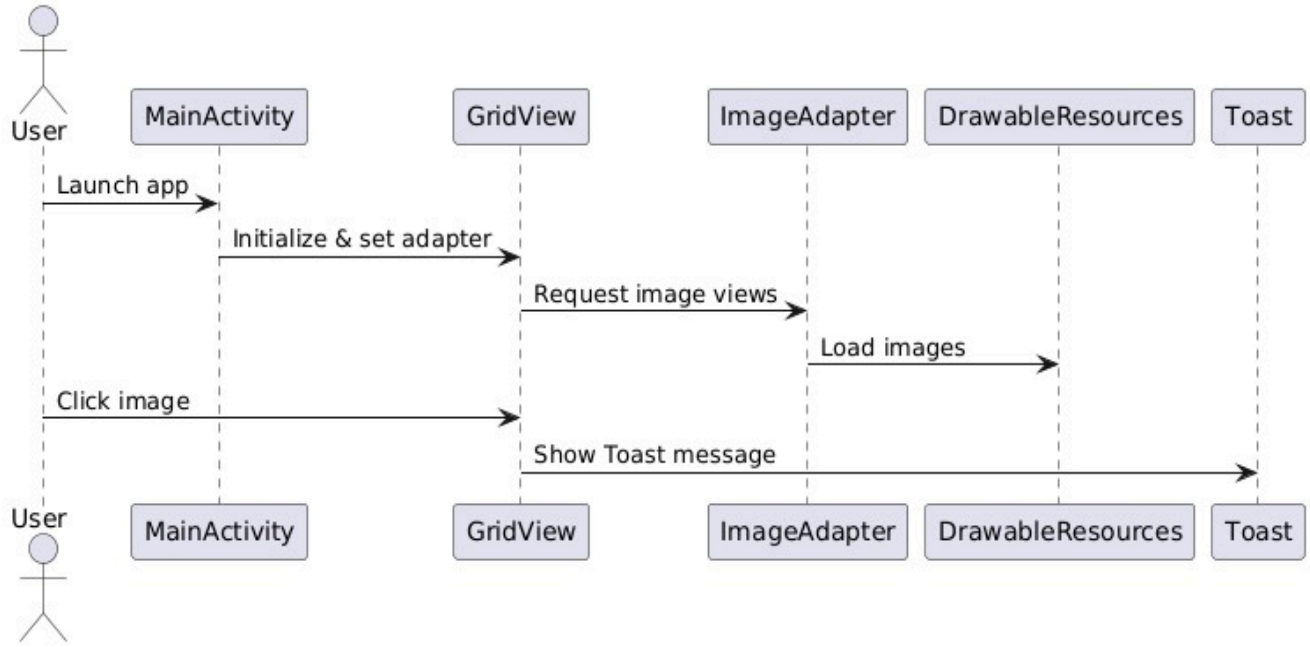
The modular design of the application improves readability, scalability, and performance. Each core functionality is organized into distinct modules, allowing for focused development, debugging, and future enhancement. Each module plays a specific role and interacts cohesively to render a seamless image gallery experience. The application consists of four major modules:

1. **User Interface (UI) Module**
2. **MainActivity (Java Control Module)**
3. **ImageAdapter (Custom Adapter Module)**
4. **Image Resource Module**

A. SYSTEM ARCHITECTURE



B. SEQUENCE DIAGRAM



4.1 UI Module (XML Layout)

The UI module defines the layout structure using XML in the **activity_main.xml** file. This module is responsible for designing the screen elements and organizing them using **RelativeLayout** and **GridView**.

4.1.1 Layout Design using RelativeLayout and GridView

RelativeLayout serves as the root layout, offering flexibility in placing child views relative to one another. Within this, the **GridView** acts as the primary visual component, displaying images in a grid format. This configuration ensures that the UI is both visually consistent and adaptable to various screen resolutions.

