VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnana Sangama", Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V. Puram, Bengaluru-560 004

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MOBILE APPLICATION DEVELOPMENT
MINI PROJECT (18CSMP68)

"PAINT APPLICATION"

Submitted By

1BI20CS048	**CHIRAYU V GANGADKAR**

for the academic year 2022-2023

Under the guidance of

Dr. Savitha S K	**Prof. Mahalakshmi C V**
Associate Professor	Assistant Professor

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“Jnana Sangama”, Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V. Puram, Bengaluru-560 004

Department of Computer Science & Engineering

Certificate

This is to certify that the implementation of **Mobile Application Development MINI PROJECT (18CSMP68)** entitled **“ PAINT APPLICATION ”** has been successfully completed by

1BI20CS048 **CHIRAYU V GANGADKAR**

of VI semester B.E. for the partial fulfillment of the requirements for the Bachelor's degree in **Computer Science & Engineering** of the **Visvesvaraya Technological University** during the academic year **2022-2023**.

Lab In charges:

Dr. Savitha S K	**Prof. Mahalakshmi C V**	**Dr. Girija J.**
Associate Professor	Assistant Professor	Professor and Head
Dept. of CS&E, BIT	Dept. of CS&E, BIT	Department of CS&E
		Bangalore Institute of Technology

Examiners: 1) 2)

ACKNOWLEDGEMENT

The knowledge & satisfaction that accompany the successful completion of any task would be incomplete without mention of people who made it possible, whose guidance and encouragement crowned my effort with success. I would like to thank all and acknowledge the help I have received to carry out this Mini Project.

I would like to convey my thanks to Head of Department **Dr. Girija J.** for being kind enough to provide the necessary support to carry out the mini project.

I am most humbled to mention the enthusiastic influence provided by the lab in-charges **Prof. Savitha S K a**nd **Prof. Mahalakshmi C V**, on the project for their ideas, time to time suggestions for being a constant guide and co-operation showed during the venture and making this project a great success.

I would also take this opportunity to thank my friends and family for their constant support and help. I'm very much pleasured to express my sincere gratitude to the friendly co-operation showed by all the **staff members** of Computer Science Department, BIT.

CHIRAYU V GANGADKAR
1BI20CS048

Table of contents

Chapter 1

INTRODUCTION

1.1 Project Summary

In general, a drawing app on Android is a mobile application that allows users to create and edit drawings using various tools and features. These apps typically provide a canvas where users can use their fingers or a stylus to draw, paint, and sketch. They often include features like different brush size, colors, erasers, undo functionality, and the ability to save and share the created drawings.

Drawing apps on Android can be used for artistic purposes, doodling, sketching, or note-taking. They can be a fun and creative way for users to express themselves digitally on their Android devices.

1.2 Project Purpose

The purpose of a drawing app on Android can vary depending on the specific project and its intended audience. However, here are some common purposes and benefits of developing a drawing app:

Creative Expression: Drawing apps provide users with a platform to express their creativity and create digital artwork. Users can explore their artistic skills, experiment with different techniques, and produce unique drawings.

Digital Sketchbook: Drawing apps can serve as a digital sketchbook, allowing users to capture their ideas and concepts on the go. It provides a convenient way to jot down quick sketches, doodles, or rough drafts without the need for traditional pen and paper.

Learning and Skill Development: Drawing apps can be used as educational tools for learning and improving drawing skills. They may offer tutorials, step-by-step guides, or tracing features to help beginners understand and practice various drawing techniques.

Accessibility and Portability: With a drawing app on Android, users can carry their art tools with them wherever they go. It eliminates the need for physical art supplies and allows users to create artwork anytime and anywhere using their Android devices.

Collaboration and Sharing: Many drawing apps provide the option to share artwork with others, fostering a sense of community and enabling collaboration among artists. Users can share their drawings on social media platforms or within the app's community, receive feedback, and connect with fellow artists.

Entertainment and Recreation: Drawing apps can be a source of entertainment and relaxation. Users can engage in a therapeutic and enjoyable activity by creating artwork, coloring, or engaging in interactive drawing games available in the app.

1.3 Project Scope

The project scope of a drawing app on Android can vary based on the specific requirements and goals.

User Interface:

The app should have an intuitive and user-friendly interface that allows users to easily navigate and access various drawing tools and features.

Drawing Tools:

The app should provide a range of drawing tools such as brushes, pencils, erasers, shapes, colours. These tools enable users to create diverse and visually appealing artwork.

Canvas:

The app should offer a canvas where users can draw and create artwork.

Undo Functionality: To enhance the drawing experience, an undo/redo feature should be included, allowing users to correct mistakes or make changes to their artwork.

Saving:

The app should provide options for users to save their drawings locally on the device or in the cloud.

Customization:

Users may desire to customize the app's settings according to their preferences. This could include options for brush sizes, colours, background selection, and overall app themes.

