COMP 1786 Logbook Exercise 2

Basic Information

1.1 Student name	**Nguyen Duc Hieu**
1.2 Who did you work with? Note that for logbook exercises you are allowed to work with one other person as long as you give their name and login id and both contribute to the work.	**Name:** **Login id:**
1.3 Which Exercise is this? Tick as appropriate.	• Exercise 1 ☐ • Exercise 2 ☒ • Exercise 3 ☐
1.4 How well did you complete the exercise? Tick as appropriate.	• I tried but couldn't complete it ☐ • I did it but I feel I should have done better ☐ • I did everything that was asked ☐ • I did more than was asked for ☒
1.5 Briefly explain your answer to question 1.4. Without any explanation/justification, your scores will be deducted.	**A, Frontend** 1. **3D coverflow effect with custom PageTransformer:** Implements a 3D carousel effect using Y-axis rotation, dynamic scaling for the center image, Z translation to create depth, and camera distance tuning to achieve a 3D perspective. Multiple parameters are configurable. 2. **Reflection system with gradient fade:** Creates a vertically flipped reflection bitmap, applies an alpha gradient fade, adjusts brightness, uses RenderEffect blur on Android 12+, and a fallback blur implementation on older Android versions. 3. **Advanced material design with gradient overlays:** Uses a CardView with a sheen overlay (diagonal gradient for a glossy look), soft edge glow, and multi-layer gradient background. Multiple semi-transparent gradient layers are combined to add depth and a premium visual look. 4. **Dynamic elevation and blur for neighbor pages:** The center page has higher elevation, while neighbor pages have progressively lower elevation. Neighbor pages use dynamic RenderEffect blur, reduced color saturation (desaturation), and lower opacity to visually highlight the center image. 5. **Adaptive dimensions with responsive design:** Uses dimension resources for sizing. On small screens, automatically reduces reflection height ratio and blur intensity to optimize readability and layout. 6. **Memory-safe image loading with optimization:** Implements a custom bitmap decoding pipeline that inspects image size before decode, computes an optimal inSampleSize, caps maximum size to avoid OutOfMemoryError, and uses a multi-stage scaling strategy.

7. **Peek effect – showing partial adjacent pages:** Configures ViewPager2 padding so that users can see the edges of adjacent images, reinforcing the 3D carousel feel and hinting that they can swipe to see more content.

B, Interaction / UX

1. **Gesture swipe navigation:** Uses ViewPager2 to allow natural left/right swipe navigation between images with smooth animations, instead of relying only on previous/next buttons.
2. **Infinite loop scrolling:** The adapter returns a very large item count and maps positions back to real indices using modulo; it also starts from a middle position so the user can scroll infinitely in both directions.
3. **Touch to pause autoplay:** Implements an OnTouchListener to detect touch events and automatically pause autoplay when the user interacts with the screen, then resumes autoplay when the pager is idle.
4. **Smooth transitions with pre-rendered pages:** Sets offscreenPageLimit so neighboring pages are pre-rendered, ensuring smooth animations with no visible lag even when the user swipes quickly.
5. **Dynamic focus management for accessibility:** OnPageChangeCallback automatically marks the current page as focusable, clears focus on the previous page, and sets dynamic content descriptions per image. Fully supports screen readers.
6. **Visual feedback with scale and opacity transitions:** The center page is rendered with larger scale and full alpha, while neighbor pages have scale and opacity smoothly interpolated based on position. This clear visual hierarchy makes it obvious which page is currently active.
7. **Decorative elements marked for accessibility:** Reflection and overlay elements are explicitly marked as not important for accessibility so screen readers ignore decorative content and focus only on the actual image and description.

C, Backend / Architecture

1. **Modular architecture with separation of concerns:** The system is split into focused classes (Controller, Adapter, Transformer, etc.). Each class has a clear responsibility, making the code easier to maintain and extend.
2. **Configurable transformer system:** The CoverFlowPageTransformer constructor accepts multiple parameters so its behavior can be tuned without changing code. This effectively applies the Strategy pattern for page transform behavior.
3. **Tunable reflection system with builder-style API:** Provides setter methods to customize reflection behavior at runtime (e.g., gap, reflection height ratio, brightness). The host activity can flexibly adjust visual parameters.
4. **Memory management with proper view recycling:** Overrides onViewRecycled to clear ImageView drawables, reset RenderEffect, and ensure bitmap references are released. This helps prevent memory leaks.
5. **Multi-stage bitmap processing pipeline:** Implements a complete image-processing pipeline: Load → Scale → Crop → Vertical flip → Apply brightness → Apply gradient → Apply blur.
6. **Handler-based autoplay with lifecycle management:** Uses a Handler with postDelayed to implement autoplay via a recursive Runnable pattern, with proper cleanup in destroy() to align with the lifecycle.
7. **Multi-version API support with graceful degradation:** Checks API level to use RenderEffect blur on Android 12+, and falls back to a

	downscale/upscale blur approximation on older APIs. 8. **Resource optimization with size calculation:** Uses two-pass decoding: first to read source dimensions, then to compute optimal inSampleSize and decode with efficient settings. This reduces memory footprint. 9. **Gradle dependency management with version catalog:** Uses a Version Catalog to centralize dependency versions and explicitly declare dependencies such as ViewPager2, RecyclerView, etc. 10. **Proper exception handling and logging:** All bitmap operations are wrapped in try-catch blocks with specific exception handling and detailed logging (DEBUG/WARNING/ERROR), making it easier to debug and monitor runtime behavior.

