Development Scenario: Third Eye Automotive Surveillance

Day 1: Automotive Security Introduction and Environment Setup, Camera Integration and Motion Detection

Task 1: Learn about Automotive Security Systems

Objective: Understand camera-based surveillance systems for both moving and parked vehicles.

Introduction: Automotive security systems have evolved significantly with the integration of cameras, sensors, and advanced software. These systems are designed to enhance vehicle security by monitoring both external threats (like theft and vandalism) and internal conditions (like passenger safety and driver behaviour). Camera-based surveillance plays a crucial role in these systems by providing real-time monitoring, event recording, and remote access capabilities.

Key Components of Camera-Based Surveillance Systems:

1. **Camera Types:**
    - Exterior Cameras: Placed strategically to monitor the surroundings of the vehicle, including blind spots and parking areas.
    - Interior Cameras: Monitor inside the vehicle, focusing on passenger and driver behaviour, providing added security and safety features.
2. **Features and Functionality:**
    - Live Video Feeds: Enable real-time monitoring of the vehicle's surroundings, providing immediate visual information to the user or a central monitoring station.
    - Event-Driven Recording: Cameras can be triggered to record based on specific events such as motion detection, door opening, or impact sensing.
    - Remote Access: Allows users to access camera feeds and recorded footage remotely via mobile apps or web interfaces, enhancing surveillance and security management capabilities.
    - Night Vision: Utilizes infrared technology or low-light sensitivity to ensure visibility during nighttime or low-light conditions, crucial for 24/7 surveillance.
3. **Integration with Vehicle Systems:**
    - Connected Car Integration: Integrating with the vehicle's onboard systems to leverage existing sensors and data sources for enhanced functionality and accuracy.

- Integration with Alarm Systems: Cameras can be integrated with alarm systems to provide visual confirmation of alarms, reducing false alarms and improving response times.

Applications:

1. Security and Theft Prevention:
    - Deterrence: Visible cameras act as a deterrent against theft and vandalism, reducing the risk of incidents.
    - Evidence Collection: Recorded footage serves as crucial evidence for law enforcement and insurance claims in case of incidents.
2. **Safety and Surveillance:**
    - Driver Monitoring: Interior cameras monitor driver behavior for signs of fatigue, distraction, or impairment, enhancing road safety.
    - Parking Assistance: Exterior cameras aid in parking maneuvers, providing views of blind spots and obstacles, reducing accidents.

Technological Advancements: Recent advancements in automotive security systems include:

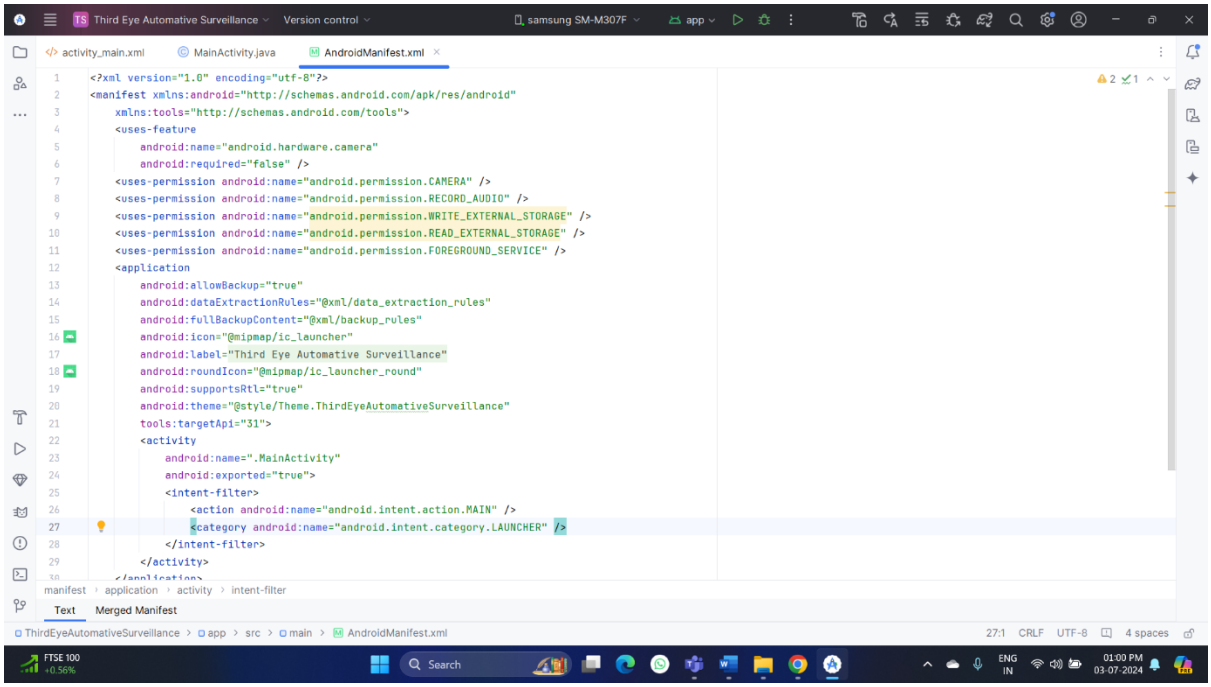
- Artificial Intelligence (AI) Integration: AI-powered analytics enable advanced features like object detection, facial recognition, and behavior analysis.
- Cloud Integration: Storing footage and accessing real-time data via cloud platforms for scalability and remote management.
- Integration with Smart Home Systems: Connecting vehicle surveillance systems with home security systems for seamless monitoring and control.

Conclusion: Camera-based surveillance systems for automotive security are integral in modern vehicles, offering enhanced protection, safety features, and peace of mind to vehicle owners and operators. Understanding these systems' capabilities and functionalities is essential for developing effective and reliable automotive security solutions.

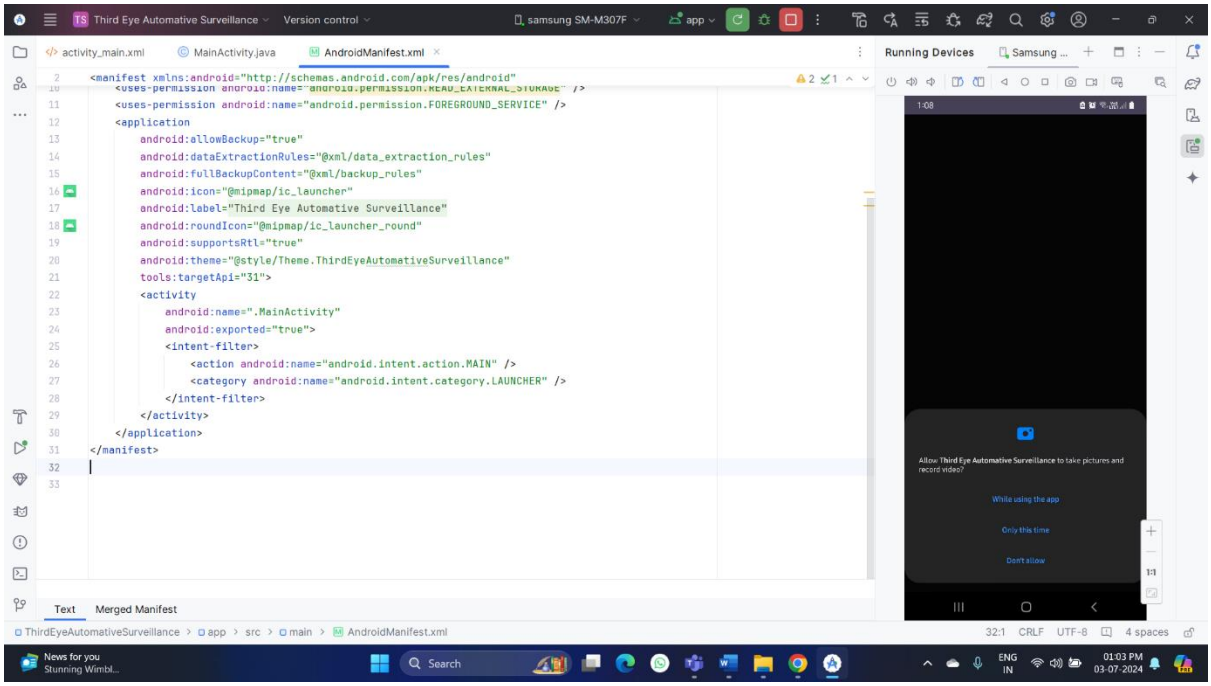
This theoretical understanding provides a foundation for implementing camera-based surveillance within your 'Third Eye Automotive Surveillance' project, ensuring comprehensive coverage of security needs.

Task 2: Set up an Android development environment and create a project that includes permissions for camera access and background processing

1. Install Android Studio: Download and install the latest version of Android Studio from the official website.
2. Create a New Project:
    - Open Android Studio.
    - Click on "Start a new Android Studio project."
    - Select "Empty Activity" and click "Next."
    - Set the project name to ThirdAutomativeEyeSurveillance.
    - Set the package name to com.example.thirdautomativeeyesurveillance.
    - Choose a save location.
    - Select a language (Java).
    - Set the Minimum API level to 21 (Lollipop) or higher.
    - Click "Finish."
3. **Add Permissions to AndroidManifest.xml**:

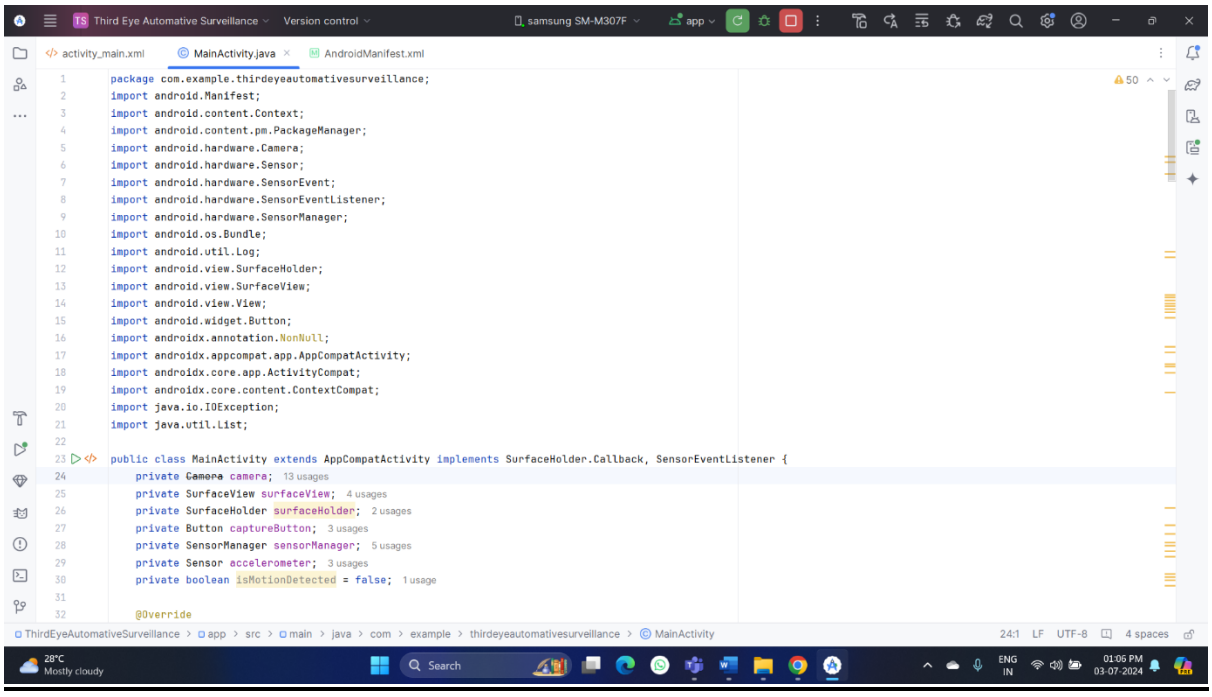


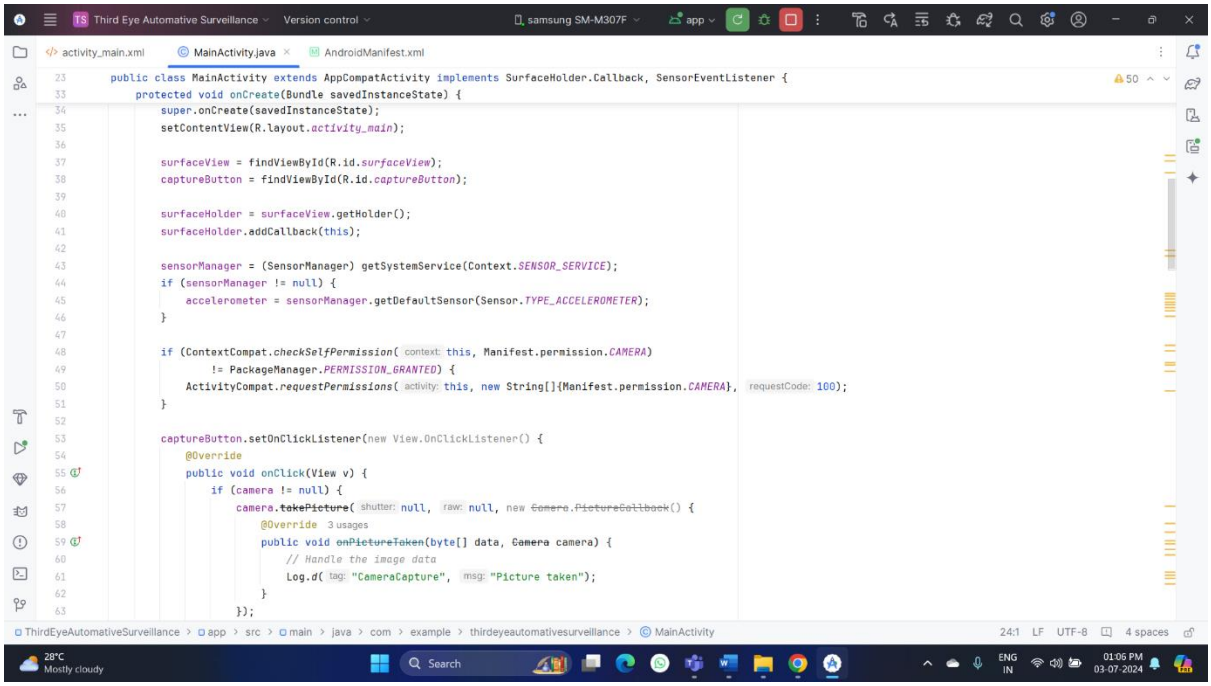
DAY 1 TASK 2 OUTPUT SCREENSHOT

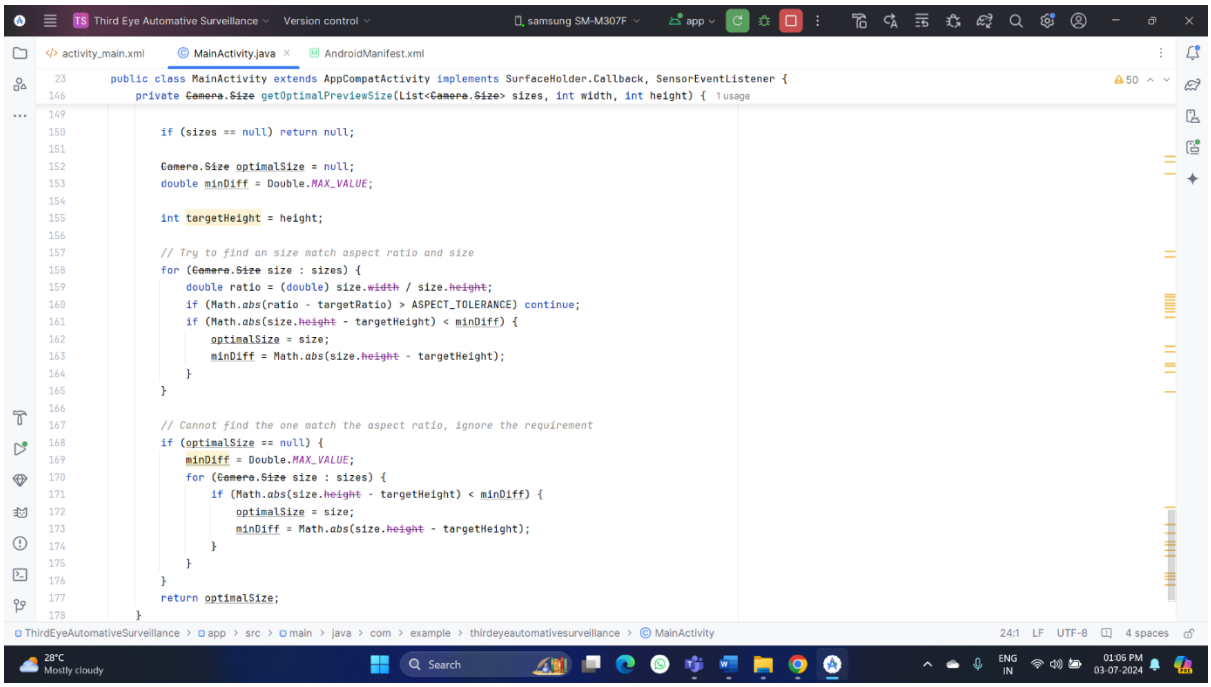


Task 3: Integrate the camera API to capture live video feeds and still images from within an Android application.

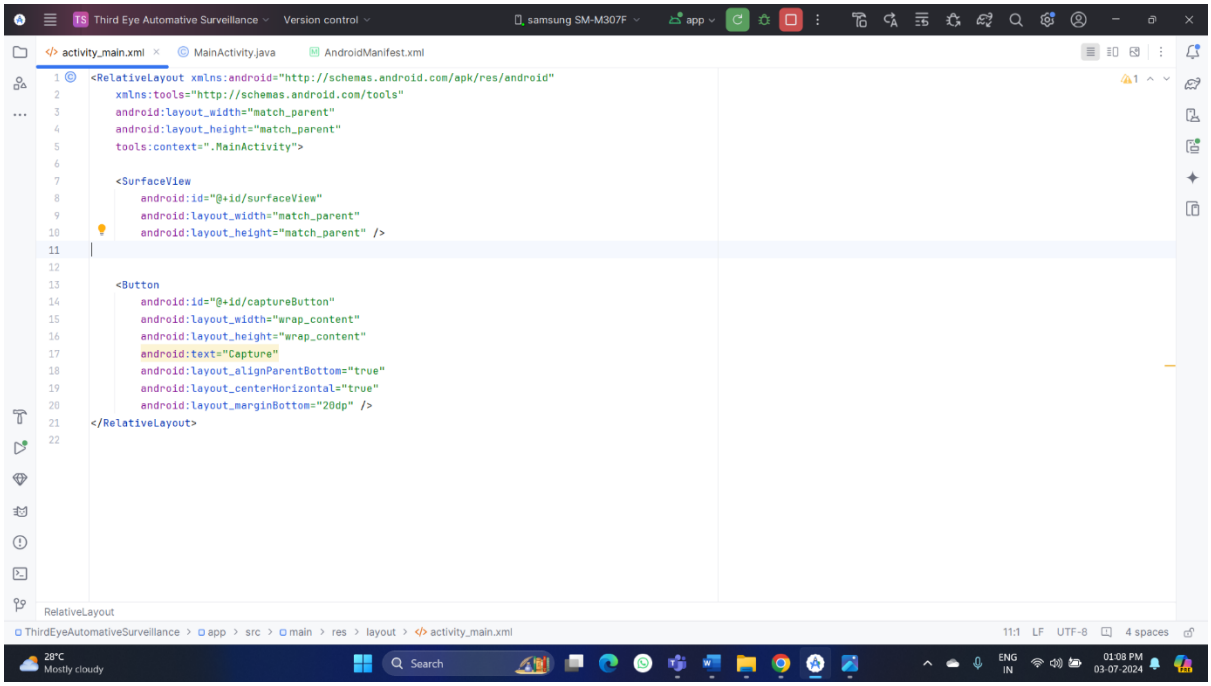
MainActivity.java



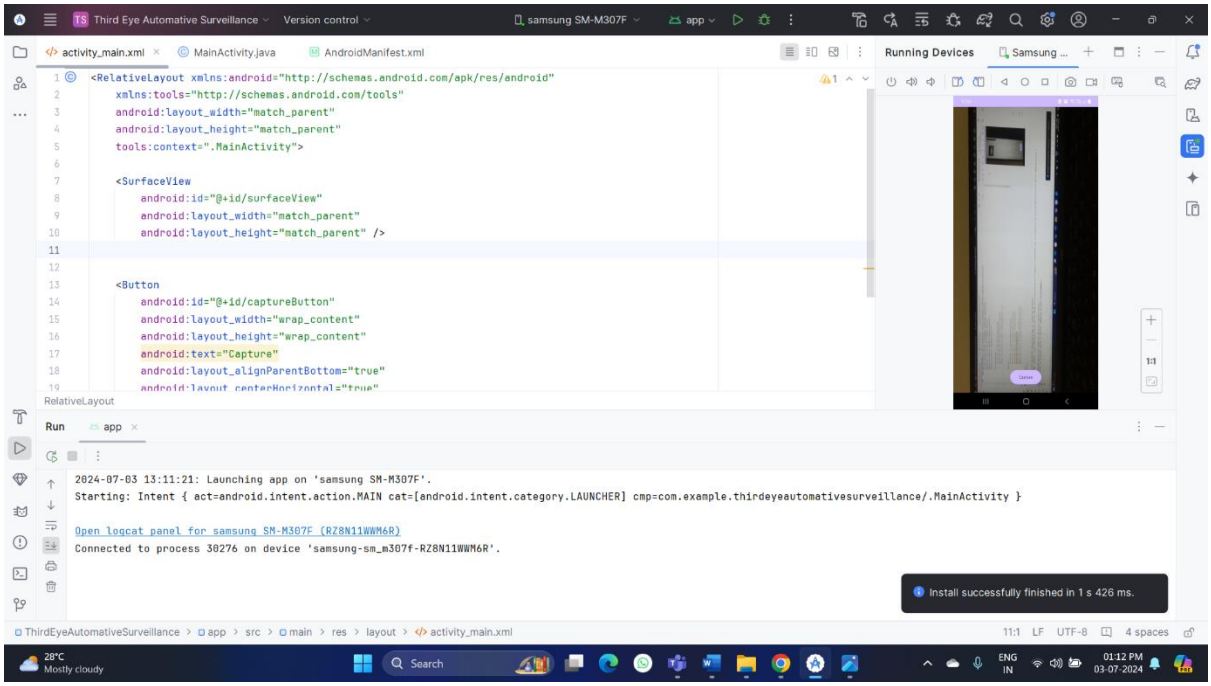

Third Eye Automative Surveillance
Version control
samsung SM-M307F
app
activity_main.xml
MainActivity.java
AndroidManifest.xml
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, SensorEventListener {
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
surfaceView = findViewById(R.id.surfaceView);
captureButton = findViewById(R.id.captureButton);
surfaceHolder = surfaceView.getHolder();
surfaceHolder.addCallback(this);
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (sensorManager != null) {
accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
}
if (ContextCompat.checkSelfPermission(context: this, Manifest.permission.CAMERA)
!= PackageManager.PERMISSION_GRANTED) {
ActivityCompat.requestPermissions(activity: this, new String[]{Manifest.permission.CAMERA}, requestCode: 100);
}
captureButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
if (camera != null) {
camera.takePicture(shutter: null, raw: null, new Camera.PictureCallback() {
@Override 3 usages
public void onPictureTaken(byte[] data, Camera camera) {
// Handle the image data
Log.d(tag: "CameraCapture", msg: "Picture taken");
}
});
ThirdEyeAutomativeSurveillance > app > src > main > java > com > example > thirdeyeautomativesurveillance > MainActivity
24:1 LF UTF-8 4 spaces
28°C
Mostly cloudy
Search
ENG
IN
01:06 PM
03-07-2024


Third Eye Automative Surveillance
Version control
samsung SM-M307F
app
activity_main.xml
MainActivity.java
AndroidManifest.xml
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, SensorEventListener {
private Camera.Size getOptimalPreviewSize(List<Camera.Size> sizes, int width, int height) { 1 usage
if (sizes == null) return null;
Camera.Size optimalSize = null;
double minDiff = Double.MAX_VALUE;
int targetHeight = height;
// Try to find an size match aspect ratio and size
for (Camera.Size size : sizes) {
double ratio = (double) size.width / size.height;
if (Math.abs(ratio - targetRatio) > ASPECT_TOLERANCE) continue;
if (Math.abs(size.height - targetHeight) < minDiff) {
optimalSize = size;
minDiff = Math.abs(size.height - targetHeight);
}
}
// Cannot find the one match the aspect ratio, ignore the requirement
if (optimalSize == null) {
minDiff = Double.MAX_VALUE;
for (Camera.Size size : sizes) {
if (Math.abs(size.height - targetHeight) < minDiff) {
optimalSize = size;
minDiff = Math.abs(size.height - targetHeight);
}
}
}
return optimalSize;
}
ThirdEyeAutomativeSurveillance > app > src > main > java > com > example > thirdeyeautomativesurveillance > MainActivity
24:1 LF UTF-8 4 spaces
28°C
Mostly cloudy
Search
ENG
IN
01:06 PM
03-07-2024