Performance and Optimization:

The app should be optimized for performance to ensure smooth and responsive drawing experience, even on devices with varying hardware capabilities.

Platform Compatibility:

The app should be designed to run on various Android devices, considering different screen sizes, resolutions, and operating system versions.

1.4 Problem Statement

Many individuals, including artists, designers, and creative enthusiasts, lack a convenient and accessible platform to express their artistic abilities digitally. The absence of a user-friendly drawing app on Android devices limits their ability to create, edit, and share their artwork easily. Additionally, the existing drawing apps may not fully cater to the diverse needs of users, lacking essential features, or being overly complex and difficult to navigate. Therefore, there is a need for a comprehensive drawing app on Android that provides a seamless and intuitive user experience, offering a wide range of drawing tools, customization options, and efficient functionalities. The app should address the limitations of existing solutions, focusing on accessibility, performance optimization, and facilitating creative expression. By fulfilling these requirements, the drawing app aims to empower users with a versatile and enjoyable platform to unleash their artistic potential and share their creations with others.

1.5 Objective of the Project

- **Provide a user-friendly interface:** The objective of a paint mobile application is to offer a user-friendly interface that allows users of all skill levels to create and edit digital artwork easily. The application should have intuitive controls and tools that make it accessible and enjoyable for users to express their creativity.
- **Offer a variety of painting tools:** The objective is to provide a diverse range of painting tools such as brushes, erasers, and colour palettes. The application should strive to replicate traditional art mediums and offer a wide array of digital tools to cater to different artistic styles and preferences.
- **Provide customization options:** The objective is to offer customization options to personalize the painting experience. This could include the ability to create custom brushes, import custom colour palettes, and configure the user interface to suit individual preferences. Customization enhances user engagement and allows artists to create a personalized workflow.
- **Ensure stability and performance:** The objective is to create a stable and performant application that can handle complex artwork without lag or crashes. Optimizing performance, minimizing load times, and regularly updating the application to address bugs and enhance stability are crucial objectives for a paint mobile application.

- **Support learning and growth:** The objective is to provide resources and tutorials within the application to help users learn and improve their painting skills. This could include step-by-step guides, video tutorials, and a community section where artists can share their knowledge and provide feedback to foster a supportive learning environment.
- **Enhance accessibility:** The objective is to make the paint mobile application accessible to a wide range of users, including those with disabilities. This involves incorporating features such as voice commands, support for screen readers, adjustable font sizes, and colour contrast options to ensure inclusivity and usability for all users.
- **Continuously innovate and evolve:** The objective is to stay ahead of the curve by continuously innovating and introducing new features that enhance the painting experience. Regular updates should address user feedback, introduce new tools or functionalities, and adapt to emerging trends and technologies to keep the application relevant and engaging.

1.6 Organization of the Report

The project was organized in a systematic way. First, we analysed what are the basic features to be included in the project to make it acceptable. As it is a mobile application project, I made the requirements prior, to have an idea like how are output must look like. After all these, the source code was formulated as a paperwork. All the required tools were gathered. Finally, the successful implementation of the project.

Chapter 2

SYSTEM REQUIREMENTS

2.1 User Characteristics

- **Android Users:** The app targets individuals who own Android devices, such as smartphones or tablets, and are familiar with the Android operating system.
- **Artists and Creatives:** The primary user group consists of artists, designers, illustrators, and individuals with a creative inclination who are interested in creating digital artwork.
- **All Skill Levels:** The app caters to users of varying skill levels, including beginners, intermediate artists, and advanced professionals. It provides features suitable for users at different stages of their artistic journey.
- **Mobile Users:** The app is designed for users who prefer the convenience and flexibility of creating artwork on their Android devices. These users may prefer a portable and accessible solution for drawing on the go.
- **Age Range:** The app can potentially attract users from different age groups, including teenagers, young adults, and adults who have an interest in drawing and digital art.
- **Visual Learners:** Users who prefer visual learning and expression may find the drawing app appealing, as it allows them to create visually engaging artwork using digital tools and techniques.
- **Social Media Enthusiasts:** The app may attract users who enjoy sharing their artwork on social media platforms or engaging with an online art community. These users may be interested in features that facilitate sharing and receiving feedback on their artwork.

2.2 Hardware and Software Requirements

- **Hardware Requirements:**
  1. Android Device: The application should be compatible with a wide range of Android devices, including smartphones and tablets. Consider targeting a minimum Android version that aligns with your target audience's device usage.