2. Exercise answer

2.1 Screen shots demonstrating what you achieved.

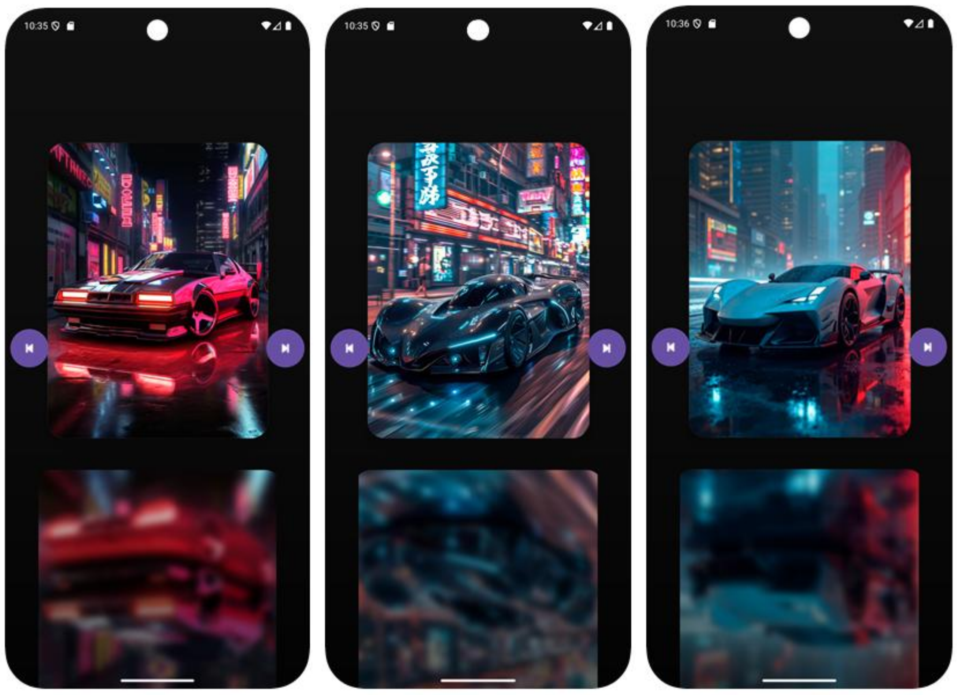

Figure 1 - Sample UI

2.2 Code that you wrote

CarouselController.java

```
package com.example.comp1786_logbook_ex2_viewimagesapplication;

import android.os.Handler;
import android.os.Looper;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.View;
import android.widget.ImageButton;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import androidx.viewpager2.widget.ViewPager2;

/**
 * Helper to wire a ViewPager2 as a coverflow carousel using ImagePagerAdapter and
CoverFlowPageTransformer.
```

```java
 */
public class CarouselController {
    private final ViewPager2 pager;
    private final Handler handler = new Handler(Looper.getMainLooper());
    private final Runnable autoplayRunnable;
    private final int autoplayDelayMs;
    private boolean autoplayFlag;

    public CarouselController(@NonNull ViewPager2 pager, @NonNull ImagePagerAdapter adapter,
@NonNull CoverFlowPageTransformer transformer,
                              boolean loop, boolean autoplay, int autoplayDelayMs, ImageButton
btnPrev, ImageButton btnNext) {
        this.pager = pager;
        this.autoplayFlag = autoplay;
        this.autoplayDelayMs = Math.max(1000, autoplayDelayMs);

        pager.setAdapter(adapter);
        pager.setOffscreenPageLimit(3);
        pager.setPageTransformer(transformer);

        // start near center so our adapter looping works (adapter repeats images)
        int imagesLen = adapter.getImagesLength();
        if (imagesLen > 0) {
            int centerStart = adapter.getItemCount() / 2;
            pager.setCurrentItem(centerStart - (centerStart % imagesLen), false);
        }

        // touch to pause autoplay; call performClick on ACTION_UP for accessibility
        pager.getChildAt(0).setOnTouchListener((v, event) -> {
            if (event.getAction() == MotionEvent.ACTION_DOWN) {
                if (autoplayFlag) stopAutoplay();
            } else if (event.getAction() == MotionEvent.ACTION_UP) {
                v.performClick();
            }
            return false;
        });

        pager.registerOnPageChangeCallback(new ViewPager2.OnPageChangeCallback() {
            @Override
            public void onPageSelected(int position) {
                super.onPageSelected(position);
                RecyclerView rv = (RecyclerView) pager.getChildAt(0);
                if (rv != null && rv.getLayoutManager() != null) {
                    View view = rv.getLayoutManager().findViewByPosition(position);
                    if (view != null) view.setFocusable(true);
                }
            }

            @Override
            public void onPageScrollStateChanged(int state) {
                super.onPageScrollStateChanged(state);
                if (state == ViewPager2.SCROLL_STATE_IDLE && autoplayFlag) scheduleAutoplay();
            }
        });

        if (btnPrev != null) btnPrev.setOnClickListener(v ->
pager.setCurrentItem(pager.getCurrentItem() - 1, true));
        if (btnNext != null) btnNext.setOnClickListener(v ->
pager.setCurrentItem(pager.getCurrentItem() + 1, true));

        pager.setOnKeyListener((v, keyCode, event) -> {
            if (event.getAction() != KeyEvent.ACTION_DOWN) return false;
            if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT) {
                pager.setCurrentItem(pager.getCurrentItem() - 1, true);
                return true;
            }
            if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT) {
                pager.setCurrentItem(pager.getCurrentItem() + 1, true);
                return true;
            }
            return false;
        });

        // initialize autoplayRunnable using a holder to avoid referencing the variable inside
its initializer
```

```java
        final Runnable[] holder = new Runnable[1];
        holder[0] = () -> {
            pager.setCurrentItem(pager.getCurrentItem() + 1, true);
            handler.postDelayed(holder[0], CarouselController.this.autoplayDelayMs);
        };
        this.autoplayRunnable = holder[0];

        if (autoplayFlag) scheduleAutoplay();
    }

    public void scheduleAutoplay() {
        stopAutoplay();
        handler.postDelayed(autoplayRunnable, autoplayDelayMs);
    }

    public void stopAutoplay() {
        handler.removeCallbacks(autoplayRunnable);
    }

    public void destroy() {
        stopAutoplay();
    }
}
```

CarouselPageTransformer.java

```java
package com.example.comp1786_logbook_ex2_viewimagesapplication;

import android.view.View;

import androidx.annotation.NonNull;
import androidx.viewpager2.widget.ViewPager2;

/**
 * CarouselPageTransformer creates a 3D carousel-like effect.
 * Center page: scale 1.0, alpha 1.0, rotationY 0
 * Adjacent pages: scaled down (0.85f), alpha reduced (0.7f), rotated along Y-axis
 * Transforms are interpolated smoothly during scroll.
 */
public class CarouselPageTransformer implements ViewPager2.PageTransformer {
    private static final float MIN_SCALE = 0.85f;
    private static final float MIN_ALPHA = 0.7f;
    private static final float MAX_ROTATION = 25f; // degrees around Y-axis

    @Override
    public void transformPage(@NonNull View page, float position) {
        // position: [-inf, +inf], centered page is 0, left is negative, right is positive
        float absPos = Math.abs(position);

        if (absPos >= 1) {
            // Page is off-screen to the left or right.
            page.setAlpha(MIN_ALPHA);
            page.setScaleX(MIN_SCALE);
            page.setScaleY(MIN_SCALE);
            page.setRotationY(position < 0 ? MAX_ROTATION : -MAX_ROTATION);
            page.setTranslationX(0f);
        } else {
            // Page is visible - interpolate scale, alpha and rotation
            // Scale interpolates between MIN_SCALE and 1.0
            float scale = MIN_SCALE + (1f - MIN_SCALE) * (1f - absPos);
            page.setScaleX(scale);
            page.setScaleY(scale);

            // Alpha interpolates between MIN_ALPHA and 1.0
            float alpha = MIN_ALPHA + (1f - MIN_ALPHA) * (1f - absPos);
            page.setAlpha(alpha);

            // RotationY: angle should be MAX_ROTATION * position (so left pages rotate
positive, right negative)
            float rotation = -position * MAX_ROTATION;
            page.setRotationY(rotation);

            // Slight translation to enhance perspective (push side pages slightly inward)
            float translationX = position * page.getWidth() * 0.2f;
            page.setTranslationX(translationX);
        }

        // Ensure pivot for rotation is vertical center
        page.setPivotY(page.getHeight() * 0.5f);
        page.setPivotX(page.getWidth() * 0.5f);
    }
}
```