activity_main.xml

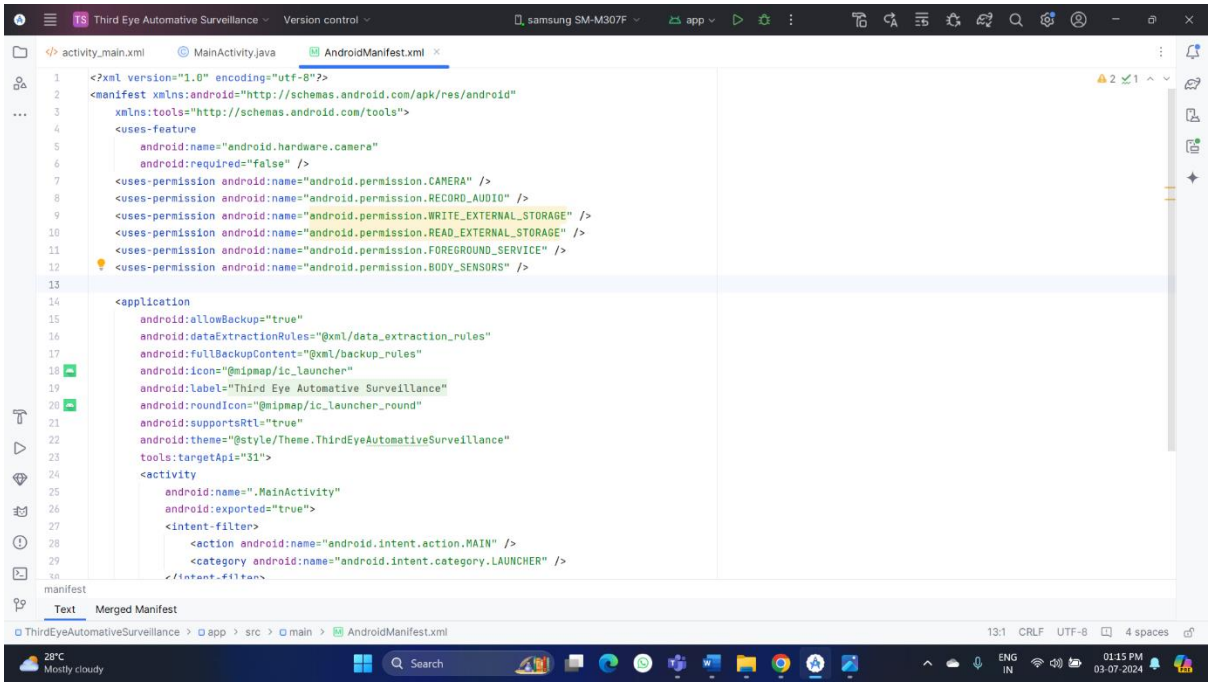


DAY 1 TASK 3 OUTPUT SCREENSHOT

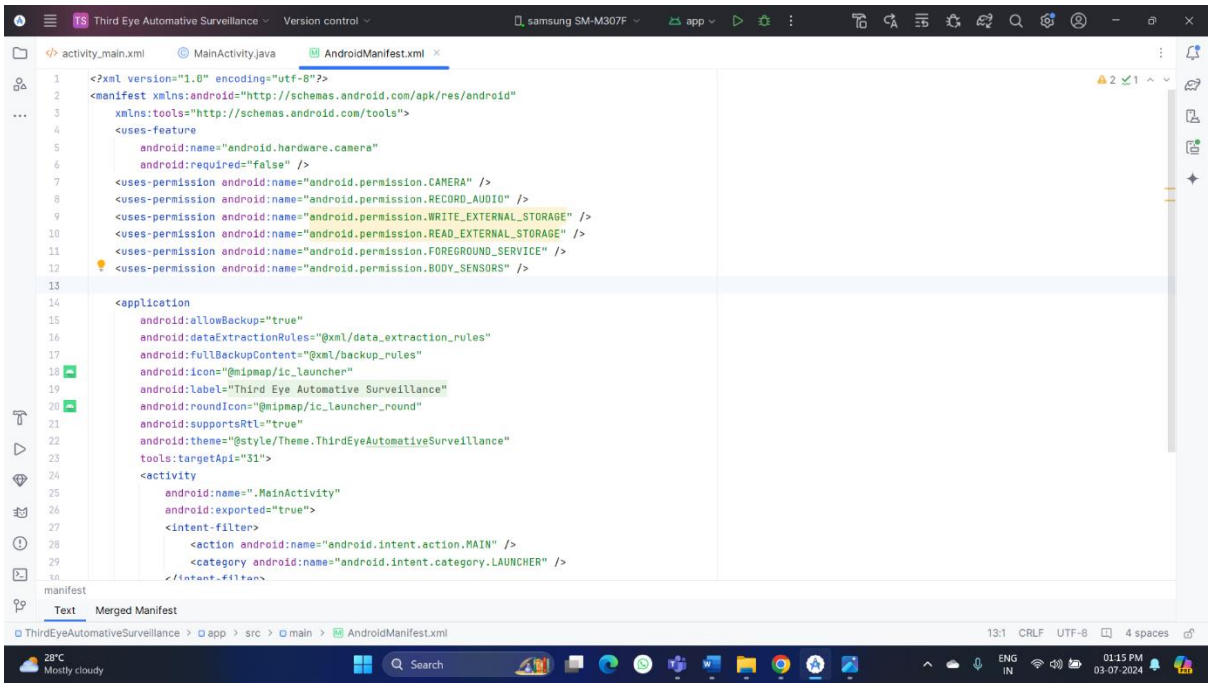


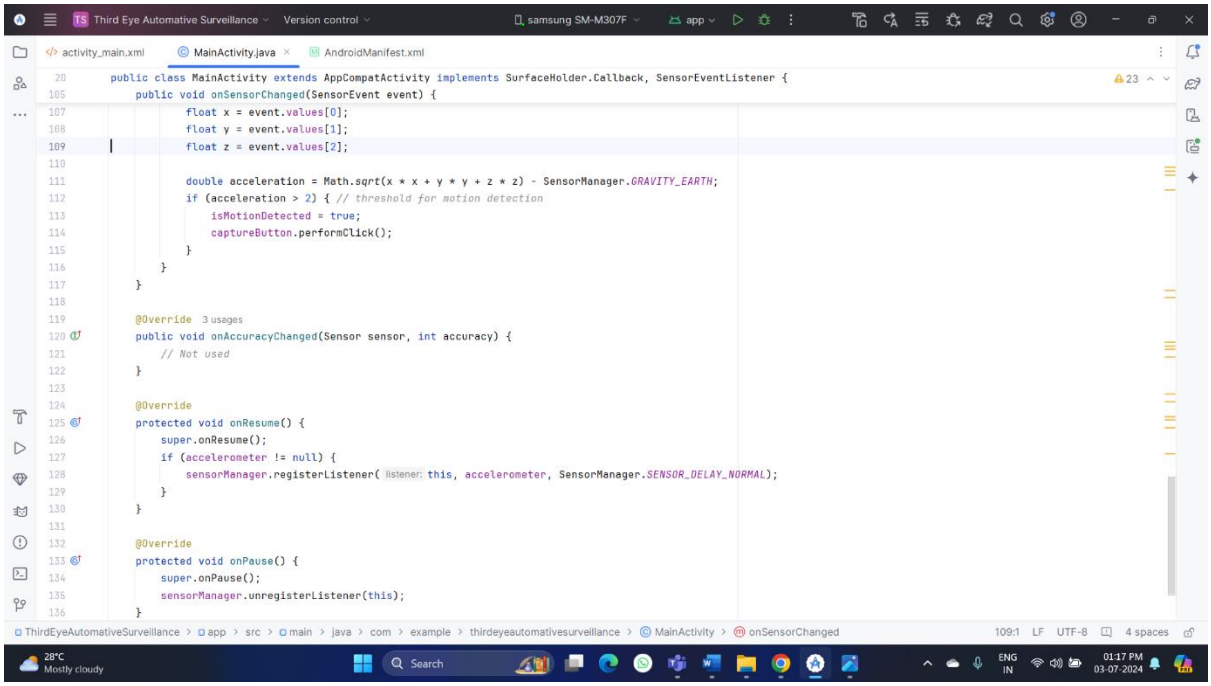
Task 4: Implement basic motion detection algorithms that trigger photo capture when movement is detected around the vehicle.

Update AndroidManifest.xml

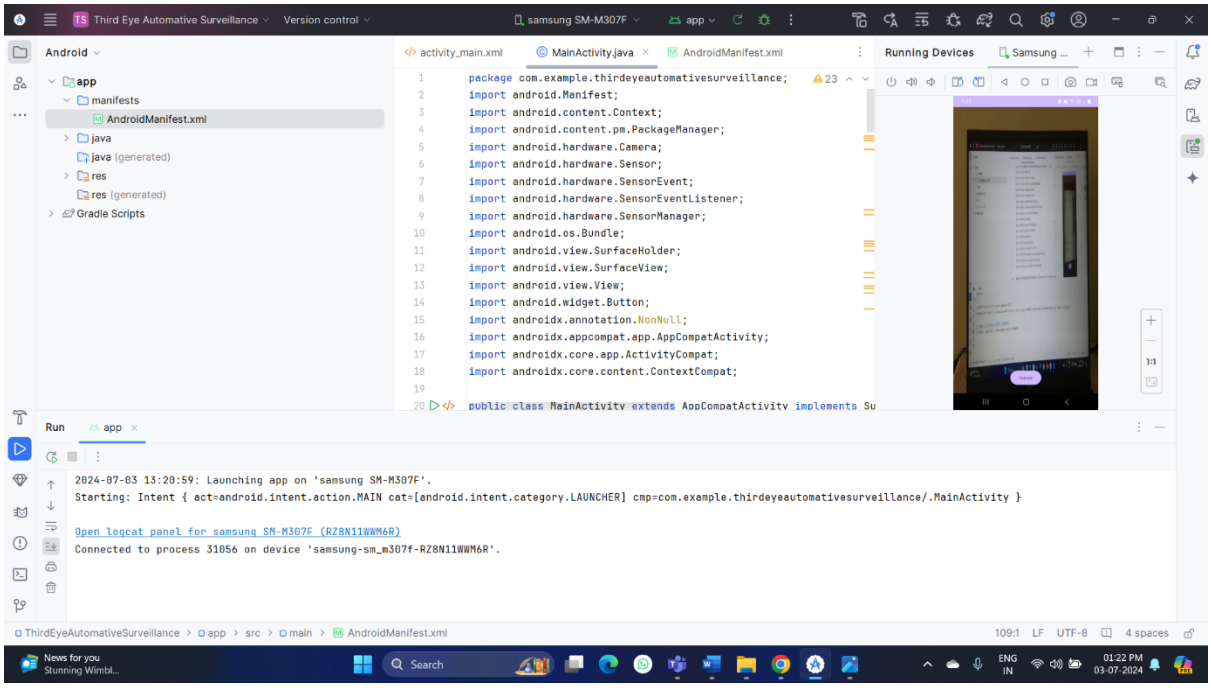


Update MainActivity.java (Extended with Motion Detection):





DAY 1 TASK 4 OUTPUT SCREENSHOTS



Day 2: Vibration Detection and Event-Driven Capture

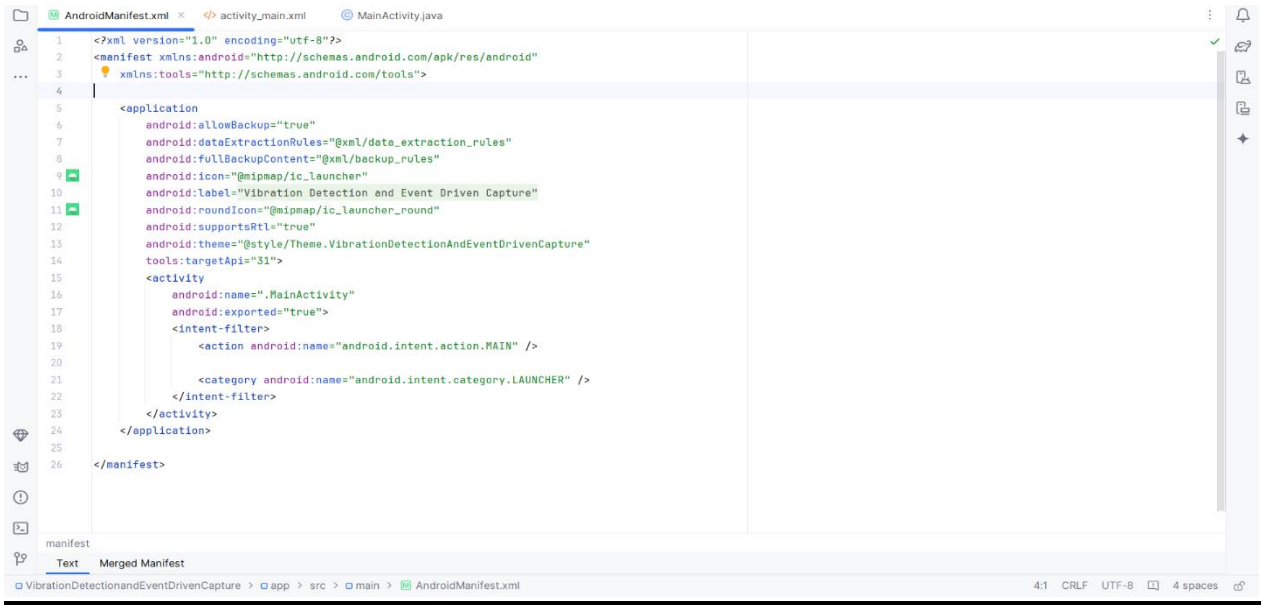
Objective: Implement a robust system to detect vehicle vibrations, capture images during vibrations, process and compress images for efficient storage, and store images with essential metadata.

Task 1: Develop a system to detect vibrations using the vehicle's built-in sensors or external hardware connected to an Android device.