- **Software Requirements:**
  1. Android Studio: The official integrated development environment (IDE) for Android app development. It provides tools for coding, debugging, and testing Android applications. Ensure you have the latest version installed.
  2. Java Development Kit (JDK): Android development primarily uses the Java programming language. Install the latest JDK version compatible with your Android Studio version.(JDK 8 and onwards).
  3. Android SDK: The Android Software Development Kit provides the necessary libraries, tools, and APIs for Android development. Make sure you have the appropriate SDK versions installed based on your target Android platform version.
  4. Google Play Services: If you plan to use Google Firebase authentication, you will need to include the Google Play Services library in your project.
  5. Firebase SDK: If you are using Google Firebase for authentication, you'll need to include the Firebase SDK in your project. Ensure you have the necessary dependencies and configurations set up to integrate Firebase into your application.

2.3 Project Operation Constraints

- Screen Size and Resolution: Mobile devices come in various screen sizes and resolutions. Your paint application should be able to adapt to different screen sizes and resolutions to ensure that the user interface and drawing elements are displayed correctly.
- Touch Input: Paint applications heavily rely on touch input for drawing and interacting with the canvas. However, touchscreens may have limitations, such as limited multitouch support or varying touch sensitivities. You should account for these constraints to provide a smooth and accurate drawing experience.
- Processing Power: Different Android devices have varying processing power, ranging from low-end to high-end devices. It's crucial to optimize your paint application to perform well on devices with lower processing power, ensuring smooth rendering and responsiveness.
- Memory Constraints: Mobile devices have limited memory resources. If your paint application involves working with large image files or complex drawing operations, it's important to manage memory efficiently to prevent crashes or slowdowns.
- Battery Life: Paint applications can be resource-intensive, and continuous usage can drain the device's battery quickly. Consider implementing power-saving techniques and optimizing resource usage to minimize the impact on battery life.
- Compatibility: The Android ecosystem is fragmented, with a wide variety of devices running different versions of the Android operating system. Ensure that your paint application is compatible with the target Android versions and thoroughly test it on a range of devices to ensure consistent behaviour.

Chapter 3

DESIGN

3.1 System Architecture Design

1. **Model-View-Controller (MVC) Architecture:** The project follows the MVC architectural pattern. The DrawView serves as the View, responsible for displaying the drawing canvas and handling user interactions. The MainActivity acts as the Controller, managing the user input and updating the View accordingly. The Model consists of the Stroke class, representing individual drawing strokes, and the paths ArrayList, storing the collection of strokes.
2. **Single-Activity Design:** The project adopts a single-activity design where the MainActivity serves as the primary activity. This approach simplifies the navigation and interaction flow within the application, keeping all the painting-related functionality within a single screen.
3. **Event-Driven Programming:** The project heavily relies on event-driven programming, with various event listeners implemented for button clicks and touch events. These listeners respond to user actions and trigger appropriate actions, such as updating the drawing canvas, changing stroke properties, or saving the drawing.

3.2 Module Description

1. **MainActivity Module:** This module represents the main activity of the application. It handles the initialization of the views, sets up event listeners for buttons, manages the range slider for stroke width selection, and controls the overall flow of the application.
2. **DrawView Module:** This module represents a custom View that acts as the drawing canvas. It handles user touch events, tracks finger movements, and draws strokes on the canvas using the specified color and stroke width. It also provides methods for setting the color and stroke width, undoing the last stroke, and saving the drawing as a bitmap.
3. **Stroke Module:** This module represents a single stroke in the drawing. It consists of attributes such as color, stroke width, and a Path object that defines the actual path drawn by the user. Strokes are stored in an ArrayList to keep track of the drawing history.

4. **Color Picker Module:** This module utilizes the AmbilWarnaDialog library to provide a color picker dialog. It allows users to select a color for the brush by displaying a dialog with a color wheel and options to confirm or cancel the color selection.
5. **RangeSlider Module:** This module represents a custom slider widget for selecting the stroke width. It allows users to slide a thumb along a range of values to adjust the stroke width dynamically.
6. **Image Saving Module:** This module handles the functionality to save the drawing as an image file. It creates a bitmap from the DrawView, creates a file in the storage directory, and writes the bitmap data into the file in PNG format using OutputStream**.**

Chapter 4

IMPLEMENTATION

4.1 Built-in Function

1. **onCreate(Bundle savedInstanceState):** This function is called when the activity is created. It initializes the activity, sets the content view, retrieves references to various views, sets click listeners, sets up the range slider, and initializes the custom DrawView.

2. **onClick(View view):** This function serves as a click event listener for buttons. It handles the actions to be performed when different buttons are clicked, such as undoing the last stroke, saving the drawing as an image, opening a color picker dialog, and toggling the visibility of the range slider.

3. **onDraw(Canvas canvas):** This function is called when the custom View is being drawn. It overrides the default behavior and handles the drawing of strokes on the canvas. It iterates over the list of stored strokes and draws each stroke using the specified color and stroke width.