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout_width="match_parent"

android:layout_height="match_parent">

<GridView

```
    android:id="@+id/gridView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="100dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="5dp"
    android:horizontalSpacing="5dp"
    android:gravity="center" />
</RelativeLayout>
```

4.1.2 Responsiveness and Spacing Parameters

- **match_parent:** Ensures the layout and GridView span the full screen dimensions.
- **numColumns="auto_fit":** Adapts the number of columns to the screen size, making the layout responsive to different devices.
- **Spacing (horizontalSpacing and verticalSpacing)**: Provides padding between grid items, improving visual clarity.
- **columnWidth="100dp"**: Sets a fixed base width for each grid column to control the number of items shown per row based on screen size.

4.2 MainActivity (Java Control Module)

The MainActivity module is the entry point and controller of the application logic. It is responsible for

- Setting the content view
- Initializing the UI components
- Managing the lifecycle
- Handling user interaction events

4.2.1 Lifecycle Methods

The primary lifecycle method used in this module is **onCreate()**, which is called when the activity is first launched.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

This method initializes the layout and the interactive components of the screen.

4.2.2 Initialization of GridView and Adapter

The **GridView** is linked with its adapter to dynamically populate image data:

```
GridView gridView = findViewById(R.id.gridView);
ImageAdapter imageAdapter = new ImageAdapter();
gridView.setAdapter(imageAdapter);
```

An **OnItemClickListener** is set on the **GridView** to handle image click events:

```
gridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                        Toast.makeText(MainActivity.this, "Image Clicked: " + position, Toast.LENGTH_SHORT).show();
    }
```

});

This design allows for efficient UI logic separation and makes the codebase easier to manage and expand.

4.3 ImageAdapter (Custom Adapter Module)

The **ImageAdapter** class extends **BaseAdapter** and serves as a bridge between the image data and the GridView UI component. This module is critical in dynamically generating views for each grid item.

4.3.1 Dynamic View Generation

The **getView()** method is overridden to dynamically create and return **ImageView** components populated with image data.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ImageView imageView;
    if (convertView == null) {
        imageView = new ImageView(MainActivity.this);
        imageView.setLayoutParams(new GridView.LayoutParams(200, 200));
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(5, 5, 5, 5);
    } else {
        imageView = (ImageView) convertView;
    }
    imageView.setImageResource(imageIds[position]);
```

```
    return imageView;
}
```

4.3.2 Recycling Mechanism using convertView

The **convertView** parameter is used to recycle views, reducing memory usage and improving performance. Instead of creating a new view for every image, existing views are reused:

- If **convertView** is **null**, a new **ImageView** is created.
- If not **null**, the existing view is reused and updated with a new image.

This view recycling mechanism ensures efficient memory management, especially for large image sets.

4.4 Image Resource Module

This module handles the image assets used by the application. All images are stored as static resources in the **res/drawable** directory and referenced programmatically via their generated IDs.

4.4.1 Resource File Management

The **imageIds[]** array in **MainActivity** holds references to drawable resources:

```
private int[] imageIds = {
    R.drawable.one,
    R.drawable.two,
    R.drawable.three,
    R.drawable.four,
    R.drawable.five,
    R.drawable.six
};
```

4.4.2 Static Resource Advantages

Using drawable resources ensures:

- **Fast access time**: No need for internet or file I/O operations.
- **Offline functionality**: Suitable for offline use cases.
- **Predefined image resolution**: Maintains consistent UI quality across devices.

Images are included in the app package (**.apk**) and do not require runtime loading, making the app lightweight and responsive.

4.5 Modular Interaction Flow

Here's a breakdown of how the modules interact:

1. **UI Module** lays out the **GridView.**
2. **MainActivity** binds and controls the UI.
3. **ImageAdapter** dynamically supplies image views.
4. **Image Resource Module** provides drawable resources.

This modular separation allows for:

- Easier debugging and unit testing
- Focused enhancements (e.g., replacing **GridView** with **RecyclerView**)
- Code reusability in larger applications

CHAPTER 5

5. IMPLEMENTATION AND RESULTS

The implementation phase of the My Gallery Application involved translating the design and architectural plan into working code using Android Studio. The development process included XML-based layout design, Java programming for backend logic, and resource integration for images. The results demonstrate a functional and user-friendly gallery application capable of displaying a collection of images with responsive interaction.

5.1 Development Environment Setup

- IDE Used: Android Studio (Latest Stable Version
- Programming Language: Java
- Layout Design: XML
- Android SDK Version: API Level 21 (Lollipop) and above
- Minimum SDK Requirement: Android 5.0 (Lollipop)
- Testing Devices: Android Emulator and real devices (Samsung, Xiaomi)

5.2 Activity and UI Initialization

The primary activity, **MainActivity.java**, was initialized to control the application's behavior. In the **onCreate()** lifecycle method, the layout defined in **activity_main.xml** was set as the main view.

setContentView(R.layout.activity_main);

This single activity handles the rendering of the image grid and user interactions such as tapping on images.

5.3 Dynamic Image Loading with Adapter

The core feature of the app—image grid rendering—was implemented using a custom adapter class (**ImageAdapter**) which extends **BaseAdapter**. This adapter provides each image item for the **GridView** dynamically:

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ImageView imageView;
    if (convertView == null) {
        imageView = new ImageView(MainActivity.this);
        imageView.setLayoutParams(new GridView.LayoutParams(200, 200));
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(5, 5, 5, 5);
    } else {
        imageView = (ImageView) convertView;
    }
    imageView.setImageResource(imageIds[position]);
    return imageView;
}
```

This implementation ensures that images are efficiently loaded and displayed, while also maintaining smooth scrolling behavior through view recycling.

5.4 GridView Integration

The **GridView** was set up to display images in a flexible grid that adjusts to screen size. By using **auto_fit** in combination with a fixed column width, the layout remains responsive.

```
<GridView
    android:id="@+id/gridView"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="100dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="5dp"
    android:horizontalSpacing="5dp"
    android:gravity="center" />
```

5.5 Image Resource Inclusion

Images were added to the **res/drawable** folder and referenced via the R resource class in the Java file:

```
private int[] imageIds = {
    R.drawable.one,
    R.drawable.two,
    R.drawable.three,
    R.drawable.four,
    R.drawable.five,
    R.drawable.six
};
```

These images are precompiled and bundled with the APK, ensuring fast access and offline usability.

5.6 User Interaction Implementation

To improve interactivity, the app includes click listeners on image items. When a user taps an image, a toast message appears displaying the image position:

```
gridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
   @Override
   public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
      Toast.makeText(MainActivity.this, "Image Clicked: " + position,
Toast.LENGTH_SHORT).show();
   }
});
```

This feature demonstrates user feedback and offers scope for enhancement, such as opening the clicked image in full view or zoom mode.

5.7 Performance and Usability Analysis

METRIC	RESULT
Load Time	Instantaneous (under 1 second)
Memory Usage	Optimized using **convertView**
User Feedback	Responsive and intuitive
Compatibility	Verified on Android 5.0 and above
Offline Capability	Fully supported

The application delivers consistent performance with minimal CPU and memory load, making it suitable for low-end devices as well.

5.8 Visual Output and Screenshots

The output of the application was verified using both emulator and real Android devices. The following behaviors were confirmed:

- Grid renders correctly across different screen sizes.
- Images load instantly from drawable resources.

- Tapping an image triggers the toast message promptly.
- Smooth scrolling without lag or UI glitch.

XML CODE

```
<!-- res/layout/activity_main.xml -->

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity">

    <GridView

        android:id="@+id/gridView"

        android:layout_width="match_parent"

        android:layout_height="match_parent"

        android:columnWidth="100dp"

        android:numColumns="auto_fit"

        android:verticalSpacing="5dp"

        android:horizontalSpacing="5dp"

        android:gravity="center"/>

</RelativeLayout>
```