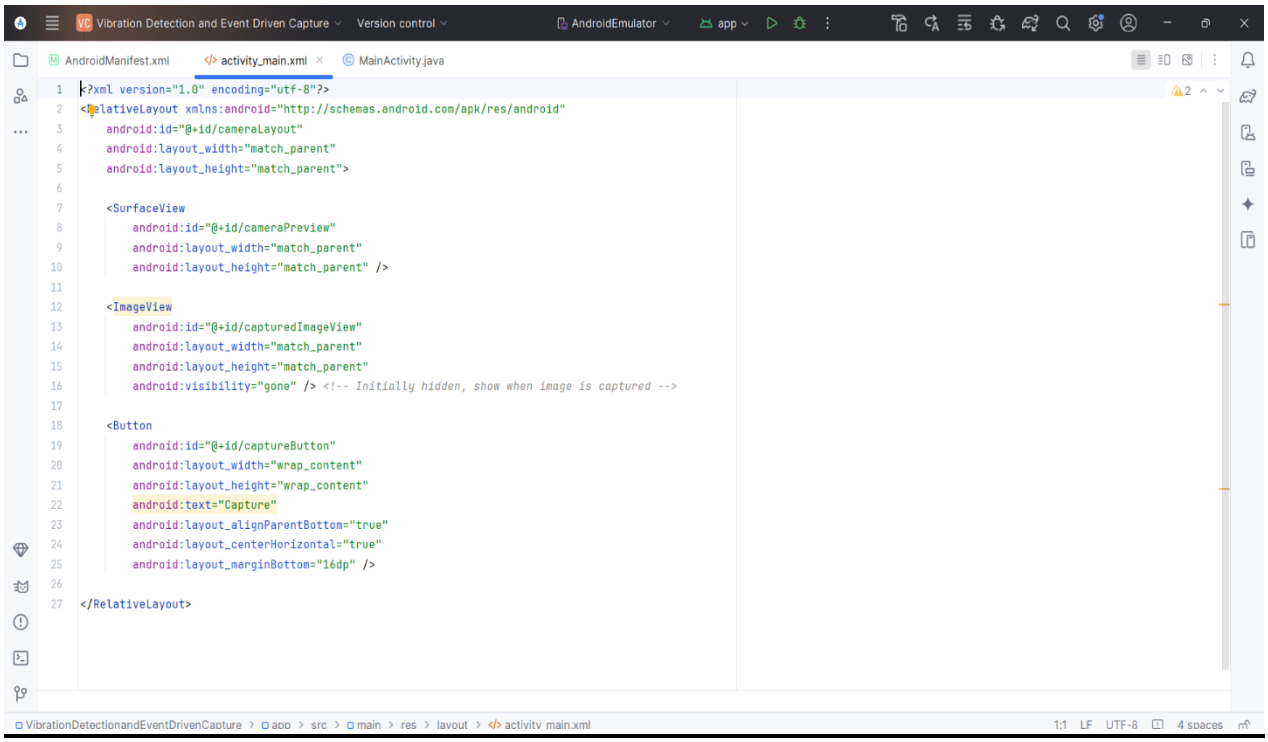
Vibration Detection Using Sensors

Vibration detection utilizes the accelerometer sensor available in Android devices. This sensor measures acceleration along three axes (x, y, z), enabling the detection of significant vibrations beyond a predefined threshold. The system continuously monitors accelerometer data to identify potential security incidents such as impacts or break-ins.

AndroidManifest.xml

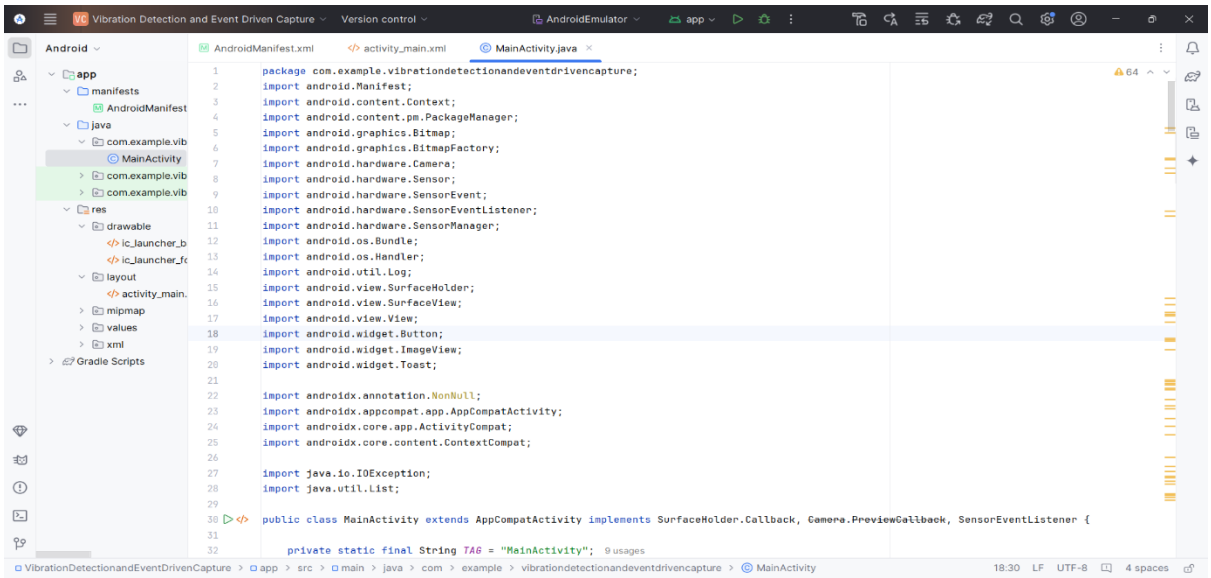


activity_main.xml

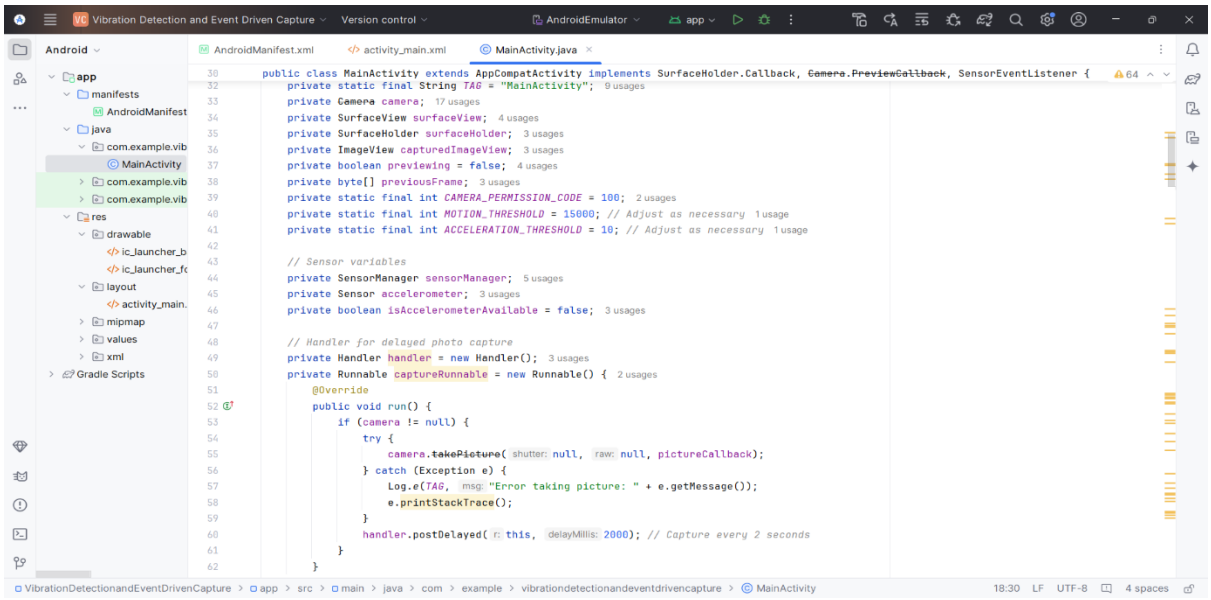


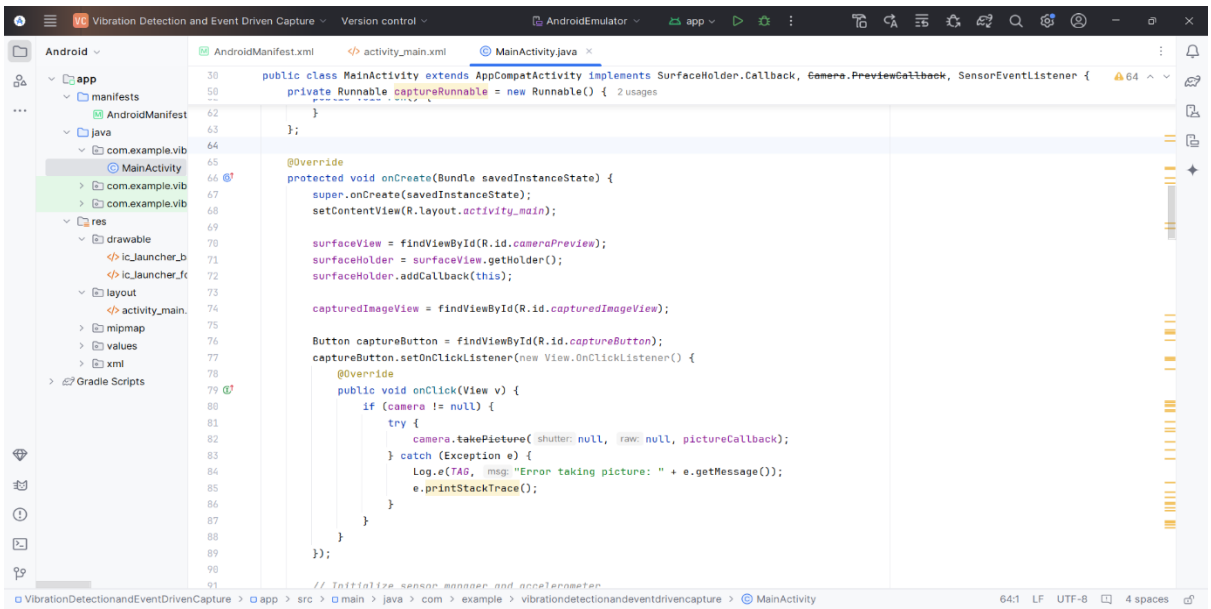
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/cameraLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <SurfaceView
        android:id="@+id/cameraPreview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <ImageView
        android:id="@+id/capturedImageView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="gone" /> <!-- Initially hidden, show when image is captured -->

    <Button
        android:id="@+id/captureButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Capture"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="16dp" />

</RelativeLayout>
```

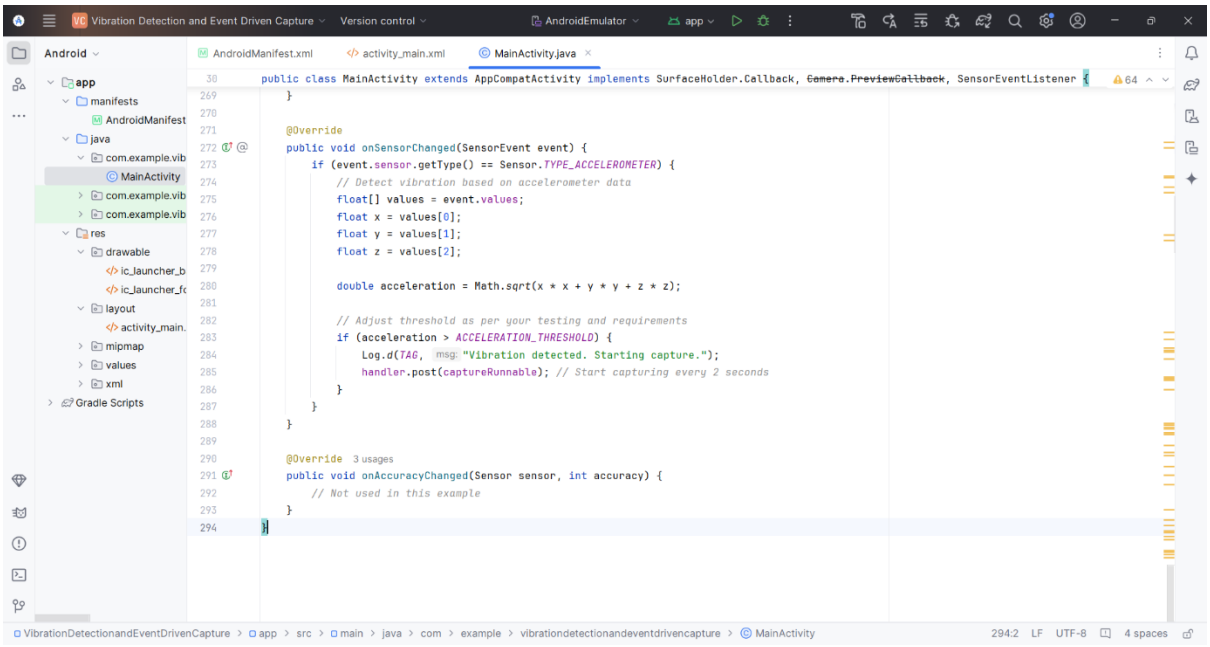
MainActivity.java



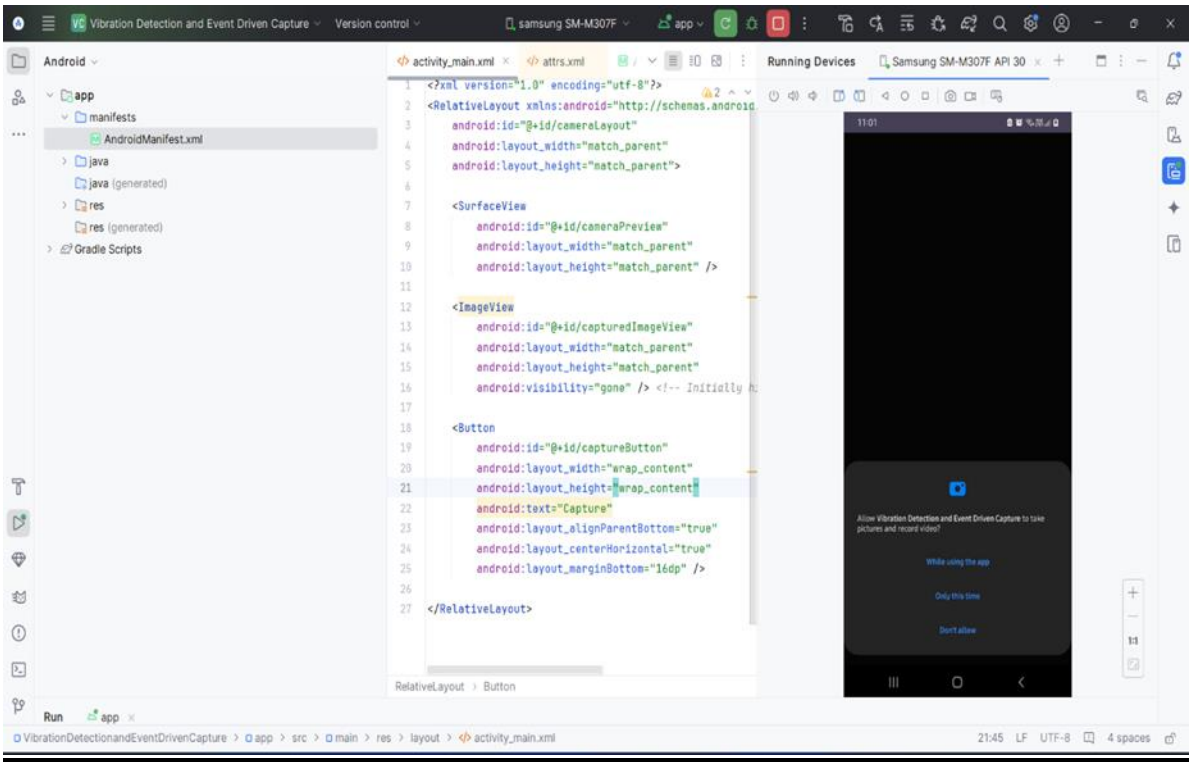
```
package com.example.vibrationdetectionandeventdrivencapture;
import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.hardware.Camera;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;