4. **onTouchEvent(MotionEvent event):** This function is called when a touch event occurs on the custom View. It overrides the default behavior and handles touch events, such as finger down, move, and up actions. It tracks the finger movement and updates the path being drawn accordingly.

5. **init(int height, int width):** This method initializes the custom DrawView by creating a bitmap and canvas with the specified height and width. It sets the initial color and stroke width for drawing.

6. **setColor(int color):** This method sets the current color for drawing strokes.

7. **setStrokeWidth(int width):** This method sets the stroke width for drawing strokes.

8. **undo():** This method removes the most recent stroke from the list of strokes, effectively undoing the last drawing action.

9. **save():** This method returns the current drawing as a bitmap, which can be saved or further processed.

10. **openColorPicker():** This method opens a color picker dialog for the user to select a color. The selected color is then set as the current color for drawing strokes.

11. Constructor **DrawView(Context context):** This constructor is used to create an instance of the custom DrawView. It initializes the Paint object with default settings for drawing strokes.

12. Constructor **Stroke(int color, int strokeWidth, Path path):** This constructor is used to create a Stroke object, which represents a single stroke with its color, stroke width, and path. It is used to store and draw strokes on the canvas.

4.2 Android Manifest and Special Permissions

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.Paint"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Special Permissions:

android.permission.WRITE_EXTERNAL_STORAGE: This permission is required to save the drawing as an image file in the external storage. It is used when the user clicks the save button to save the drawing as a PNG image.

android.permission.READ_EXTERNAL_STORAGE: This permission is required to read the media files from the external storage. It is used when accessing the directory for saving the drawing image.

4.3 XML and Java Code

1. Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:background="#FFFFFF"
    android:layout_height="match_parent"
    tools:context=".MainActivity">



    <LinearLayout
        android:id="@+id/linear"
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical">

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <ImageButton
        android:id="@+id/btn_undo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:src="@drawable/ic_undo"
        android:text="Undo" />

    <ImageButton
        android:id="@+id/btn_save"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:src="@drawable/ic_floppy_disk"
        android:text="Save" />

    <ImageButton
        android:id="@+id/btn_color"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
            android:layout_weight="1"
            android:src="@drawable/ic_colorpicker"
            android:text="Color" />

        <ImageButton
            android:id="@+id/btn_stroke"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:src="@drawable/ic_paint_brush"
            android:text="Stroke" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <com.google.android.material.slider.RangeSlider
            android:id="@+id/rangebar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:visibility="gone" />
    </LinearLayout>
</LinearLayout>
<com.example.paint.DrawView
    android:id="@+id/draw_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
        android:layout_below="@id/linear"
        android:layout_centerInParent="true" />



</RelativeLayout>
```

2. MainActivity.java

```
package com.example.paint;

import android.content.ContentValues;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.provider.MediaStore;
import android.view.View;
import android.view.ViewTreeObserver;
import android.widget.ImageButton;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.ContextCompat;



import com.google.android.material.slider.RangeSlider;

import java.io.OutputStream;
import yuku.ambilwarna.AmbilWarnaDialog;

public class MainActivity extends AppCompatActivity {
    int defaultColor;
    // creating the object of type DrawView
    // in order to get the reference of the View
    private DrawView paint;

    // creating objects of type button
    private ImageButton save, color, stroke, undo;

    // creating a RangeSlider object, which will
    // help in selecting the width of the Stroke
    private RangeSlider rangeSlider;

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

        // getting the reference of the views from their ids
        paint = (DrawView) findViewById(R.id.draw_view);
        rangeSlider = (RangeSlider) findViewById(R.id.rangebar);
        undo = (ImageButton) findViewById(R.id.btn_undo);
        save = (ImageButton) findViewById(R.id.btn_save);
        color = (ImageButton) findViewById(R.id.btn_color);
        stroke = (ImageButton) findViewById(R.id.btn_stroke);
```

```
// creating a OnClickListener for each button,
// to perform certain actions

// the undo button will remove the most
// recent stroke from the canvas
undo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        paint.undo();
    }
});

// the save button will save the current
// canvas which is actually a bitmap
// in form of PNG, in the storage
save.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        // getting the bitmap from DrawView class
        Bitmap bmp = paint.save();

        // opening a OutputStream to write into the file
        OutputStream imageOutStream = null;

        ContentValues cv = new ContentValues();

        // name of the file
        cv.put(MediaStore.Images.Media.DISPLAY_NAME, 
"drawing.png");

        // type of the file
        cv.put(MediaStore.Images.Media.MIME_TYPE, "image/png");

        // location of the file to be saved
        cv.put(MediaStore.Images.Media.RELATIVE_PATH, 
Environment.DIRECTORY_PICTURES);

        // get the Uri of the file which is to be created in the storage
        Uri uri = 
getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTEN
T_URI, cv);
        try {
          // open the output stream with the above uri
          imageOutStream = getContentResolver().openOutputStream(uri);