CoverFlowPageTransformer.java

```
package com.example.comp1786_logbook_ex2_viewimagesapplication;

import android.graphics.ColorMatrix;
import android.graphics.ColorMatrixColorFilter;
import android.graphics.RenderEffect;
import android.graphics.Shader;
import android.os.Build;
import android.view.View;
import android.widget.ImageView;

import androidx.annotation.NonNull;
import androidx.viewpager2.widget.ViewPager2;

/**
 * Configurable 3D coverflow-style PageTransformer for ViewPager2.
 *
 * Usage:
 * viewPager.setOffscreenPageLimit(3);
 * viewPager.setPageTransformer(new CoverFlowPageTransformer(...));
 *
 * Requires pages to have an ImageView with id R.id.itemImage (matches pager_item layout).
 */
public class CoverFlowPageTransformer implements ViewPager2.PageTransformer {
    private final float rotationDegrees; // e.g., 30-40
    private final float neighborPeekPercent; // percent of width visible for neighbor (0.2 -
0.35)
    private final float neighborBlurPx; // 2-4
    private final float neighborOpacity; // 0.4-0.65
    private final float centerScaleMax; // 1.0-1.08
    private final int perspectiveDistance; // camera distance/perspective

    public CoverFlowPageTransformer() {
        this(30f, 0.28f, 3f, 0.55f, 1.04f, 1000);
    }

    public CoverFlowPageTransformer(float rotationDegrees, float neighborPeekPercent, float
neighborBlurPx, float neighborOpacity, float centerScaleMax, int perspectiveDistance) {
        this.rotationDegrees = rotationDegrees;
        this.neighborPeekPercent = neighborPeekPercent;
        this.neighborBlurPx = neighborBlurPx;
        this.neighborOpacity = neighborOpacity;
        this.centerScaleMax = centerScaleMax;
        this.perspectiveDistance = perspectiveDistance;
    }

    @Override
    public void transformPage(@NonNull View page, float position) {
        // position: 0 is centered page. -1 is one page to the left, +1 to the right
        float absPos = Math.abs(position);

        // set camera distance for perspective
        page.setCameraDistance(perspectiveDistance *
page.getResources().getDisplayMetrics().density);

        // compute visibility / peeking translation
        int pageWidth = page.getWidth();
        float peekOffset = pageWidth * neighborPeekPercent;

        // scale: center page scales up slightly
        float scale = 1f;
        if (absPos <= 1f) {
            float scaleFactor = 1f + (centerScaleMax - 1f) * (1f - absPos);
```

```java
            scale = scaleFactor;
        } else {
            scale = 1f;
        }

        // rotation on Y for 3D coverflow
        float rot = rotationDegrees * position;

        // translationX to make neighbors peek
        float translationX = -position * (peekOffset);

        // z ordering via translationZ (pages behind center have lower translationZ)
        float translationZ = (1f - absPos) * 200f; // front pages have higher z

        page.setTranslationX(translationX);
        page.setTranslationZ(translationZ);
        page.setScaleX(scale);
        page.setScaleY(scale);
        page.setRotationY(rot);

        // Opacity and blur/desaturation for neighbors
        ImageView img = page.findViewById(R.id.itemImage);
        if (img != null) {
            if (absPos >= 1f) {
                // fully off-center: hide
                page.setAlpha(0f);
                clearEffects(img);
            } else if (absPos > 0f) {
                // neighbor
                float op = 1f - (1f - neighborOpacity) * (1f - (1f - absPos));
                page.setAlpha(op);

                // apply blur on API31+
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                    float blur = neighborBlurPx * absPos; // reduce blur closer to center
                    img.setRenderEffect(RenderEffect.createBlurEffect(blur, blur,
Shader.TileMode.CLAMP));
                }

                // desaturate slightly
                float saturation = 1f - 0.18f * (1f - absPos); // ~10-25% desat depending on
position
                ColorMatrix cm = new ColorMatrix();
                cm.setSaturation(saturation);
                img.setColorFilter(new ColorMatrixColorFilter(cm));

                // remove drop shadow from neighbors if any
                page.setElevation(4f * (1f - absPos));
            } else {
                // center page
                page.setAlpha(1f);
                clearEffects(img);
                page.setElevation(12f);
            }
        } else {
            // fallback: just adjust alpha
            page.setAlpha(1f - Math.min(0.6f, absPos * 0.6f));
        }
    }

    private void clearEffects(ImageView img) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            img.setRenderEffect(null);
        }
        img.clearColorFilter();
    }
}
```

ImageAdapter.java

```java
package com.example.comp1786_logbook_ex2_viewimagesapplication;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

/**
 * Simple RecyclerView.Adapter for ViewPager2 that shows an ImageView per page.
 * This adapter expects there to be drawable resources in the project named image1..image5
 */
public class ImageAdapter extends RecyclerView.Adapter<ImageAdapter.VH> {
    private final Context context;
    private final int[] images;

    public ImageAdapter(Context context, int[] images) {
        this.context = context;
        this.images = images == null ? new int[0] : images;
    }

    @NonNull
    @Override
    public VH onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View v = LayoutInflater.from(context).inflate(R.layout.item_image, parent, false);
        return new VH(v);
    }

    @Override
    public void onBindViewHolder(@NonNull VH holder, int position) {
        if (images.length == 0) return;
        int idx = position % images.length;
        holder.image.setImageResource(images[idx]);
    }

    @Override
    public int getItemCount() {
        if (images.length <= 1) return images.length;
        // small loop multiplier to allow swiping both directions
        return images.length * 1000;
    }

    static class VH extends RecyclerView.ViewHolder {
        ImageView image;
        VH(@NonNull View itemView) {
            super(itemView);
            image = itemView.findViewById(R.id.imageView);
        }
    }
}
```