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

import java.io.IOException;
import java.util.List;

public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {

    private static final String TAG = "MainActivity"; 9 usages
```

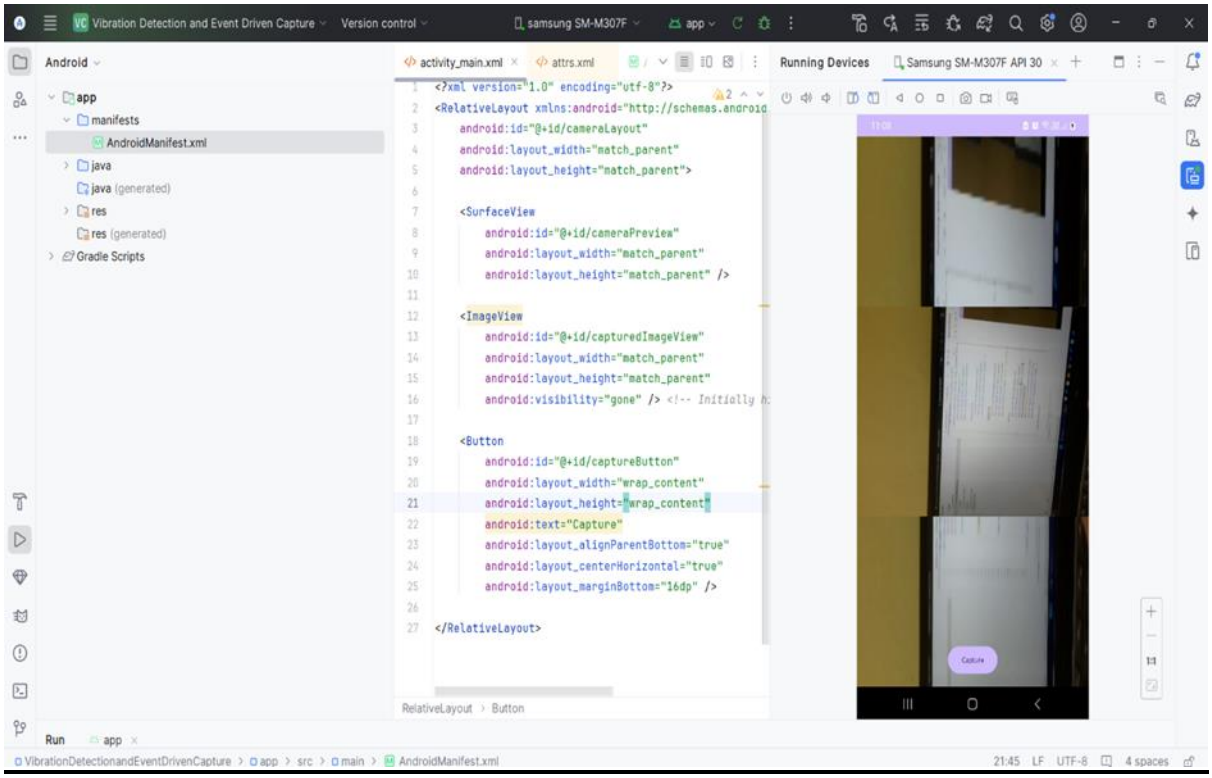

Vibration Detection and Event Driven Capture
Version control
AndroidEmulator
app
Android
AndroidManifest.xml
activity_main.xml
MainActivity.java
app
manifests
AndroidManifest
java
com.example.vib
MainActivity
com.example.vib
com.example.vib
res
drawable
ic_launcher_b
ic_launcher_fc
layout
activity_main.
mipmap
values
xml
Gradle Scripts
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {
private static final String TAG = "MainActivity"; 9 usages
private Camera camera; 17 usages
private SurfaceView surfaceView; 4 usages
private SurfaceHolder surfaceHolder; 3 usages
private ImageView capturedImageView; 3 usages
private boolean previewing = false; 4 usages
private byte[] previousFrame; 3 usages
private static final int CAMERA_PERMISSION_CODE = 100; 2 usages
private static final int MOTION_THRESHOLD = 15000; // Adjust as necessary 1 usage
private static final int ACCELERATION_THRESHOLD = 10; // Adjust as necessary 1 usage
// Sensor variables
private SensorManager sensorManager; 5 usages
private Sensor accelerometer; 3 usages
private boolean isAccelerometerAvailable = false; 3 usages
// Handler for delayed photo capture
private Handler handler = new Handler(); 3 usages
private Runnable captureRunnable = new Runnable() { 2 usages
@Override
public void run() {
if (camera != null) {
try {
camera.takePicture(shutter: null, raw: null, pictureCallback);
} catch (Exception e) {
Log.e(TAG, msg: "Error taking picture: " + e.getMessage());
e.printStackTrace();
}
handler.postDelayed(r: this, delayMillis: 2000); // Capture every 2 seconds
}
}
VibrationDetectionandEventDrivenCapture
app
src
main
java
com
example
vibrationdetectionandeventdrivencapture
MainActivity
18:30
LF
UTF-8
4 spaces


Vibration Detection and Event Driven Capture
Version control
AndroidEmulator
app
Android
AndroidManifest.xml
activity_main.xml
MainActivity.java
app
manifests
AndroidManifest
java
com.example.vib
MainActivity
com.example.vib
com.example.vib
res
drawable
ic_launcher_b
ic_launcher_fc
layout
activity_main.
mipmap
values
xml
Gradle Scripts
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {
private Runnable captureRunnable = new Runnable() { 2 usages
};
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
surfaceView = findViewById(R.id.cameraPreview);
surfaceHolder = surfaceView.getHolder();
surfaceHolder.addCallback(this);
capturedImageView = findViewById(R.id.capturedImageView);
Button captureButton = findViewById(R.id.captureButton);
captureButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
if (camera != null) {
try {
camera.takePicture(shutter: null, raw: null, pictureCallback);
} catch (Exception e) {
Log.e(TAG, msg: "Error taking picture: " + e.getMessage());
e.printStackTrace();
}
}
}
});
// Initialize sensor manager and accelerometer
VibrationDetectionandEventDrivenCapture
app
src
main
java
com
example
vibrationdetectionandeventdrivencapture
MainActivity
64:1
LF
UTF-8
4 spaces



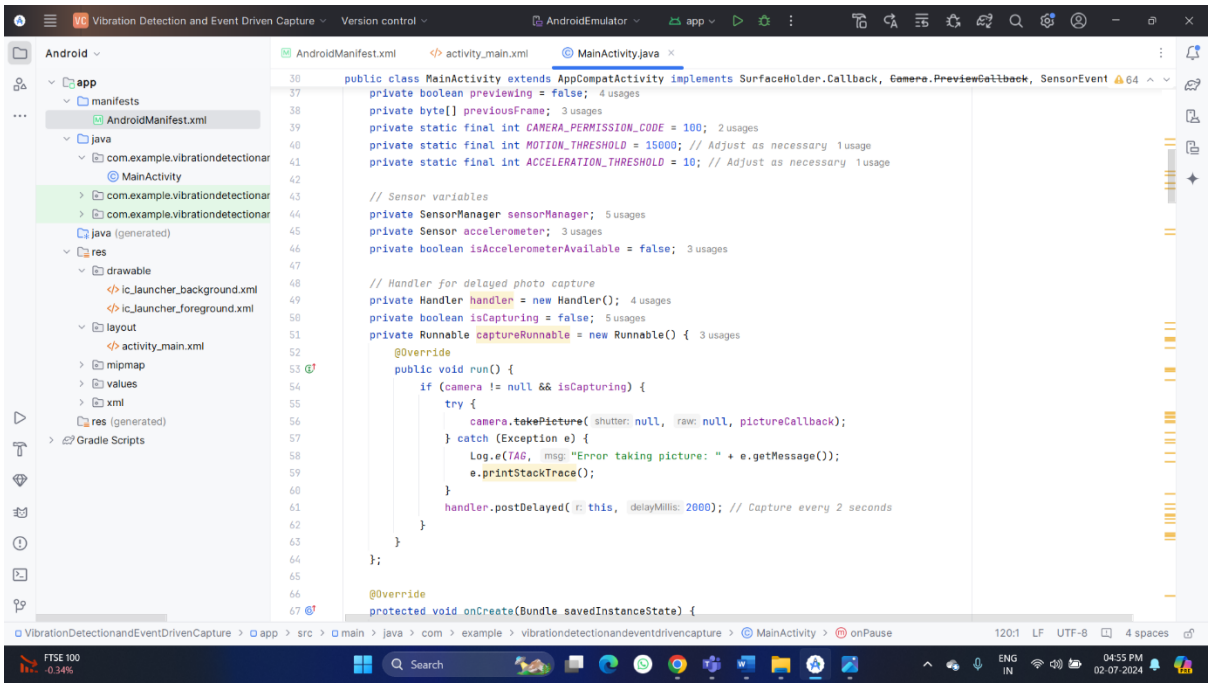
DAY 2 TASK 1 OUTPUT SCREENSHOTS

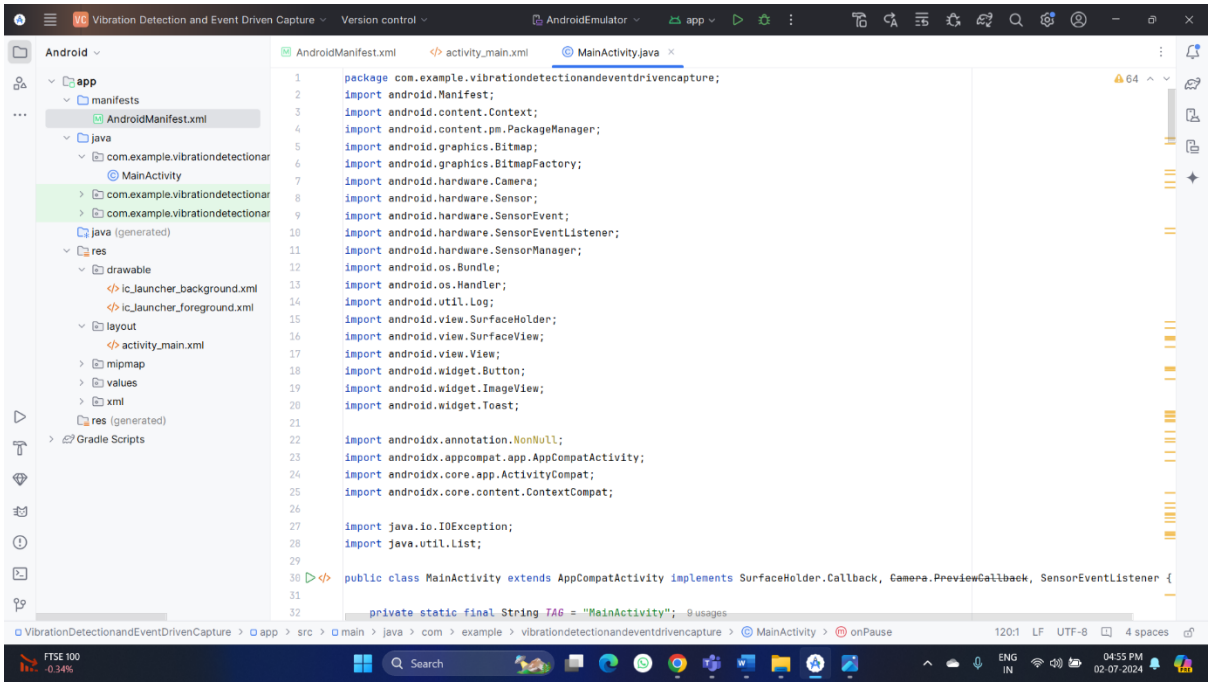


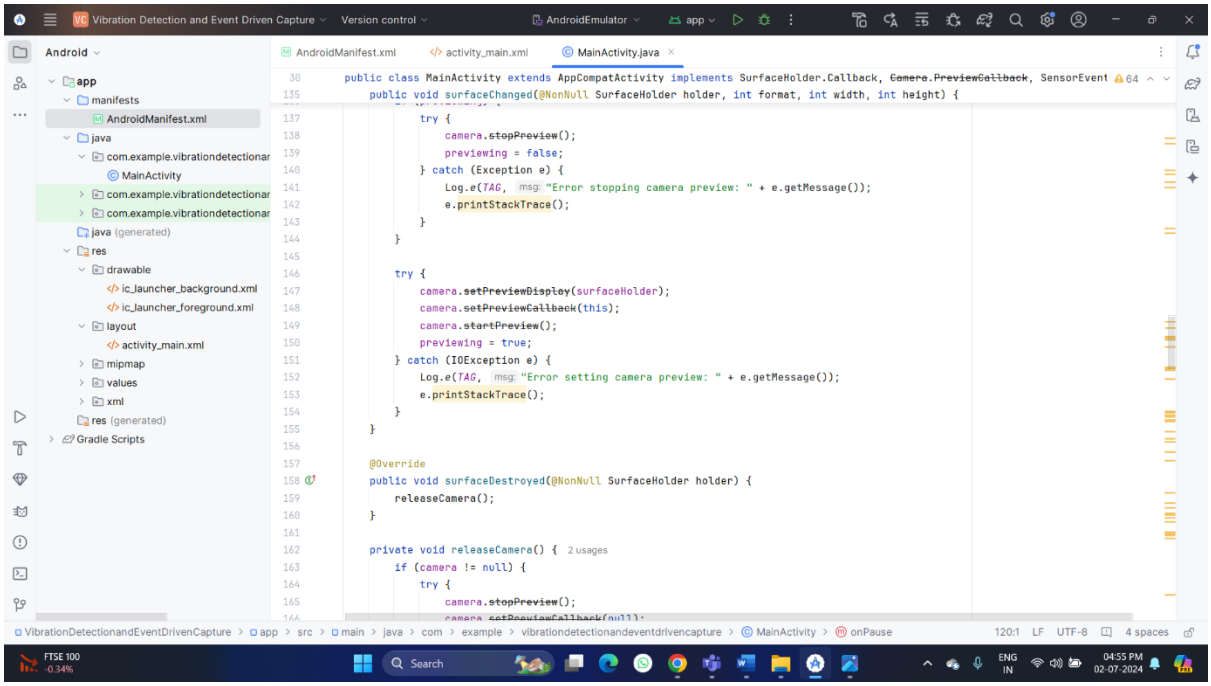


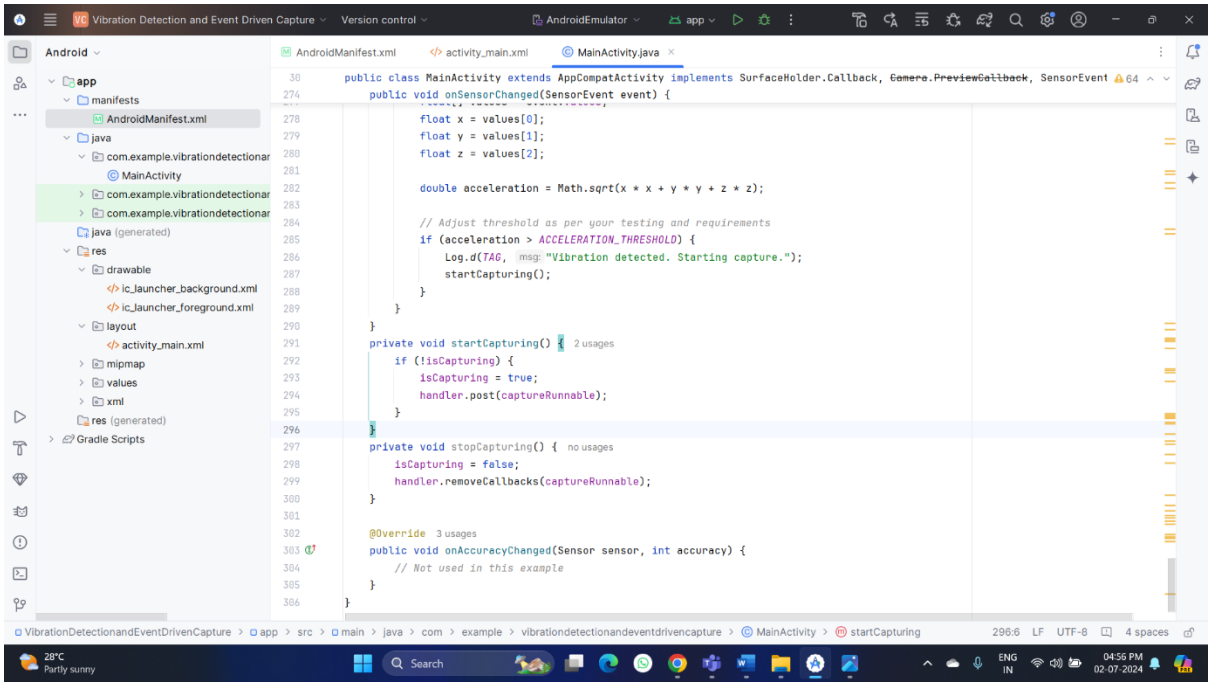
Task 2: Set up an event-driven mechanism that initiates the camera to take photos every 2 seconds when vibrations are detected while parked.

Update the MainActivity.java

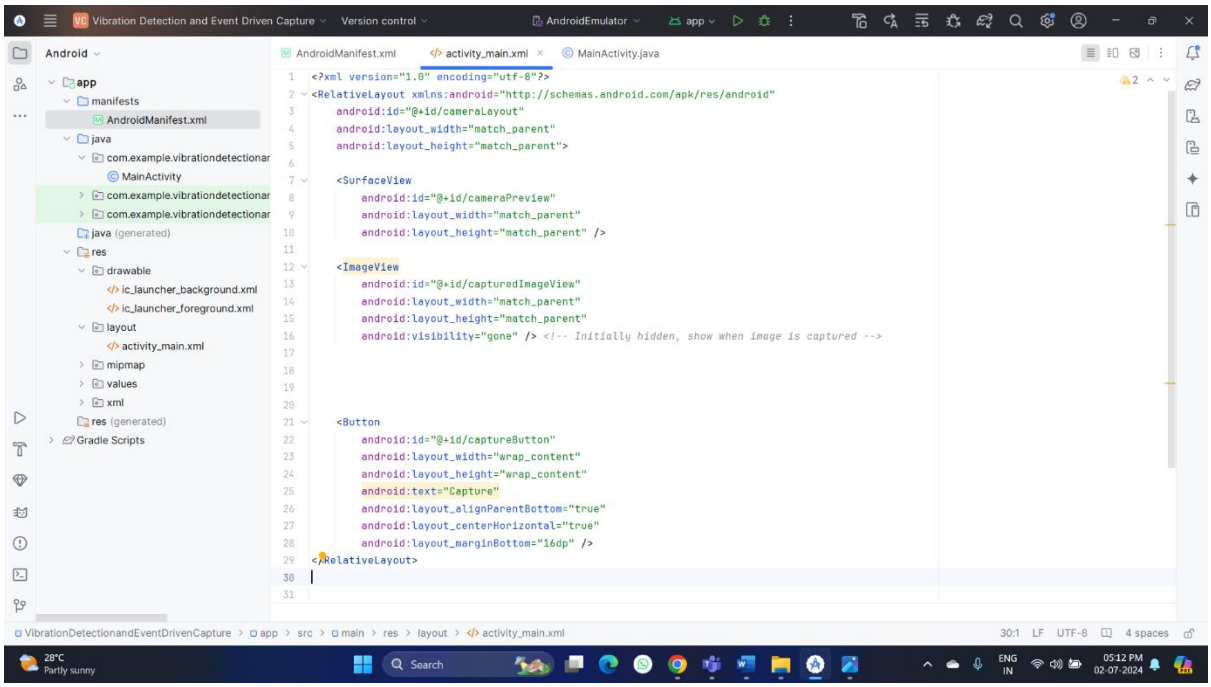



Vibration Detection and Event Driven Capture
Version control
AndroidEmulator
app
Android
AndroidManifest.xml
activity_main.xml
MainActivity.java
app
manifests
AndroidManifest.xml
java
com.example.vibrationdetectionar
MainActivity
com.example.vibrationdetectionar
com.example.vibrationdetectionar
java (generated)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml
layout
activity_main.xml
mipmap
values
xml
res (generated)
Gradle Scripts
package com.example.vibrationdetectionandeventdrivencapture;
import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.hardware.Camera;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import java.io.IOException;
import java.util.List;
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {
private static final String TAG = "MainActivity";
9 usages
VibrationDetectionandEventDrivenCapture
app
src
main
java
com
example
vibrationdetectionandeventdrivencapture
MainActivity
onPause
120:1
LF
UTF-8
4 spaces
FTSE 100
-0.34%
Search
ENG
IN
04:55 PM
02-07-2024

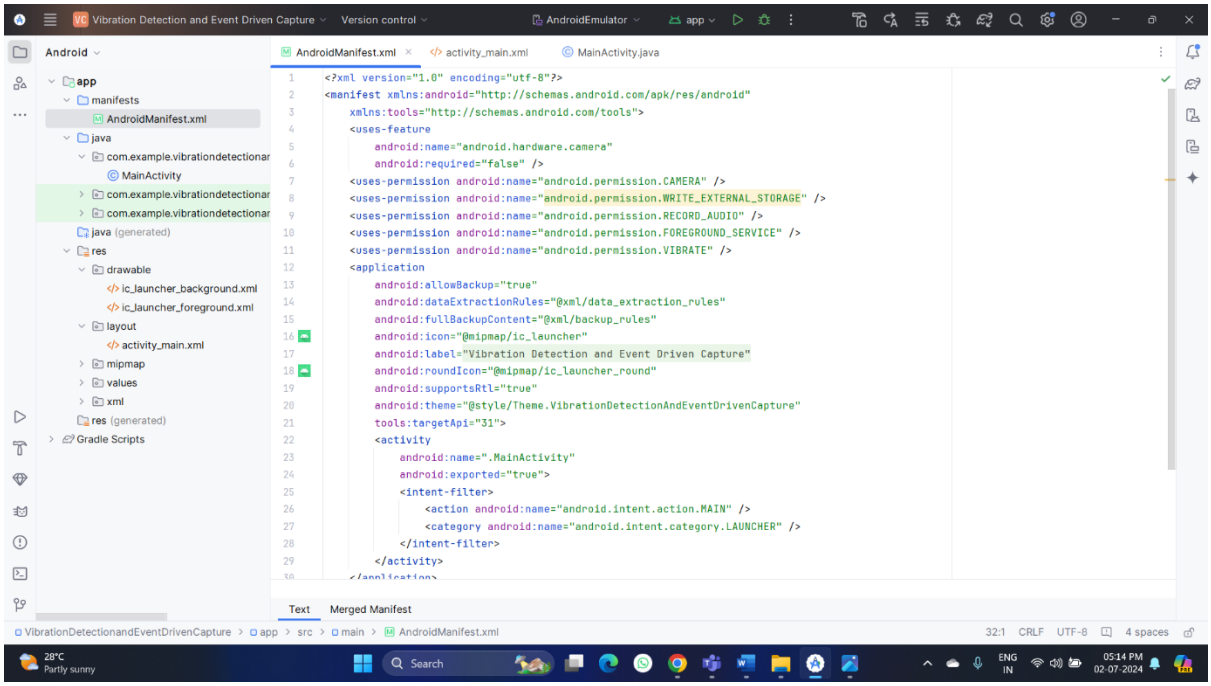

Vibration Detection and Event Driven Capture