          // this method writes the files in storage
          bmp.compress(Bitmap.CompressFormat.PNG, 100, 
imageOutStream);

          // close the output stream after use
          imageOutStream.close();
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    });
```

```
defaultColor = ContextCompat.getColor(MainActivity.this,R.color.black);
// the color button will allow the user
// to select the color of his brush
color.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        openColorPicker();
    }
});
// the button will toggle the visibility of the RangeBar/RangeSlider
stroke.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (rangeSlider.getVisibility() == View.VISIBLE)
            rangeSlider.setVisibility(View.GONE);
        else
            rangeSlider.setVisibility(View.VISIBLE);
    }
});

// set the range of the RangeSlider
rangeSlider.setValueFrom(0.0f);
rangeSlider.setValueTo(100.0f);

// adding a OnChangeListener which will
// change the stroke width
// as soon as the user slides the slider
```

```
        rangeSlider.addOnChangeListener(new RangeSlider.OnChangeListener()
{
            @Override
            public void onValueChange(@NonNull RangeSlider slider, float value,
boolean fromUser) {
                paint.setStrokeWidth((int) value);
            }
        });

        // pass the height and width of the custom view
        // to the init method of the DrawView object
        ViewTreeObserver vto = paint.getViewTreeObserver();
        vto.addOnGlobalLayoutListener(new
ViewTreeObserver.OnGlobalLayoutListener() {
            @Override
            public void onGlobalLayout() {
                paint.getViewTreeObserver().removeOnGlobalLayoutListener(this);
                int width = paint.getMeasuredWidth();
                int height = paint.getMeasuredHeight();
                paint.init(height, width);
            }
        });
    }

    private void openColorPicker() {
        AmbilWarnaDialog ambilWarnaDialog = new AmbilWarnaDialog(this,
defaultColor, new AmbilWarnaDialog.OnAmbilWarnaListener() {
            @Override
            public void onCancel(AmbilWarnaDialog dialog) {
            }

            @Override
            public void onOk(AmbilWarnaDialog dialog, int color) {
                defaultColor = color;
                paint.setColor(color);

            }
        });
        ambilWarnaDialog.show();
    }
}
```

3. DrawView.java

```
package com.example.paint;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;

import com.example.paint.Stroke;
import java.util.ArrayList;

public class DrawView extends View {

    private static final float TOUCH_TOLERANCE = 4;
    private float mX, mY;
    private Path mPath;

    // the Paint class encapsulates the color
    // and style information about
    // how to draw the geometries,text and bitmaps
    private Paint mPaint;

    // ArrayList to store all the strokes
    // drawn by the user on the Canvas
    private ArrayList<Stroke> paths = new ArrayList<>();
    private int currentColor;
    private int strokeWidth;
    private Bitmap mBitmap;
    private Canvas mCanvas;
    private Paint mBitmapPaint = new Paint(Paint.DITHER_FLAG);

    // Constructors to initialise all the attributes
    public DrawView(Context context) {
        this(context, null);
    }

    public DrawView(Context context, AttributeSet attrs) {
        super(context, attrs);
        mPaint = new Paint();

        // the below methods smoothens
        // the drawings of the user
        mPaint.setAntiAlias(true);
        mPaint.setDither(true);
        mPaint.setColor(Color.GREEN);
        mPaint.setStyle(Paint.Style.STROKE);
        mPaint.setStrokeJoin(Paint.Join.ROUND);
        mPaint.setStrokeCap(Paint.Cap.ROUND);

        // 0xff=255 in decimal
        mPaint.setAlpha(0xff);

    }

    // this method instantiate the bitmap and object
    public void init(int height, int width) {

        mBitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888);
        mCanvas = new Canvas(mBitmap);

        // set an initial color of the brush
        currentColor = Color.GREEN;

        // set an initial brush size
        strokeWidth = 20;
```

```
}

// sets the current color of stroke
public void setColor(int color) {
    currentColor = color;
}

// sets the stroke width
public void setStrokeWidth(int width) {
    strokeWidth = width;
}

public void undo() {
    // check whether the List is empty or not
    // if empty, the remove method will return an error
    if (paths.size() != 0) {
        paths.remove(paths.size() - 1);
        invalidate();
    }
}

// this methods returns the current bitmap
public Bitmap save() {
    return mBitmap;
}

// this is the main method where
// the actual drawing takes place
@Override
```

```
protected void onDraw(Canvas canvas) {
    // save the current state of the canvas before,
    // to draw the background of the canvas
    canvas.save();

    // DEFAULT color of the canvas
    int backgroundColor = Color.WHITE;
    mCanvas.drawColor(backgroundColor);