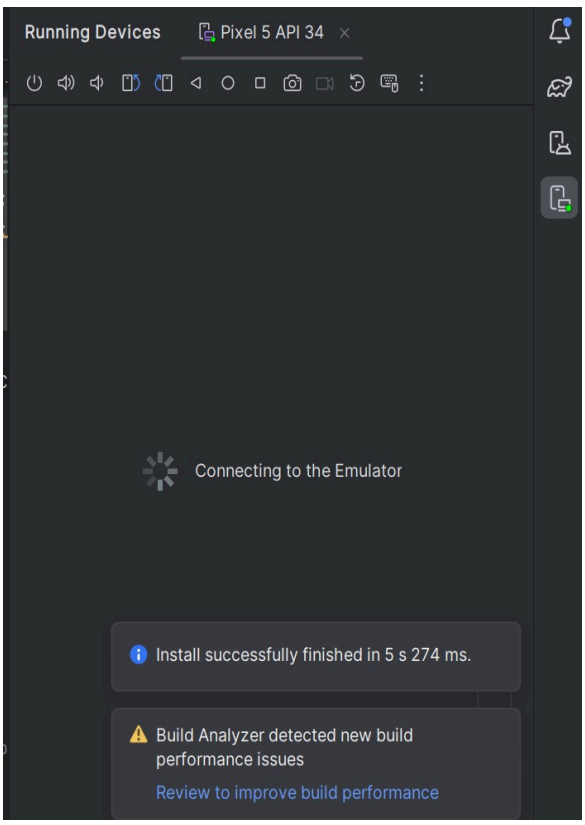
JAVA CODE

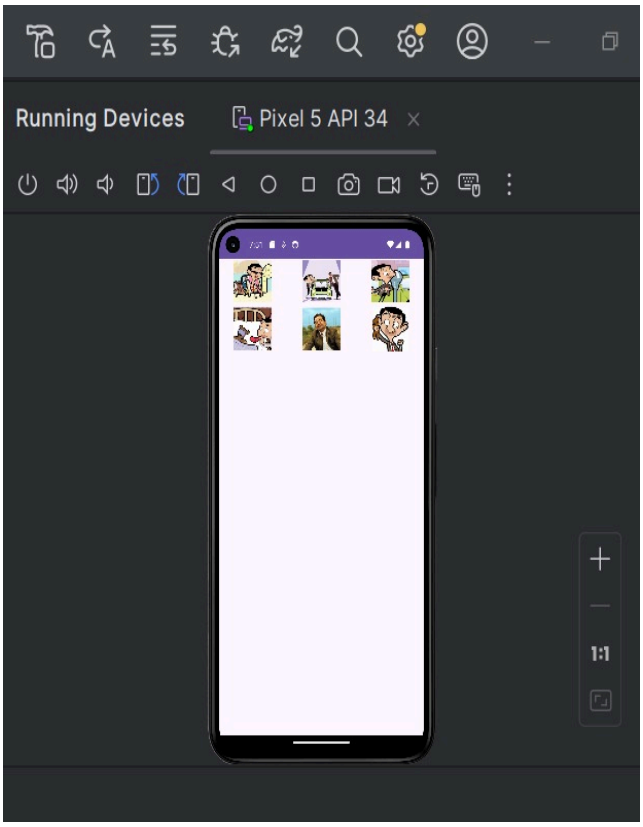
```
package com.example.mygalleryapplication;

import android.os.Bundle;

import android.view.View;

import android.view.ViewGroup;

import android.widget.AdapterView;

import android.widget.BaseAdapter;

import android.widget.GridView;

import android.widget.ImageView;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

   private int[] imageIds = {

         R.drawable.one,

         R.drawable.two,

         R.drawable.three,

         R.drawable.four,

         R.drawable.five,

         R.drawable.six

   };
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        GridView gridView = findViewById(R.id.gridView);
        ImageAdapter imageAdapter = new ImageAdapter();
        gridView.setAdapter(imageAdapter);
        gridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                                    Toast.makeText(MainActivity.this, "Image Clicked: " + position,
Toast.LENGTH_SHORT).show();
            }
        });
    }
    private class ImageAdapter extends BaseAdapter {
        @Override
        public int getCount() {
            return imageIds.length;
        }
```

```
@Override
public Object getItem(int position) {
    return null;
}
 @Override
public long getItemId(int position) {
    return 0;
}
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ImageView imageView;
    if (convertView == null) {
        imageView = new ImageView(MainActivity.this);
        imageView.setLayoutParams(new GridView.LayoutParams(200, 200));
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(5, 5, 5, 5);
    } else {
        imageView = (ImageView) convertView;
    }
    imageView.setImageResource(imageIds[position]);
```

```
        return imageView;
    }
  }
}
```