Version control
AndroidEmulator
app
Android
AndroidManifest.xml
activity_main.xml
MainActivity.java
app
manifests
AndroidManifest.xml
java
com.example.vibrationdetectionar
MainActivity
com.example.vibrationdetectionar
com.example.vibrationdetectionar
java (generated)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml
layout
activity_main.xml
mipmap
values
xml
res (generated)
Gradle Scripts
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEvent
public void surfaceChanged(@NonNull SurfaceHolder holder, int format, int width, int height) {
try {
camera.stopPreview();
previewing = false;
} catch (Exception e) {
Log.e(TAG, msg: "Error stopping camera preview: " + e.getMessage());
e.printStackTrace();
}
}
try {
camera.setPreviewDisplay(surfaceHolder);
camera.setPreviewCallback(this);
camera.startPreview();
previewing = true;
} catch (IOException e) {
Log.e(TAG, msg: "Error setting camera preview: " + e.getMessage());
e.printStackTrace();
}
}
@Override
public void surfaceDestroyed(@NonNull SurfaceHolder holder) {
releaseCamera();
}
private void releaseCamera() { 2 usages
if (camera != null) {
try {
camera.stopPreview();
VibrationDetectionandEventDrivenCapture
app
src
main
java
com
example
vibrationdetectionandeventdrivencapture
MainActivity
onPause
120:1
LF
UTF-8
4 spaces
FTSE 100
-0.34%
Search
ENG
IN
04:55 PM
02-07-2024



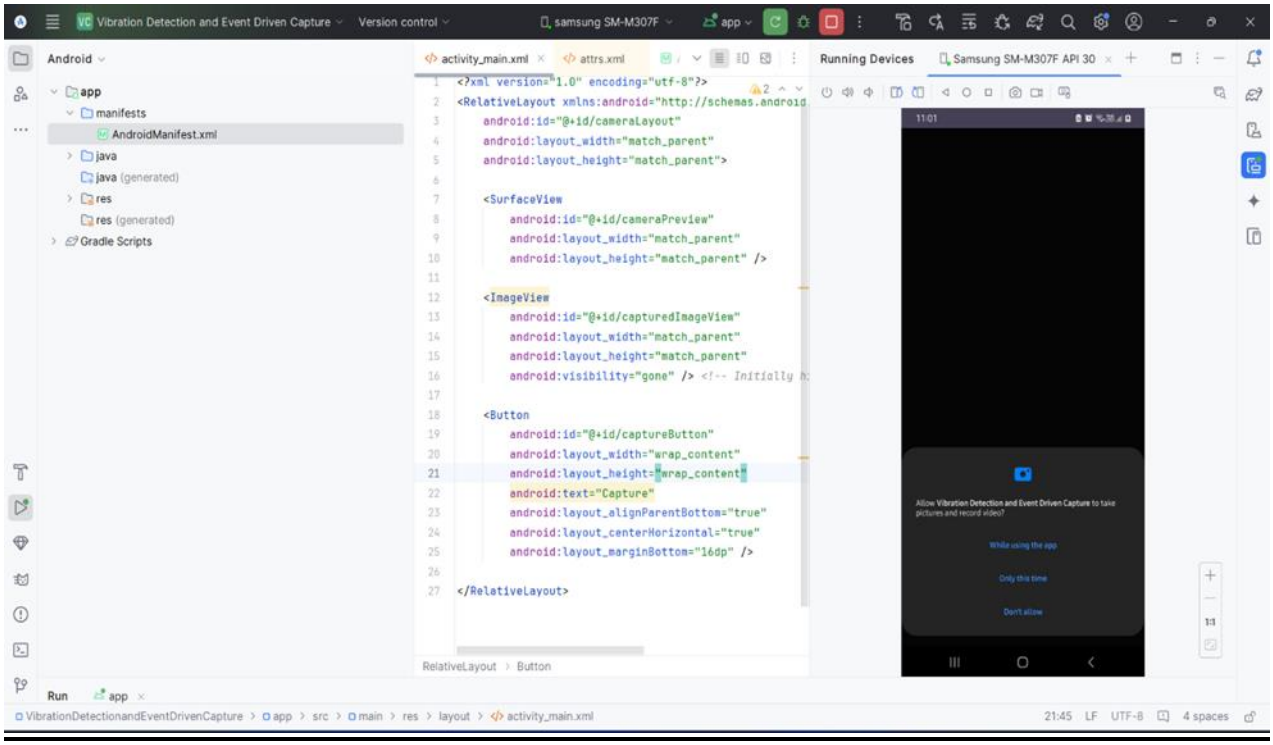
Update the activity_main.xml

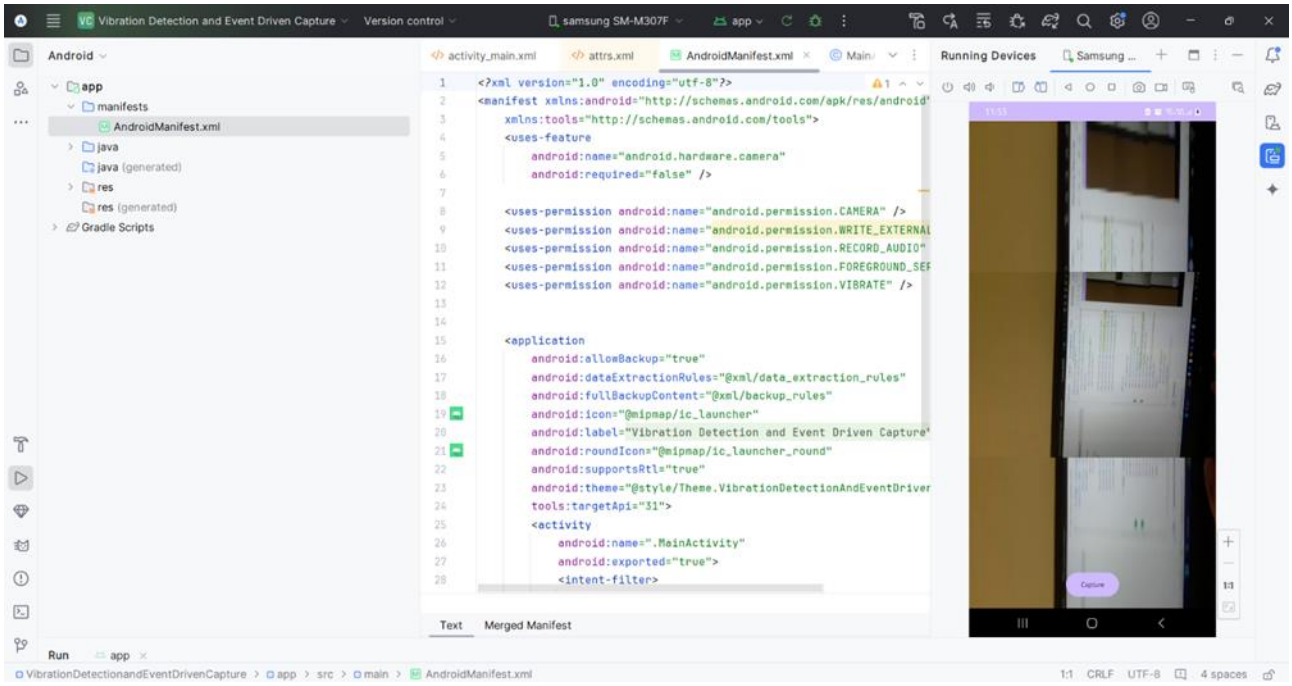


Update the AndroidManifest.xml



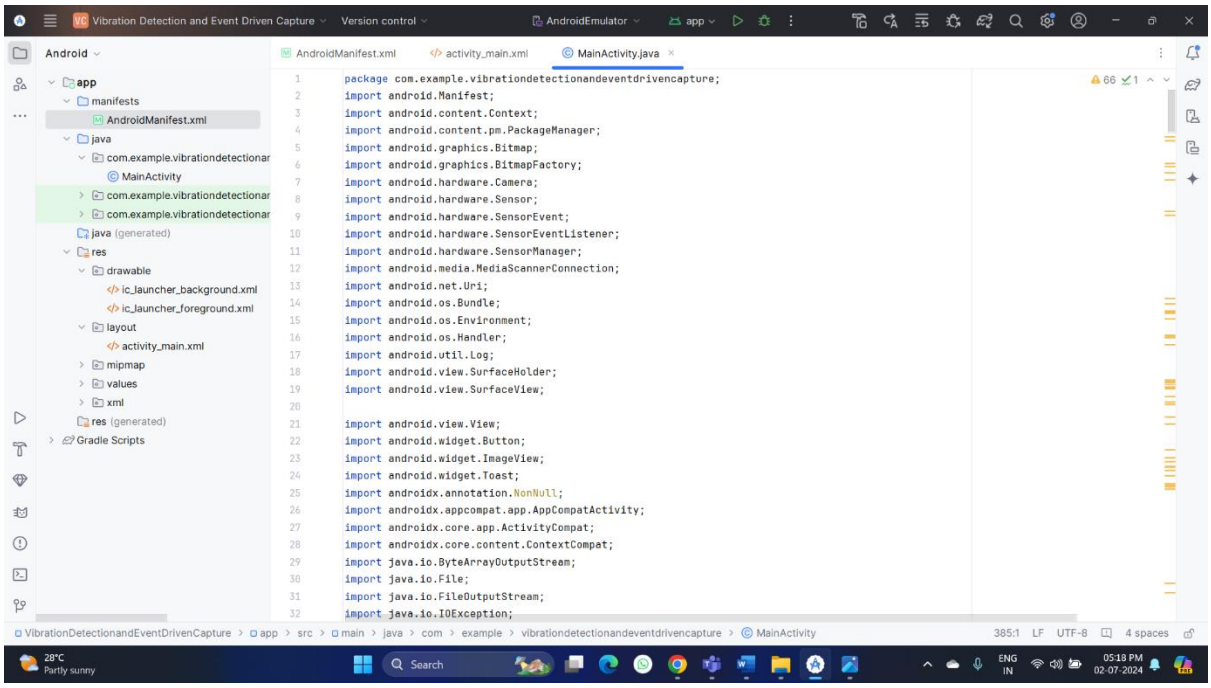
DAY 2 TASK 2 OUTPUT SCREENSHOTS

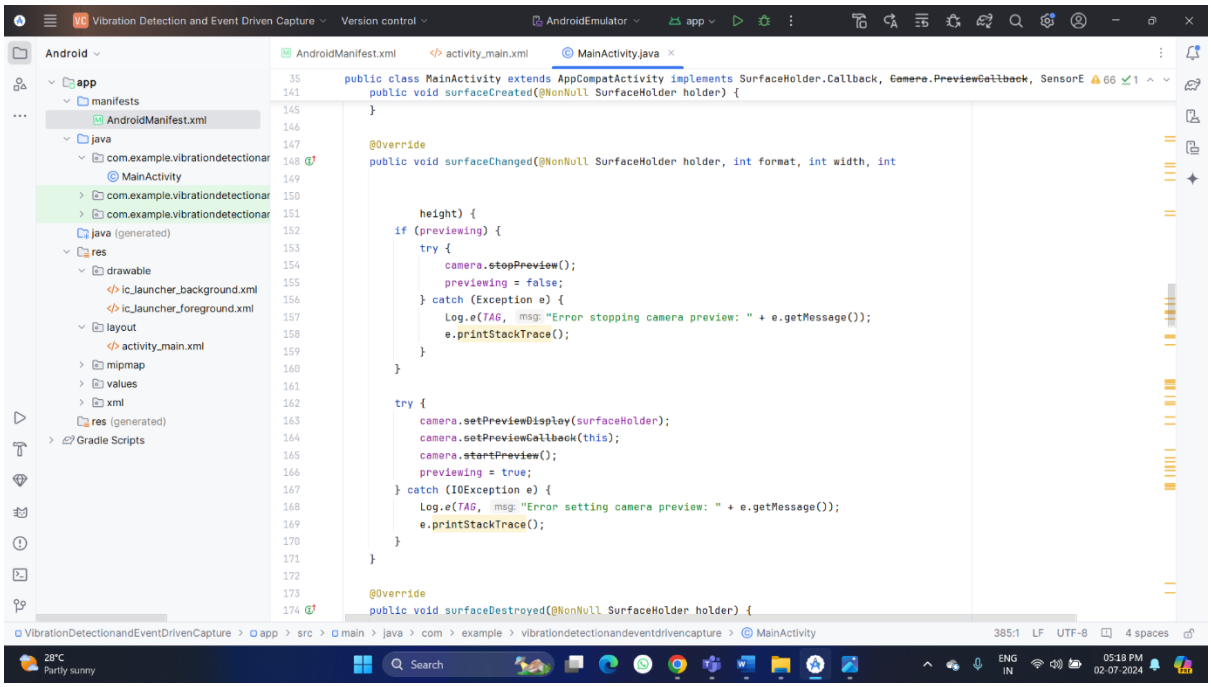


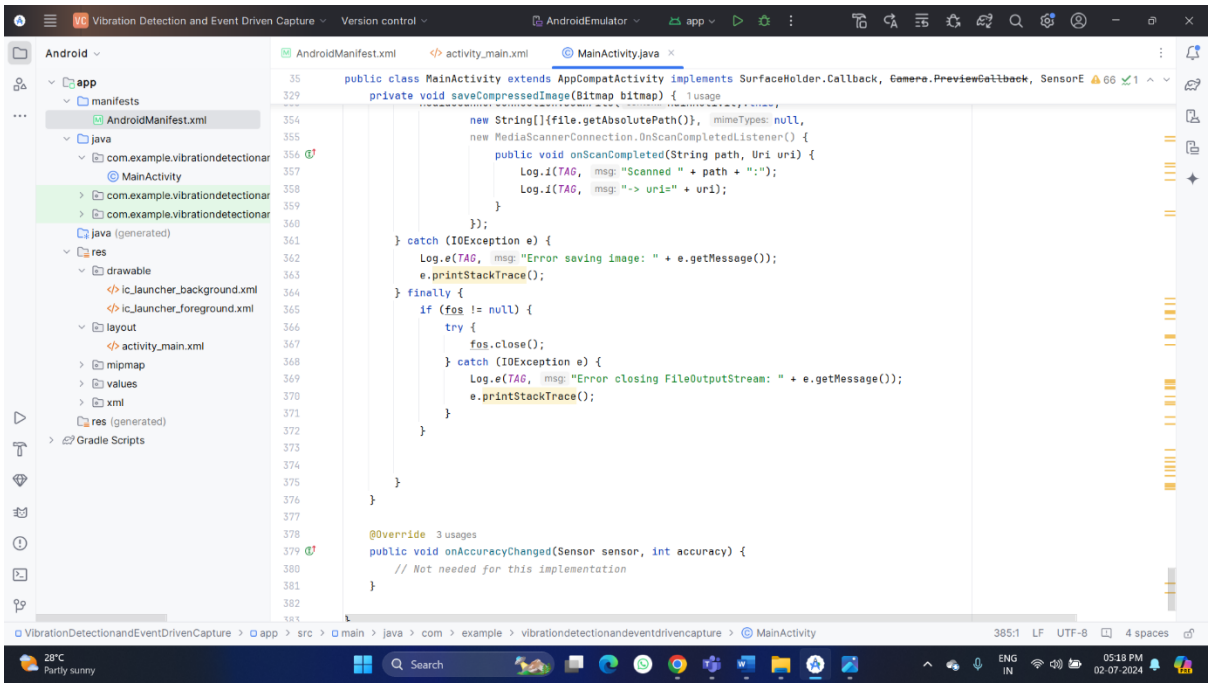


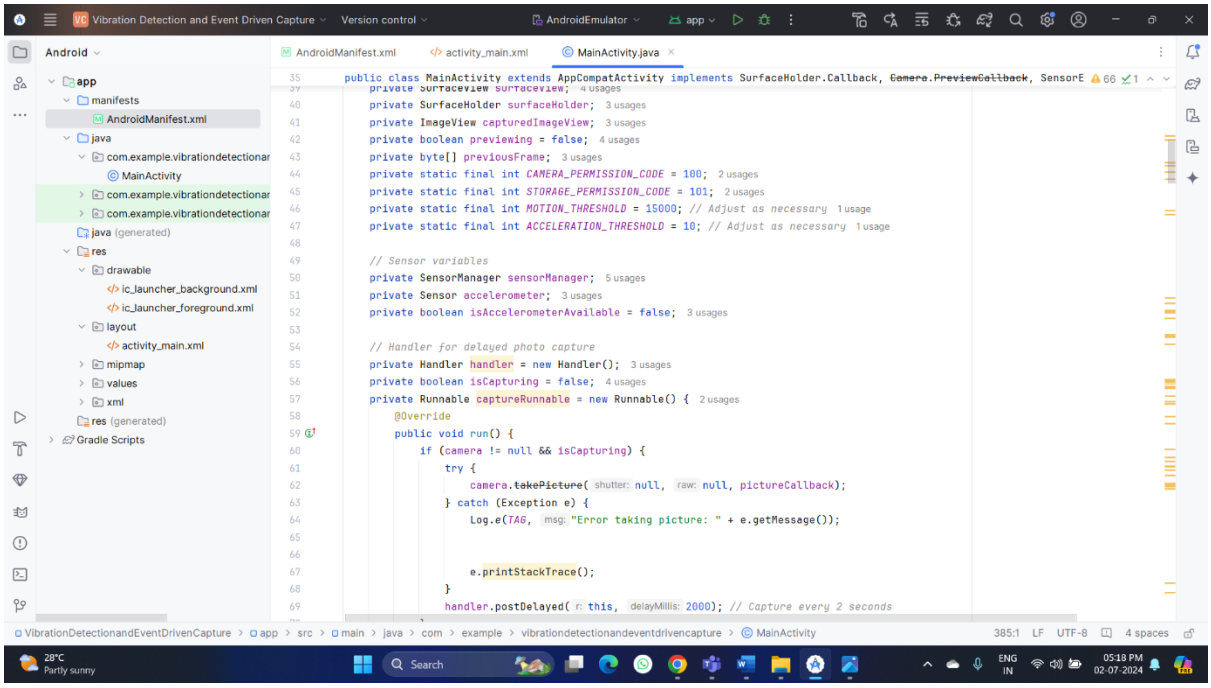
Task 3: Process and compress images captured by the camera to reduce storage space without compromising quality.

Update the MainActivity.Java

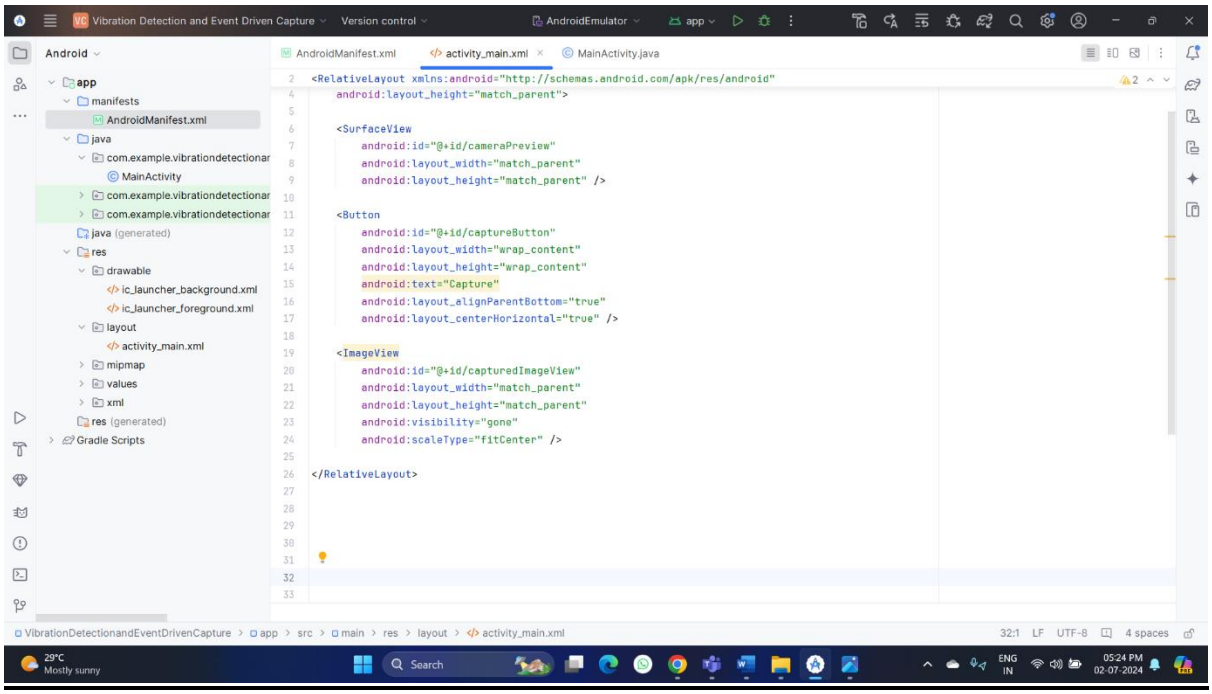



Vibration Detection and Event Driven Capture
Version control
AndroidEmulator
app
Android
AndroidManifest.xml
activity_main.xml
MainActivity.java
app
manifests
AndroidManifest.xml
java
com.example.vibrationdetectionar
MainActivity
com.example.vibrationdetectionar
com.example.vibrationdetectionar
java (generated)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml
layout
activity_main.xml
mipmap
values
xml
res (generated)
Gradle Scripts