    // now, we iterate over the list of paths
    // and draw each path on the canvas
    for (Stroke fp : paths) {
        mPaint.setColor(fp.color);
        mPaint.setStrokeWidth(fp.strokeWidth);
        mCanvas.drawPath(fp.path, mPaint);
    }
    canvas.drawBitmap(mBitmap, 0, 0, mBitmapPaint);
    canvas.restore();
}

// the below methods manages the touch
// response of the user on the screen

// firstly, we create a new Stroke
// and add it to the paths list
private void touchStart(float x, float y) {
    mPath = new Path();
    Stroke fp = new Stroke(currentColor, strokeWidth, mPath);
    paths.add(fp);
```

```
        // finally remove any curve
        // or line from the path
        mPath.reset();

        // this methods sets the starting
        // point of the line being drawn
        mPath.moveTo(x, y);

        // we save the current
        // coordinates of the finger
        mX = x;
        mY = y;
    }

    // in this method we check
    // if the move of finger on the
    // screen is greater than the
    // Tolerance we have previously defined,
    // then we call the quadTo() method which
    // actually smooths the turns we create,
    // by calculating the mean position between
    // the previous position and current position
    private void touchMove(float x, float y) {
        float dx = Math.abs(x - mX);
        float dy = Math.abs(y - mY);

        if (dx >= TOUCH_TOLERANCE || dy >= TOUCH_TOLERANCE) {
            mPath.quadTo(mX, mY, (x + mX) / 2, (y + mY) / 2);
            mX = x;
            mY = y;
        }
    }

    // at the end, we call the lineTo method
    // which simply draws the line until
    // the end position
    private void touchUp() {
        mPath.lineTo(mX, mY);
    }

    // the onTouchEvent() method provides us with
    // the information about the type of motion
    // which has been taken place, and according
    // to that we call our desired methods
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        float x = event.getX();
        float y = event.getY();

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                touchStart(x, y);
                invalidate();
                break;
            case MotionEvent.ACTION_MOVE:
                touchMove(x, y);
                invalidate();
                break;
            case MotionEvent.ACTION_UP:
                touchUp();
                invalidate();
                break;
        }
        return true;
    }
}
com.example.paint;

import android.content.ContentValues;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.provider.MediaStore;
import android.view.View;
import android.view.ViewTreeObserver;
import android.widget.ImageButton;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.ContextCompat;


import com.google.android.material.slider.RangeSlider;
import java.io.OutputStream;

import yuku.ambilwarna.AmbilWarnaDialog;

public class MainActivity extends AppCompatActivity {
    int defaultColor;
    // creating the object of type DrawView
    // in order to get the reference of the View
    private DrawView paint;

    // creating objects of type button
    private ImageButton save, color, stroke, undo;

    // creating a RangeSlider object, which will
    // help in selecting the width of the Stroke
    private RangeSlider rangeSlider;

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

        // getting the reference of the views from their ids
        paint = (DrawView) findViewById(R.id.draw_view);
        rangeSlider = (RangeSlider) findViewById(R.id.rangebar);
        undo = (ImageButton) findViewById(R.id.btn_undo);
        save = (ImageButton) findViewById(R.id.btn_save);
        color = (ImageButton) findViewById(R.id.btn_color);
        stroke = (ImageButton) findViewById(R.id.btn_stroke);
```

```
// creating a OnClickListener for each button,
// to perform certain actions

// the undo button will remove the most
// recent stroke from the canvas
undo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        paint.undo();
    }
});

// the save button will save the current
// canvas which is actually a bitmap
// in form of PNG, in the storage
save.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        // getting the bitmap from DrawView class
        Bitmap bmp = paint.save();

        // opening a OutputStream to write into the file
        OutputStream imageOutStream = null;

        ContentValues cv = new ContentValues();

        // name of the file
```

```
            cv.put(MediaStore.Images.Media.DISPLAY_NAME,
"drawing.png");

            // type of the file
            cv.put(MediaStore.Images.Media.MIME_TYPE, "image/png");

            // location of the file to be saved
            cv.put(MediaStore.Images.Media.RELATIVE_PATH,
Environment.DIRECTORY_PICTURES);

            // get the Uri of the file which is to be created in the storage
            Uri uri =
getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTEN
T_URI, cv);
            try {
               // open the output stream with the above uri
               imageOutStream = getContentResolver().openOutputStream(uri);

               // this method writes the files in storage
               bmp.compress(Bitmap.CompressFormat.PNG, 100,
imageOutStream);

               // close the output stream after use
               imageOutStream.close();
            } catch (Exception e) {
               e.printStackTrace();
            }
         }
      });
```

```
defaultColor = ContextCompat.getColor(MainActivity.this,R.color.black);
// the color button will allow the user
// to select the color of his brush
color.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        openColorPicker();
    }
});
// the button will toggle the visibility of the RangeBar/RangeSlider
stroke.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (rangeSlider.getVisibility() == View.VISIBLE)
            rangeSlider.setVisibility(View.GONE);
        else
            rangeSlider.setVisibility(View.VISIBLE);
    }
});