ImagePagerAdapter.java

```java
package com.example.comp1786_logbook_ex2_viewimagesapplication;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.ColorMatrix;
import android.graphics.ColorMatrixColorFilter;
import android.graphics.LinearGradient;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.PorterDuff;
import android.graphics.PorterDuffXfermode;
import android.graphics.Rect;
import android.graphics.Shader;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.os.Build;
import android.util.DisplayMetrics;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;

import androidx.annotation.NonNull;
import androidx.cardview.widget.CardView;
import androidx.recyclerview.widget.RecyclerView;

import android.content.res.Resources;

public class ImagePagerAdapter extends RecyclerView.Adapter<ImagePagerAdapter.VH> {
    private final int[] images;
    private final int cardWidthPx;

    // A large multiplier so ViewPager appears infinite; keep within int range
    private static final int LOOP_COUNT = 1000;

    // Reflection configurable properties (defaults chosen to match prompt ranges)
    private int gapPx = 10; // original gap in px
    // Revert to original: reflection height equals source height
    private float reflectionHeightRatio = 1.0f; // reflection height equals source height
    private float topOpacity = 1.00f; // original top opacity
    private float blurEndPx = 3.0f; // reduce base blur (sharper)
    private float brightness = 1.10f; // slight boost (>1 increases perceived intensity; may
clip to 255)

    public ImagePagerAdapter(int[] images) {
        this(images, -1);
    }

    public ImagePagerAdapter(int[] images, int cardWidthPx) {
        this.images = images;
        this.cardWidthPx = cardWidthPx; // if -1, we use the layout default dimen
    }

    // Optional setters so host can tune reflection behavior at runtime
    public void setGapPx(int gapPx) { this.gapPx = gapPx; }
    public void setReflectionHeightRatio(float ratio) { this.reflectionHeightRatio = ratio; }
    public void setTopOpacity(float topOpacity) { this.topOpacity = topOpacity; }
    public void setBlurEndPx(float endPx) { this.blurEndPx = endPx; }
    public void setBrightness(float brightness) { this.brightness = brightness; }

    @NonNull
    @Override
    public VH onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.pager_item, parent,
false);
        return new VH(v);
    }

    @Override
    public void onBindViewHolder(@NonNull VH holder, int position) {
        // Map the large adapter position to actual image index
        int realPos = position % images.length;

        // Determine safe target dimensions for the image to avoid huge bitmap allocation
        DisplayMetrics dm = holder.itemView.getResources().getDisplayMetrics();
        int screenW = dm.widthPixels;

        int targetW = 0, targetH = 0;
        if (cardWidthPx > 0) {
            targetW = cardWidthPx;
            targetH = (int) (cardWidthPx * (4f / 3f));
        } else if (holder.card.getWidth() > 0 && holder.card.getHeight() > 0) {
            targetW = holder.card.getWidth();
            targetH = holder.card.getHeight();
        } else if (holder.image.getWidth() > 0 && holder.image.getHeight() > 0) {
            targetW = holder.image.getWidth();
            targetH = holder.image.getHeight();
        } else {
            // fallback to a reasonable size (80% of screen width, 4:3)
            targetW = Math.max(100, (int) (screenW * 0.8f));
            targetH = (int) (targetW * (4f / 3f));
        }

        // Load a scaled bitmap from resources to fit target size (this prevents OOM for very
large drawables)
```

```java
        Bitmap srcBmp = null;
        try {
            srcBmp = loadScaledBitmapFromResource(holder.itemView.getResources(),
images[realPos], targetW, targetH);
            if (srcBmp != null) {
                Log.d("ImagePagerAdapter", "Loaded scaled srcBmp w=" + srcBmp.getWidth() + " h="
+ srcBmp.getHeight() + " for pos=" + realPos);
            } else {
                Log.d("ImagePagerAdapter", "Scaled decode returned null for pos=" + realPos);
            }
        } catch (Throwable t) {
            Log.w("ImagePagerAdapter", "loadScaledBitmapFromResource failed", t);
            srcBmp = null;
        }

        if (srcBmp != null) {
            holder.image.setImageBitmap(srcBmp);
        } else {
            // fallback to resource setter if decoding failed; this may still cause large
drawables but preserves behavior
            holder.image.setImageResource(images[realPos]);
        }

        // Ensure reflection is presentational for accessibility
        holder.reflection.setImportantForAccessibility(View.IMPORTANT_FOR_ACCESSIBILITY_NO);
        holder.reflection.setFocusable(false);
        holder.reflection.setClickable(false);

        // If a runtime card size was provided, apply it to the card so layout math matches
        if (cardWidthPx > 0) {
            ViewGroup.LayoutParams lp = holder.card.getLayoutParams();
            lp.width = cardWidthPx;
            // compute height preserving 3:4 aspect ratio (height = width * 4 / 3)
            lp.height = (int) (cardWidthPx * (4f / 3f));
            holder.card.setLayoutParams(lp);
        }

        // Make reflection follow clipping/rounded corners of source if set
        try {
            holder.reflection.setClipToOutline(holder.image.getClipToOutline());
        } catch (Throwable ignored) { }

        // Small-screen adaptive behavior: if screen width < 360dp, reduce reflection size/blur
        float screenDp = dm.widthPixels / dm.density;
        float effectiveReflectionRatio = reflectionHeightRatio;
        float effectiveBlurEnd = blurEndPx;
        if (screenDp < 360f) {
            effectiveReflectionRatio = Math.max(0.25f, reflectionHeightRatio * 0.6f); // reduce
height
            effectiveBlurEnd = Math.max(0f, blurEndPx * 0.5f); // reduce blur
        }

        // Create reflection from the scaled bitmap (if available) or fallback to drawable
handling
        try {
            if (srcBmp != null) {
                int reflectionHeight = Math.max(1, (int) (srcBmp.getHeight() *
effectiveReflectionRatio));
                Bitmap reflected = createReflectionBitmapFromBitmap(srcBmp, reflectionHeight,
topOpacity, brightness);
                if (reflected != null) {
                    Log.d("ImagePagerAdapter", "Reflection created w=" + reflected.getWidth() +
" h=" + reflected.getHeight() + " for pos=" + realPos);
                    holder.reflection.setImageBitmap(reflected);
                    holder.reflection.setTranslationY(gapPx);
                    holder.reflection.setScaleY(1f);

                    // Compute blur radius as 20% of reflection height (original behavior)
                    float blurRadiusPx = Math.max(0f, reflected.getHeight() * 0.2f);
                     // clamp to reasonable range
                     blurRadiusPx = Math.max(0f, Math.min(25f, blurRadiusPx));
                     Log.d("ImagePagerAdapter", "Applying blurRadiusPx=" + blurRadiusPx + " for
reflection pos=" + realPos);

                     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                         try {
holder.reflection.setRenderEffect(android.graphics.RenderEffect.createBlurEffect(blurRadiusPx,
blurRadiusPx, Shader.TileMode.CLAMP));
                         } catch (Throwable t) {
                             Log.e("ImagePagerAdapter", "RenderEffect blur failed", t);
                         }
                     } else {
                         // Fallback: a cheap blur approximation by downscaling and upscaling
the reflection bitmap
                         try {
                             Bitmap fallbackBlur = fastDownscaleBlur(reflected, blurRadiusPx);
                             if (fallbackBlur != null) {
                                 // replace the reflection bitmap with the blurred version and
recycle the original
                                 holder.reflection.setImageBitmap(fallbackBlur);
                                 if (!reflected.isRecycled()) reflected.recycle();
                                 reflected = fallbackBlur;
                             }
                         } catch (Throwable t) {
                             Log.w("ImagePagerAdapter", "fallback blur failed", t);
                         }
                     }

                     holder.image.post(() -> {
                        try {
                            holder.image.setPivotX(holder.image.getWidth() / 2f);
                            holder.image.setPivotY(holder.image.getHeight());
                            holder.reflection.setPivotX(holder.image.getWidth() / 2f);
                            holder.reflection.setPivotY(0f);

                            holder.reflection.setRotationX(-holder.image.getRotationX());
                            holder.reflection.setRotationY(holder.image.getRotationY());
                            holder.reflection.setRotation(holder.image.getRotation());
                            holder.reflection.setScaleX(holder.image.getScaleX());
                            holder.reflection.setScaleY(holder.image.getScaleY());
holder.reflection.setCameraDistance(holder.image.getCameraDistance());
                        } catch (Throwable t) {
                            Log.w("ImagePagerAdapter", "mirror transform setup failed", t);
                        }
                    });
                }
            } else {
                // Drawable fallback path (older behavior) - keep but flipped
                Drawable srcDrawable = holder.image.getDrawable();
                if (srcDrawable != null) {
                    // determine source dimensions (use drawable intrinsic if view size not
available yet)
                    int srcW = srcDrawable.getIntrinsicWidth() > 0 ?
srcDrawable.getIntrinsicWidth() : holder.image.getWidth();
                    int srcH = srcDrawable.getIntrinsicHeight() > 0 ?
srcDrawable.getIntrinsicHeight() : holder.image.getHeight();
                    if (srcW <= 0) srcW = holder.card.getWidth() > 0 ? holder.card.getWidth() :
400;
                    if (srcH <= 0) srcH = holder.card.getHeight() > 0 ? holder.card.getHeight()
: (int)(srcW * (4f/3f));

                    int reflectionHeight = Math.max(1, (int) (srcH * effectiveReflectionRatio));
                    Bitmap reflected = createReflectionBitmap(srcDrawable, srcW, srcH,
reflectionHeight, topOpacity, brightness, effectiveBlurEnd);
                    if (reflected != null) {
                        holder.reflection.setImageBitmap(reflected);
                        holder.reflection.setTranslationY(gapPx);
                        holder.reflection.setScaleY(1f);
                    } else {
                        holder.reflection.setImageDrawable(srcDrawable);
                        holder.reflection.setScaleY(-1f);
                        holder.reflection.setAlpha(0.35f);
                        // convert gap from dp-like value to pixels using display density
                        float density =
holder.itemView.getResources().getDisplayMetrics().density;
                        int gapPxPx = (int) (gapPx * density + 0.5f);
                        holder.reflection.setTranslationY(gapPxPx);
                    }
                }
            }
        } catch (Throwable t) {
            Log.e("ImagePagerAdapter", "reflection creation failed", t);
        }


holder.itemView.setContentDescription(holder.itemView.getContext().getString(R.string.image_cont
ent_description) + " " + (realPos + 1));
    }