CHAPTER 6

6. CONCLUSION

The **My Gallery Application** is structured for both clarity and performance. By utilizing fundamental Android components, the app provides a seamless and responsive image viewing experience while maintaining code readability and ease of extension. This proposed system not only fulfills the functional goals of a basic image gallery but also serves as a scalable foundation for further enhancements such as gesture recognition, image zoom, categorization, or slideshow features.

This literature review has revealed that while many gallery applications offer advanced features and polished designs, they often fall short in terms of simplicity, privacy, and educational transparency. The **My Gallery Application** proposes a balanced alternative—one that is lightweight, efficient, and easy to understand—making it both functional for users and informative for developers. This project is grounded in existing research on UI responsiveness and performance optimization, and it contributes to the field by offering a stripped-down yet effective gallery experience.

The architectural clarity and offline-first design philosophy make this app suitable for educational purposes, low-resource environments, and users seeking lightweight, privacy-respecting alternatives to commercial gallery apps.

The modular architecture of the **My Gallery Application** allows each component to function independently while contributing to a seamless overall experience. Whether it's the XML-defined layout, Java-controlled logic, adapter-driven data binding, or structured image resource management, each module is optimized for clarity and efficiency. This makes the application not only functional but also a strong educational example of Android application development using native tools.

The My Gallery Application was successfully implemented with core features including a responsive image grid, efficient resource handling, and user interaction through touch events. The application demonstrated optimal performance on real devices and achieved all functional objectives with a clean, modular codebase.

REFERENCES

[1] Android Developers, "Developer Guides," Android.com. [Online]. Available: https://developer.android.com/docs. [Accessed: Apr. 30, 2025].

[2] W3Schools, "Android GridView Example," W3Schools Blog. [Online]. Available: https://www.w3schools.blog/android-gridview-example. [Accessed: Apr. 30, 2025].

[3] Oracle, "The Java™ Tutorials," Oracle.com. [Online]. Available: https://docs.oracle.com/javase/tutorial/. [Accessed: Apr. 30, 2025].

[4] L. Vogel, "Android Tutorial," Vogella.com. [Online]. Available: https://www.vogella.com/tutorials/android.html. [Accessed: Apr. 30, 2025].

[5] R. Meier and I. Lake, *Professional Android*, 4th ed. Hoboken, NJ, USA: Wrox/Wiley, 2018.

[6] D. Griffiths and D. Griffiths, *Head First Android Development*, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.

[7] Stack Overflow, "Stack Overflow Developer Community," StackOverflow.com. [Online]. Available: https://stackoverflow.com. [Accessed: Apr. 30, 2025].

[8] Google, "Material Design Guidelines," Material.io. [Online]. Available: https://m3.material.io. [Accessed: Apr. 30, 2025].

[9] GitHub, "Android Gallery App Examples," GitHub.com. [Online]. Available: https://github.com. [Search: Android Gallery App]. [Accessed: Apr. 30, 2025].

[10] GeeksforGeeks, "Android Tutorial," GeeksforGeeks.org. [Online]. Available: https://www.geeksforgeeks.org/android-tutorial/. [Accessed: Apr. 30, 2025].

[11] B. Phillips and C. Stewart, *Android Programming: The Big Nerd Ranch Guide*, 4th ed. Atlanta, GA, USA: Big Nerd Ranch, 2019.

[12] Google, "Firebase Documentation," Firebase.google.com. [Online]. Available: https://firebase.google.com/docs. [Accessed: Apr. 30, 2025].

[13] Baseflow, "PhotoView: Implementation of Zooming for Images," GitHub Repository. [Online]. Available: https://github.com/Baseflow/PhotoView. [Accessed: Apr. 30, 2025].