// set the range of the RangeSlider
rangeSlider.setValueFrom(0.0f);
rangeSlider.setValueTo(100.0f);

// adding a OnChangeListener which will
// change the stroke width
// as soon as the user slides the slider
```

```
        rangeSlider.addOnChangeListener(new RangeSlider.OnChangeListener()
{
            @Override
            public void onValueChange(@NonNull RangeSlider slider, float value,
boolean fromUser) {
                paint.setStrokeWidth((int) value);
            }
        });

        // pass the height and width of the custom view
        // to the init method of the DrawView object
        ViewTreeObserver vto = paint.getViewTreeObserver();
        vto.addOnGlobalLayoutListener(new
ViewTreeObserver.OnGlobalLayoutListener() {
            @Override
            public void onGlobalLayout() {
                paint.getViewTreeObserver().removeOnGlobalLayoutListener(this);
                int width = paint.getMeasuredWidth();
                int height = paint.getMeasuredHeight();
                paint.init(height, width);
            }
        });
    }

    private void openColorPicker() {
        AmbilWarnaDialog ambilWarnaDialog = new AmbilWarnaDialog(this,
defaultColor, new AmbilWarnaDialog.OnAmbilWarnaListener() {
            @Override
            public void onCancel(AmbilWarnaDialog dialog) {
            }

            @Override
            public void onOk(AmbilWarnaDialog dialog, int color) {
                defaultColor = color;
                paint.setColor(color);

            }
        });
        ambilWarnaDialog.show();
    }
}
```

4. Stroke.java

```
package com.example.paint;
import android.graphics.Path;

public class Stroke {

    // color of the stroke
    public int color;

    // width of the stroke
    public int strokeWidth;

    // a Path object to
    // represent the path drawn
    public Path path;