    @Override
    public int getItemCount() {
        // Safely handle null images array
        if (images == null) return 0;
        // If only one image, don't loop
        if (images.length <= 1) return images.length;
        return images.length * LOOP_COUNT; // large number to simulate infinite scrolling
    }

    @Override
    public void onViewRecycled(@NonNull VH holder) {
        super.onViewRecycled(holder);
        try {
            // clear drawables to break references to large bitmaps
            holder.image.setImageDrawable(null);
            holder.reflection.setImageDrawable(null);
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                try {
                    holder.reflection.setRenderEffect(null);
                } catch (Throwable ignored) { }
            }
        } catch (Throwable t) {
            Log.w("ImagePagerAdapter", "onViewRecycled cleanup failed", t);
        }
    }

    public static final class VH extends RecyclerView.ViewHolder {
        ImageView image;
        ImageView reflection;
        CardView card;
        public VH(@NonNull View v) {
            super(v);
            image = v.findViewById(R.id.itemImage);
            reflection = v.findViewById(R.id.reflection);
            card = v.findViewById(R.id.card);
        }
    }

    // Expose number of distinct images so external controllers can compute starting indices
without accessing private field
    public int getImagesLength() {
        return images == null ? 0 : images.length;
    }

    // New: create reflection from Bitmap source (avoids creating extra huge intermediate
bitmaps)
    private Bitmap createReflectionBitmapFromBitmap(Bitmap srcBmp, int reflectionHeightPx, float
topOpacity, float brightness) {
        try {
            if (srcBmp == null) return null;
            int srcW = srcBmp.getWidth();
            int srcH = srcBmp.getHeight();
            int cropY = Math.max(0, srcH - reflectionHeightPx);
            Bitmap cropped = Bitmap.createBitmap(srcBmp, 0, cropY, Math.max(1, srcW),
Math.max(1, reflectionHeightPx));

            Matrix flip = new Matrix();
            // Flip vertically (up-down) so the reflection is inverted vertically (like a water
reflection)
            flip.preScale(1f, -1f);
            Bitmap flipped = Bitmap.createBitmap(cropped, 0, 0, cropped.getWidth(),
cropped.getHeight(), flip, true);

            Bitmap out = Bitmap.createBitmap(flipped.getWidth(), flipped.getHeight(),
```

```java
Bitmap.Config.ARGB_8888);
            Canvas canvas = new Canvas(out);
            Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.FILTER_BITMAP_FLAG);

            if (brightness != 1f) {
                ColorMatrix cm = new ColorMatrix(new float[] {
                        brightness, 0, 0, 0, 0,
                        0, brightness, 0, 0, 0,
                        0, 0, brightness, 0, 0,
                        0, 0, 0, 1, 0
                });
                paint.setColorFilter(new ColorMatrixColorFilter(cm));
            }

            canvas.drawBitmap(flipped, 0, 0, paint);

            Paint maskPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
            int height = out.getHeight();
            int topAlpha = Math.max(0, Math.min(255, (int) (topOpacity * 255f)));
            LinearGradient lg = new LinearGradient(0, 0, 0, height,
                    Color.argb(topAlpha, 255, 255, 255), Color.argb(0, 255, 255, 255),
Shader.TileMode.CLAMP);
            maskPaint.setShader(lg);
            maskPaint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.DST_IN));
            canvas.drawRect(new Rect(0,0,out.getWidth(), out.getHeight()), maskPaint);

            // Clean up intermediates
            if (!cropped.isRecycled()) cropped.recycle();
            if (!flipped.isRecycled()) flipped.recycle();

            return out;
        } catch (Throwable t) {
            Log.e("ImagePagerAdapter", "createReflectionBitmapFromBitmap failed", t);
            return null;
        }
    }

    // Fast cheap blur fallback: downscale then upscale to approximate blur. Not a true Gaussian
but inexpensive.
    private static Bitmap fastDownscaleBlur(Bitmap src, float blurPx) {
        try {
            if (src == null) return null;
            // Compute scale factor: higher blurPx -> smaller scale. Clamp between 0.05 and 0.5
to avoid degenerate sizes.
            float f = 1f / (1f + (blurPx / Math.max(1f, Math.min(src.getHeight(),
src.getWidth())) * 10f));
            f = Math.max(0.05f, Math.min(0.5f, f));
            int smallW = Math.max(1, (int) (src.getWidth() * f));
            int smallH = Math.max(1, (int) (src.getHeight() * f));
            Bitmap small = Bitmap.createScaledBitmap(src, smallW, smallH, true);
            Bitmap up = Bitmap.createScaledBitmap(small, src.getWidth(), src.getHeight(), true);
            if (!small.isRecycled()) small.recycle();
            return up;
        } catch (OutOfMemoryError oom) {
            Log.w("ImagePagerAdapter", "fastDownscaleBlur OOM", oom);
            return null;
        } catch (Throwable t) {
            Log.w("ImagePagerAdapter", "fastDownscaleBlur failed", t);
            return null;
        }
    }

    // Utility: generate a vertically flipped reflection bitmap from a drawable, preserving
alpha and corners.
    private Bitmap createReflectionBitmap(Drawable srcDrawable, int srcW, int srcH, int
reflectionHeightPx, float topOpacity, float brightness, float blurEndPx) {
        try {
            // Render drawable to bitmap at desired size
            Bitmap srcBmp = drawableToBitmap(srcDrawable, srcW, srcH);
            if (srcBmp == null) return null;

            // Crop the bottom portion of the source to use as reflection source
            int cropY = Math.max(0, srcH - reflectionHeightPx);
            Bitmap cropped = Bitmap.createBitmap(srcBmp, 0, cropY, srcW, reflectionHeightPx);
            // Flip vertically
            Matrix flip = new Matrix();
            // Flip vertically (up-down) to act like a vertical reflection
            flip.preScale(1f, -1f);
            Bitmap flipped = Bitmap.createBitmap(cropped, 0, 0, cropped.getWidth(),
cropped.getHeight(), flip, true);

            // Prepare output bitmap with alpha preserved
            Bitmap out = Bitmap.createBitmap(flipped.getWidth(), flipped.getHeight(),
Bitmap.Config.ARGB_8888);
            Canvas canvas = new Canvas(out);

            Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.FILTER_BITMAP_FLAG);

            // Apply brightness via ColorMatrix
            if (brightness != 1f) {
                ColorMatrix cm = new ColorMatrix(new float[] {
                        brightness, 0, 0, 0, 0,
                        0, brightness, 0, 0, 0,
                        0, 0, brightness, 0, 0,
                        0, 0, 0, 1, 0
                });
                paint.setColorFilter(new ColorMatrixColorFilter(cm));
            }

            // Draw the flipped image first
            canvas.drawBitmap(flipped, 0, 0, paint);

            // Create a vertical alpha gradient mask from topOpacity -> 0
            Paint maskPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
            int height = out.getHeight();
            int topAlpha = Math.max(0, Math.min(255, (int) (topOpacity * 255f)));
            LinearGradient lg = new LinearGradient(0, 0, 0, height,
                    Color.argb(topAlpha, 255, 255, 255), Color.argb(0, 255, 255, 255),
Shader.TileMode.CLAMP);
            maskPaint.setShader(lg);
            maskPaint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.DST_IN));
            canvas.drawRect(new Rect(0,0,out.getWidth(), out.getHeight()), maskPaint);

            // Note: RenderEffect blur is applied at the view level in onBindViewHolder for
API31+.

            // Clean up intermediates
            cropped.recycle();
            flipped.recycle();
            if (!srcBmp.isRecycled()) srcBmp.recycle();

            return out;
        } catch (Throwable t) {
            Log.e("ImagePagerAdapter", "createReflectionBitmap failed", t);
            return null;
        }
    }

    private static Bitmap drawableToBitmap(Drawable drawable, int reqW, int reqH) {
        if (drawable instanceof BitmapDrawable) {
            Bitmap bmp = ((BitmapDrawable) drawable).getBitmap();
            if (bmp.getWidth() == reqW && bmp.getHeight() == reqH) return bmp;
        }
        try {
            Bitmap bitmap = Bitmap.createBitmap(Math.max(1, reqW), Math.max(1, reqH),
Bitmap.Config.ARGB_8888);
            Canvas canvas = new Canvas(bitmap);
            drawable.setBounds(0, 0, canvas.getWidth(), canvas.getHeight());
            drawable.draw(canvas);
            return bitmap;
        } catch (OutOfMemoryError oom) {
            Log.e("ImagePagerAdapter", "drawableToBitmap OOM", oom);
            return null;
        }
    }

    // New: load a downsampled bitmap from resources to target size safely.
    private static Bitmap loadScaledBitmapFromResource(Resources res, int resId, int targetW,
int targetH) {
        if (res == null) return null;
        BitmapFactory.Options opts = new BitmapFactory.Options();
        opts.inJustDecodeBounds = true;
        BitmapFactory.decodeResource(res, resId, opts);
        int srcW = opts.outWidth;
        int srcH = opts.outHeight;
        if (srcW <= 0 || srcH <= 0) return null;

        // Safety cap: avoid creating bitmaps with extremely large pixel counts.
        final long MAX_PIXELS = 4_000_000; // ~4MP -> ~16MB ARGB_8888
        long srcPixels = (long) srcW * (long) srcH;
        float scaleForPixels = srcPixels > MAX_PIXELS ? (float) Math.sqrt((double) MAX_PIXELS /
(double) srcPixels) : 1f;

        // Desired scale to fit target dimensions
        float scaleW = targetW > 0 ? (float) targetW / (float) srcW : 1f;
        float scaleH = targetH > 0 ? (float) targetH / (float) srcH : 1f;
        float desiredScale = Math.min(scaleW, scaleH);

        // Use the smaller of desiredScale and pixel-safety scale
        float finalScale = Math.min(desiredScale, scaleForPixels);
        if (finalScale <= 0f) finalScale = 0.1f;

        // Calculate inSampleSize (power of 2) for efficient decoding
        int inSampleSize = 1;
        while ((srcW / inSampleSize) * (srcH / inSampleSize) > MAX_PIXELS) {
            inSampleSize *= 2;
        }

        // Also try to make decoded size >= target to avoid upscaling after decode
        int approxW = Math.max(1, srcW / inSampleSize);
        int approxH = Math.max(1, srcH / inSampleSize);

        opts = new BitmapFactory.Options();
        opts.inSampleSize = inSampleSize;
        opts.inPreferredConfig = Bitmap.Config.ARGB_8888;

        Bitmap decoded = null;
        try {
            decoded = BitmapFactory.decodeResource(res, resId, opts);
            if (decoded == null) return null;

            // If decoded is still larger than final desired, scale down preserving aspect
            int finalW = Math.max(1, (int) Math.round(srcW * finalScale));
            int finalH = Math.max(1, (int) Math.round(srcH * finalScale));

            if (decoded.getWidth() != finalW || decoded.getHeight() != finalH) {
                Bitmap scaled = Bitmap.createScaledBitmap(decoded, finalW, finalH, true);
                if (scaled != decoded) decoded.recycle();
                decoded = scaled;
            }
            return decoded;
        } catch (OutOfMemoryError oom) {
            Log.e("ImagePagerAdapter", "decodeResource OOM", oom);
            if (decoded != null && !decoded.isRecycled()) decoded.recycle();
            return null;
        }
    }
}
```