    // constructor to initialise the attributes
    public Stroke(int color, int strokeWidth, Path path) {
        this.color = color;
        this.strokeWidth = strokeWidth;
        this.path = path;
    }
}
```

Chapter 6

CONCLUSION

6.1 Future Enhancement

1. Layers: Implementing support for multiple layers would allow users to draw and manage different elements separately. Layers can be stacked, reordered, and have individual properties such as opacity and blending modes.

2. Brush Effects: Adding various brush effects such as different brush shapes (e.g., square, triangular), textured brushes, and pattern brushes can enhance the creative possibilities for users.

3. Shape Tools: Introducing shape tools like rectangles, circles, and polygons would enable users to easily draw geometric shapes with customizable properties such as fill color, stroke color, and stroke width.

4. Image Import and Editing: Enabling users to import existing images as layers or background and providing basic image editing capabilities (e.g., resizing, rotating, applying filters) within the paint application would offer more versatility in creating artwork.

5. Brush Customization: Allowing users to customize brush parameters such as opacity, flow, hardness, and texture can provide greater control over brush strokes and enable a wider range of artistic effects.

6. Advanced Editing Tools: Adding advanced editing tools like selection tools (lasso, magic wand), transformation tools (scale, rotate, distort), and filters (blur, sharpen, adjust colors) can enhance the editing capabilities and give users more options for refining their artwork.

6.2 Bibliography

In an Android alarm clock project with wake-up call toast and Google Firebase authentication, the bibliography can include references to any external resources, libraries, or frameworks that were used during the development process. Since the provided code snippet doesn't include any external references, here is a general example of what a bibliography could look like for an Android development project:

Android Developer Documentation:
Website: https://developer.android.com/

Firebase Documentation:
Website: https://firebase.google.com/docs

Android Developers Blog:
Website: https://android-developers.googleblog.com/

Stack Overflow:
Website: https://stackoverflow.com/

Chapter 5

SNAPSHOTS

Fig 5.1 Home Page

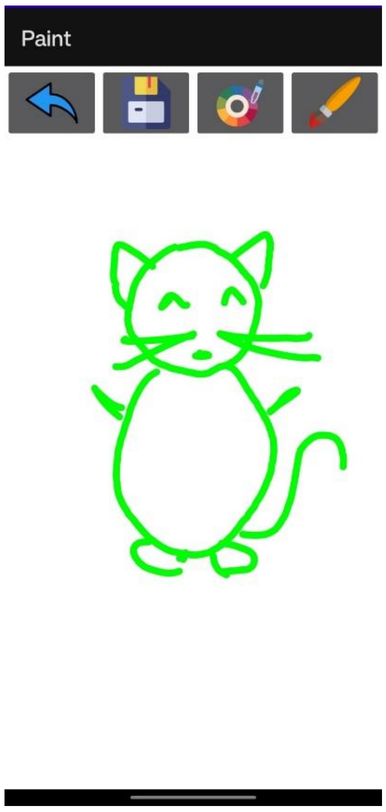

Fig 5.2 Drawing with default brush colour and size

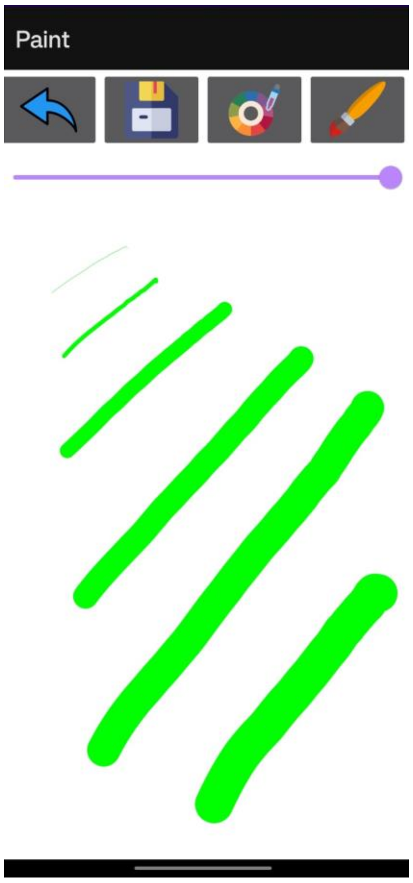

Fig 5.3 Change brush Size

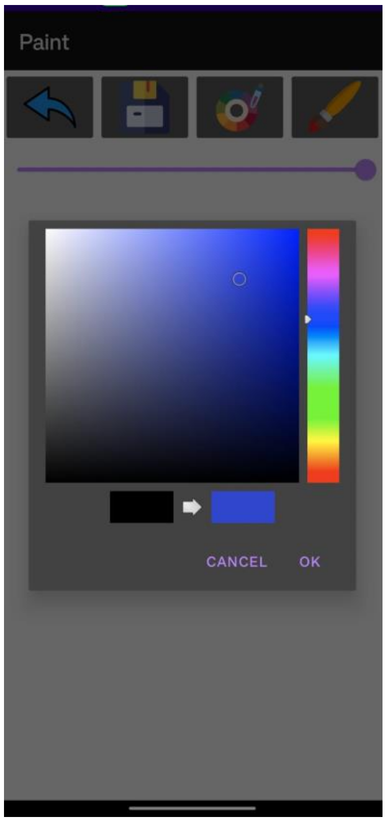

Fig 5.4 Change Brush Colour

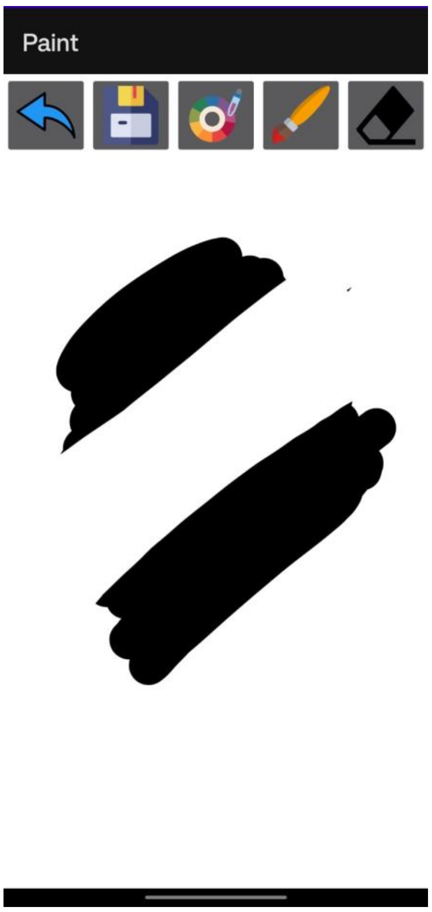

Fig 5.5 Erase the Drawn Image

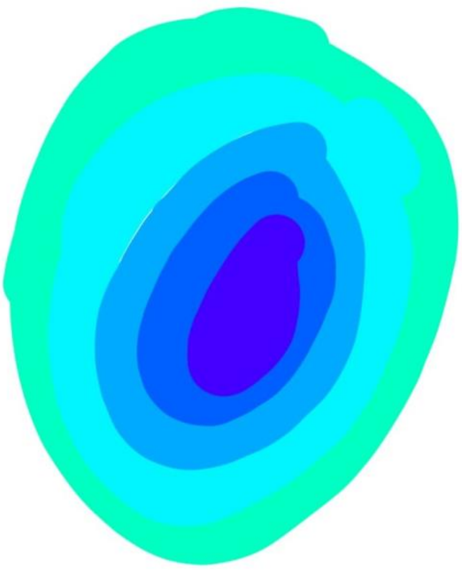

Fig 5.6 Saving the Drawn Image into the Device Gallery