MainActivity.java

```
package com.example.comp1786_logbook_ex2_viewimagesapplication;

import android.os.Bundle;
import android.view.View;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.RecyclerView;
import androidx.viewpager2.widget.ViewPager2;

import com.google.android.material.floatingactionbutton.FloatingActionButton;

/**
```

```java
 * MainActivity that uses the project's existing ImagePagerAdapter (which creates reflections,
sizing, etc.)
 * and the CoverFlowPageTransformer to preserve the original 3D effect and image sizing.
 * The toolbar was removed per user request.
 */
public class MainActivity extends AppCompatActivity {

    private ViewPager2 viewPager;
    private FloatingActionButton btnBackward;
    private FloatingActionButton btnForward;

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

        viewPager = findViewById(R.id.viewPager);
        btnBackward = findViewById(R.id.btnBackward);
        btnForward = findViewById(R.id.btnForward);

        // Use all drawable images present in res/drawable (image1..image9)
        final int[] images = new int[] {
                R.drawable.image1,
                R.drawable.image2,
                R.drawable.image3,
                R.drawable.image4,
                R.drawable.image5,
                R.drawable.image6,
                R.drawable.image7,
                R.drawable.image8,
                R.drawable.image9
        };

        // Use the existing, more advanced adapter in this project which preserves image sizing
and reflections
        int cardWidthPx = getResources().getDimensionPixelSize(R.dimen.card_width);
        ImagePagerAdapter adapter = new ImagePagerAdapter(images, cardWidthPx);
        viewPager.setAdapter(adapter);

        // Preserve several offscreen pages for smoothness
        viewPager.setOffscreenPageLimit(3);

        // Allow adjacent pages to peek
        int peek = getResources().getDimensionPixelSize(R.dimen.pager_peek);
        viewPager.setPadding(peek, 0, peek, 0);
        RecyclerView rv = (RecyclerView) viewPager.getChildAt(0);
        if (rv != null) {
            rv.setClipToPadding(false);
            rv.setClipChildren(false);
            rv.setOverScrollMode(View.OVER_SCROLL_NEVER);
        }

        // Use the original 3D coverflow transformer available in the project
        viewPager.setPageTransformer(new CoverFlowPageTransformer());

        // Set up button click listeners
        btnBackward.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int currentItem = viewPager.getCurrentItem();
                if (currentItem > 0) {
                    viewPager.setCurrentItem(currentItem - 1, true);
                }
            }
        });

        btnForward.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int currentItem = viewPager.getCurrentItem();
                if (adapter != null && currentItem < adapter.getItemCount() - 1) {
                    viewPager.setCurrentItem(currentItem + 1, true);
                }
            }
        });
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/bg_gradient">

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        android:clipToPadding="false"
        android:clipChildren="false"
        android:overScrollMode="never" />

    <!-- Bottom dot indicators (optional; kept hidden by default) -->
    <LinearLayout
        android:id="@+id/dotsContainer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_gravity="bottom|center"
        android:layout_marginBottom="28dp"
        android:padding="6dp"
        android:gravity="center"
        android:visibility="gone" />

    <!-- Backward button (left side) -->
    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/btnBackward"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|start"
        android:layout_marginStart="8dp"
        android:contentDescription="@string/prev_button_description"
        android:src="@android:drawable/ic_media_previous"
        app:backgroundTint="@color/md_primary"
        app:tint="@color/md_on_primary"
        app:fabSize="normal"
        app:elevation="0dp" />

    <!-- Forward button (right side) -->
    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/btnForward"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|end"
        android:layout_marginEnd="8dp"
        android:contentDescription="@string/next_button_description"
        android:src="@android:drawable/ic_media_next"
        app:backgroundTint="@color/md_primary"
        app:tint="@color/md_on_primary"
        app:fabSize="normal"
        app:elevation="0dp" />

</FrameLayout>
```

Item_image.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:adjustViewBounds="true"
        android:scaleType="centerCrop"
        android:contentDescription="@string/image_content_description"/>

</FrameLayout>
```

Paper_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="8dp">

    <!-- Card container enforcing 3:4 aspect ratio; centered horizontally and placed near top so
reflection can sit under it -->

    <!-- Reflection immediately beneath the CardView so it aligns with the bottom edge; gap is
configurable via layout_marginTop -->

    <androidx.cardview.widget.CardView
        android:id="@+id/card"
        android:layout_width="@dimen/card_width"
        android:layout_height="@dimen/card_height"
        android:layout_marginTop="176dp"
        android:clipChildren="false"
        android:clipToPadding="false"
        android:foregroundTintMode="src_over"
        app:cardCornerRadius="24dp"
        app:cardElevation="12dp"
        app:cardUseCompatPadding="true"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.491"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_margin="0dp"
            android:padding="0dp">

            <ImageView
                android:id="@+id/itemImage"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:adjustViewBounds="true"
                android:background="@drawable/round_rect"
                android:clipToOutline="true"
                android:contentDescription="@string/image_content_description"
                android:scaleType="centerCrop" />

            <!-- Sheen overlay: subtle glossy surface using a gradient with low alpha -->
            <View
                android:id="@+id/sheen"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:alpha="0.28"
                android:background="@drawable/sheen_overlay"
                android:importantForAccessibility="no" />

            <!-- Soft glow edge (very faint) to emphasize center when scaled up -->
            <View
                android:id="@+id/glow"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:alpha="0.0"
                android:background="@drawable/soft_edge_glow"
                android:importantForAccessibility="no" />

        </FrameLayout>
    </androidx.cardview.widget.CardView>
    <ImageView
        android:id="@+id/reflection"
        android:layout_width="@dimen/card_width"
        android:layout_height="@dimen/card_height"
        android:layout_marginTop="16dp"
        android:adjustViewBounds="true"
        android:alpha="1.00"
        android:background="@drawable/round_rect"
        android:clipToOutline="true"
        android:contentDescription="@null"
        android:importantForAccessibility="no"
        android:scaleType="centerCrop"
        android:visibility="visible"
        app:layout_constraintEnd_toEndOf="@id/card"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="@id/card"
        app:layout_constraintTop_toBottomOf="@id/card" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Justification

1. Overall architecture

The app is a **single-Activity image viewer** with a **3D coverflow-style carousel** based on ViewPager2:

- Entry point: MainActivity
- Core UI: ViewPager2 showing images with a 3D coverflow effect + bottom reflection
- Navigation: previous/next via two FloatingActionButtons, plus swipe
- Image pipeline:
    - ImagePagerAdapter (advanced adapter: reflection + scaling + memory safety)
    - CoverFlowPageTransformer (3D animation and neighbor effects)

There are also **reusable helpers** that aren't wired in MainActivity but are part of the project's architecture:

- CarouselController – generic controller for ViewPager2 autoplay + looping + keyboard nav
- CarouselPageTransformer – simpler 3D carousel transformer (alternate effect)
- ImageAdapter – simple adapter for basic image slides

So you effectively have a **small UI framework for image carousels**: multiple transformers + multiple adapters + a controller, with MainActivity using the "premium" stack (ImagePagerAdapter + CoverFlowPageTransformer).

2. MainActivity.java

MainActivity is a thin composition layer:

Aspect	Description
Layout	Uses activity_main.xml with a full-screen ViewPager2 and two FloatingActionButtons (btnBackward, btnForward) over a gradient background
Images	Hard-coded drawable array image1…image9
Adapter	Instantiates ImagePagerAdapter(images, cardWidthPx), where cardWidthPx is read from R.dimen.card_width to maintain consistent 3:4 aspect ratio via pager_item

Aspect	Description
ViewPager2	Sets adapter, setOffscreenPageLimit(3) for smooth scrolling, paddings to show peeking neighbors, and disables overscroll on the underlying RecyclerView child
Transformer	viewPager.setPageTransformer(new CoverFlowPageTransformer()) to apply the 3D coverflow effect on each page
Navigation	btnBackward and btnForward move the current item by −1 / +1 with animation, bounds-checked against adapter.getItemCount()

In other words: MainActivity just configures the carousel and delegates all heavy lifting to ImagePagerAdapter + CoverFlowPageTransformer.

3. ImagePagerAdapter.java – advanced pager with reflections

This is the **core visual engine**. It extends RecyclerView.Adapter and is designed for ViewPager2, supporting:

- Infinite-feeling scrolling via multiplying image count by LOOP_COUNT
- Dynamic reflection under each card
- Memory-efficient bitmap loading
- Adaptive reflection behavior for small screens

Structure & state

Field	Role
int[] images	Drawable IDs representing the image set
int cardWidthPx	Target card width; used to compute height (3:4) if provided
Config fields	gapPx, reflectionHeightRatio, topOpacity, blurEndPx, brightness – reflection and visual tuning knobs
LOOP_COUNT	Large multiplier for pseudo-infinite scrolling (itemCount = images.length * LOOP_COUNT)

Public setters (setGapPx, setReflectionHeightRatio, etc.) allow runtime tuning of reflection look & feel.

ViewHolder

VH holds:

- ImageView image – main image (R.id.itemImage)
- ImageView reflection – reflection image (R.id.reflection)
- CardView card – outer container enforcing aspect ratio and shadow

Layout is defined in pager_item.xml.

onBindViewHolder flow

High-level pipeline:

1. **Map adapter position to real image index**
2. int realPos = position % images.length;
3. **Determine target dimensions** (card width/height or fallback based on screen width) to avoid allocating huge bitmaps.
4. **Load a scaled bitmap** from resources via loadScaledBitmapFromResource(...):
   - Uses inJustDecodeBounds + inSampleSize + size caps to prevent OOM
   - Optionally downscales further to meet a pixel cap (~4MP)
5. **Set main image**:
   - If scaled bitmap is available → holder.image.setImageBitmap(srcBmp)
   - Otherwise fallback to setImageResource(images[realPos])
6. **Accessibility**:
   - reflection marked non-accessible (IMPORTANT_FOR_ACCESSIBILITY_NO)
   - itemView.setContentDescription(...) includes “image N” text for screen readers
7. **Apply card width / aspect**:
   - If cardWidthPx > 0, sets card layout width to that and height to cardWidthPx * 4/3
8. **Small-screen adaptation**:
   - If screen width in dp < 360, reflection height and blur are reduced to keep visuals readable on small devices
9. **Create and apply reflection**:

   There are two paths:

Path	Trigger	Behavior
Bitmap path	srcBmp != null	Crops bottom part of the bitmap, flips vertically, applies brightness + vertical alpha gradient, generates reflection bitmap; then applies blur (RenderEffect on API 31+ or fastDownscaleBlur fallback)
Drawable fallback	srcBmp == null but image.getDrawable() != null	Renders drawable to bitmap based on dimensions, then similar reflection creation via createReflectionBitmap

   After reflection is created, it is assigned to holder.reflection with a vertical gap (setTranslationY(gapPx)) and scale. Pivots/rotations/scale/camera distance are synchronized with the main image in a post callback so reflection follows any 3D transforms applied by the transformer.

10. **Cleanup on recycle** (onViewRecycled):
    - Clears both image and reflection drawables
    - Clears RenderEffect on API 31+
    - Minimizes lingering references to heavy bitmaps

Helper methods

Key utilities:

Method	Purpose

Method	Purpose
getImagesLength()	Exposes distinct image count to external controllers (e.g. for centering)
createReflectionBitmapFromBitmap(...)	Builds reflection from a Bitmap: crop bottom, vertical flip, brightness adjust, alpha gradient, and cleanup intermediates
createReflectionBitmap(...)	Similar, but starting from a Drawable and explicit sizes
fastDownscaleBlur(Bitmap src, float blurPx)	Cheap blur approximation: downscale and upscale to simulate blur
drawableToBitmap(Drawable, int, int)	Renders Drawable to a bitmap of requested size
loadScaledBitmapFromResource(...)	Size-aware, OOM-safe resource decoding for images

Net effect: ImagePagerAdapter is responsible for **high-quality visuals (reflection, gradient, slight brightness boost)** and **safe memory usage**.

4. CoverFlowPageTransformer.java – 3D coverflow effect

CoverFlowPageTransformer implements ViewPager2.PageTransformer with configurable parameters:

Parameter	Meaning
rotationDegrees	Y-axis rotation magnitude for side pages (e.g. ±30°)
neighborPeekPercent	Percentage of page width that neighbors “peek” into the viewport
neighborBlurPx	Base blur radius for neighbor images
neighborOpacity	Opacity of side pages (e.g. 0.55)
centerScaleMax	Max scale factor for the centered page (e.g. 1.04)
perspectiveDistance	Camera distance to influence perspective depth

Core transform logic in transformPage(View page, float position):

- position ranges: 0 = center, −1 / +1 = one page left/right
- Sets camera distance per device-density
- Computes:

Property	Behavior
scale	Center page slightly scaled up; decreases with abs(position)

Property	Behavior
rotationY	rotationDegrees * position for 3D coverflow tilt
translationX	Based on neighborPeekPercent so neighbors peek
translationZ	Center has highest Z, neighbors go “behind”

- If page’s layout contains ImageView with R.id.itemImage:
  - For fully off-center (absPos ≥ 1): set alpha=0, clear effects
  - For neighbors (0 < absPos < 1):
    - Adjust page alpha toward neighborOpacity
    - On API 31+: apply blur using RenderEffect.createBlurEffect, decreasing blur as page gets closer to center
    - Slight desaturation using ColorMatrix (simulate focus on center image)
    - Lower elevation for side pages
  - For center page: alpha=1, clear blur and color filter, set high elevation

If itemImage is not found, it falls back to a simpler alpha-only effect. This transformer is what creates the **3D coverflow look with center focus, blurred/desaturated neighbors, and depth**.

5. CarouselController.java – optional autoplay + keyboard controller

CarouselController is a **helper for ViewPager2** that adds:

- Centered start index for pseudo-infinite adapters
- Autoplay with configurable delay
- Touch to pause
- Keyboard navigation (DPAD)
- Previous/Next button wiring

Constructor signature:

```
CarouselController(ViewPager2 pager,
            ImagePagerAdapter adapter,
            CoverFlowPageTransformer transformer,
            boolean loop,
            boolean autoplay,
            int autoplayDelayMs,
            ImageButton btnPrev,
            ImageButton btnNext)
```

Key aspects:

Feature	Behavior
Adapter wiring	Sets adapter, offscreen page limit, transformer
Initial index	Starts close to middle of adapter’s pseudo-infinite range so modulo-based looping feels natural
Autoplay	Uses Handler and a Runnable that increments current item every autoplayDelayMs

Feature	Behavior
Touch handling	On ACTION_DOWN, pauses autoplay; on ACTION_UP, calls performClick for accessibility
PageChange callback	Ensures focused page is focusable (for D-pad / accessibility focus)
Keyboard nav	KEYCODE_DPAD_LEFT and KEYCODE_DPAD_RIGHT move pages backwards/forwards
API methods	scheduleAutoplay(), stopAutoplay(), destroy() to manage lifecycle from host Activity

In this submission, MainActivity doesn't use CarouselController directly, but the class is ready to be plugged in as a more advanced controller.

6. CarouselPageTransformer.java

This is an alternative PageTransformer with a simpler effect than CoverFlowPageTransformer:

- Uses constants MIN_SCALE, MIN_ALPHA, MAX_ROTATION
- For pages off-screen (absPos >= 1), applies a fixed small scale, low alpha, and rotationY
- For visible pages (absPos < 1), interpolates:
  - scale between MIN_SCALE and 1.0
  - alpha between MIN_ALPHA and 1.0
  - rotationY between ±MAX_ROTATION based on position
- Sets pivots to center of the page to rotate around its center

It gives a straightforward "carousel zoom + Y-tilt" feel, versus the more complex coverflow behavior.

7. ImageAdapter.java – simple adapter option

ImageAdapter is a minimal RecyclerView.Adapter for ViewPager2 with a single full-screen ImageView per page, using item_image.xml.

Highlights:

- images is an int[] of drawable IDs
- onCreateViewHolder inflates item_image
- onBindViewHolder sets imageView.setImageResource(images[idx]), where idx = position % images.length
- getItemCount returns:
  - images.length for 0–1 images
  - images.length * 1000 to simulate looping when there are multiple images

This is a simpler alternative to ImagePagerAdapter when you don't need reflections or advanced scaling.

8. Layouts

activity_main.xml

Root FrameLayout with:

Element	Description
ViewPager2 viewPager	Full-screen, horizontal orientation, clipToPadding=false, clipChildren=false, overScrollMode="never"
LinearLayout dotsContainer	Optional bottom dot indicator container (currently visibility="gone")
FloatingActionButton btnBackward	Left side, vertically centered, media-previous icon, tinted with md_primary / md_on_primary
FloatingActionButton btnForward	Right side, vertically centered, media-next icon, same tinting

Background is a gradient drawable (bg_gradient).

item_image.xml

A simple full-screen single-image item:

- FrameLayout root
- Child ImageView imageView:
    - match_parent width/height
    - adjustViewBounds="true"
    - scaleType="centerCrop"
    - contentDescription tied to @string/image_content_description

Used with ImageAdapter.

pager_item.xml (named Paper_item.xml in text, but layout id is pager_item)

This is the **card + reflection layout** used by ImagePagerAdapter:

Element	Description
Root ConstraintLayout	Includes padding and positions card + reflection
CardView card	Size defined by @dimen/card_width / @dimen/card_height, with 24dp corner radius, 12dp elevation, centered near top
Inside card	FrameLayout containing: itemImage (rounded, centerCrop), a sheen View (light glossy overlay via gradient), and a glow View (soft edge glow)
ImageView reflection	Same width/height as card, positioned directly below it; rounded background, alpha 1.0, clipToOutline=true; serves as the reflection canvas filled by ImagePagerAdapter