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorE
public void surfaceCreated(@NonNull SurfaceHolder holder) {
}
@Override
public void surfaceChanged(@NonNull SurfaceHolder holder, int format, int width, int
height) {
if (previewing) {
try {
camera.stopPreview();
previewing = false;
} catch (Exception e) {
Log.e(TAG, msg: "Error stopping camera preview: " + e.getMessage());
e.printStackTrace();
}
}
try {
camera.setPreviewDisplay(surfaceHolder);
camera.setPreviewCallback(this);
camera.startPreview();
previewing = true;
} catch (IOException e) {
Log.e(TAG, msg: "Error setting camera preview: " + e.getMessage());
e.printStackTrace();
}
}
@Override
public void surfaceDestroyed(@NonNull SurfaceHolder holder) {
VibrationDetectionandEventDrivenCapture
app
src
main
java
com
example
vibrationdetectionandeventdrivencapture
MainActivity
385:1
LF
UTF-8
4 spaces
28°C
Partly sunny
Search
ENG
IN
05:18 PM
02-07-2024

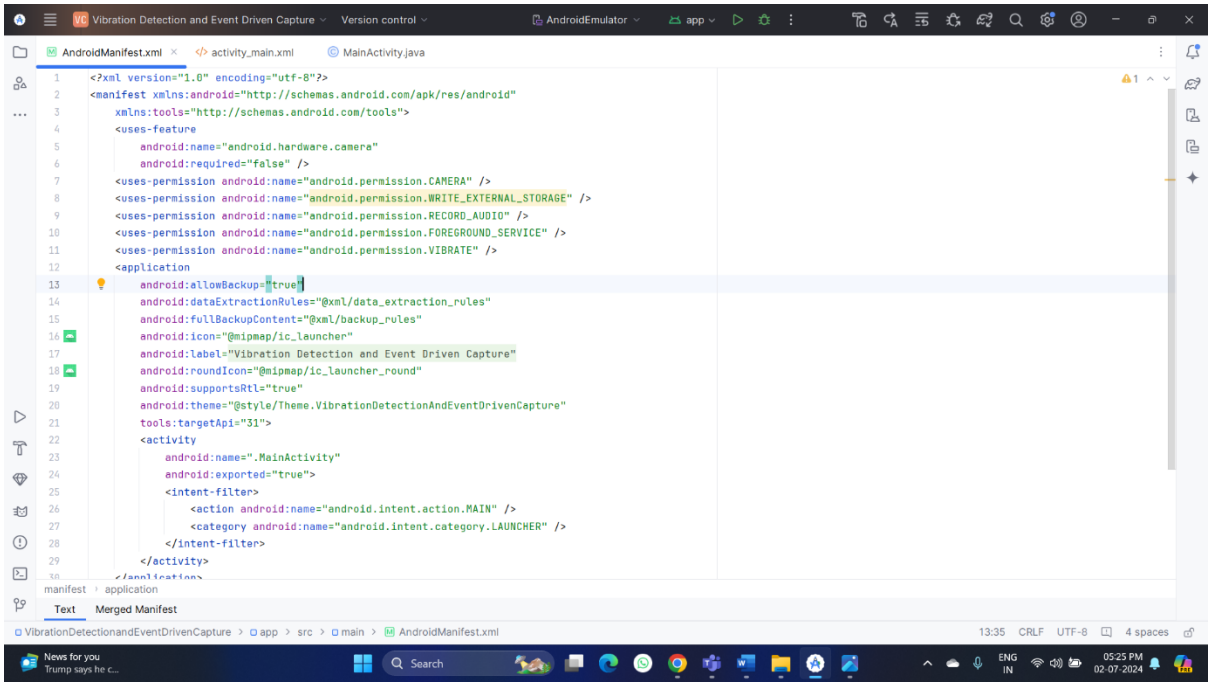

Vibration Detection and Event Driven Capture
Version control
AndroidEmulator
app
Android
AndroidManifest.xml
activity_main.xml
MainActivity.java
app
manifests
AndroidManifest.xml
java
com.example.vibrationdetectionar
MainActivity
com.example.vibrationdetectionar
com.example.vibrationdetectionar
java (generated)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml
layout
activity_main.xml
mipmap
values
xml
res (generated)
Gradle Scripts
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorE
private void saveCompressedImage(Bitmap bitmap) { 1 usage
new String[]{file.getAbsolutePath()}, mimeTypes: null,
new MediaScannerConnection.OnScanCompletedListener() {
public void onScanCompleted(String path, Uri uri) {
Log.i(TAG, msg: "Scanned " + path + ":");
Log.i(TAG, msg: "-> uri=" + uri);
}
});
} catch (IOException e) {
Log.e(TAG, msg: "Error saving image: " + e.getMessage());
e.printStackTrace();
} finally {
if (fos != null) {
try {
fos.close();
} catch (IOException e) {
Log.e(TAG, msg: "Error closing FileOutputStream: " + e.getMessage());
e.printStackTrace();
}
}
}
}
@Override 3 usages
public void onAccuracyChanged(Sensor sensor, int accuracy) {
// Not needed for this implementation
}
VibrationDetectionandEventDrivenCapture
app
src
main
java
com
example
vibrationdetectionandeventdrivencapture
MainActivity
385:1
LF
UTF-8
4 spaces
28°C
Partly sunny
Search
ENG
IN
05:18 PM
02-07-2024



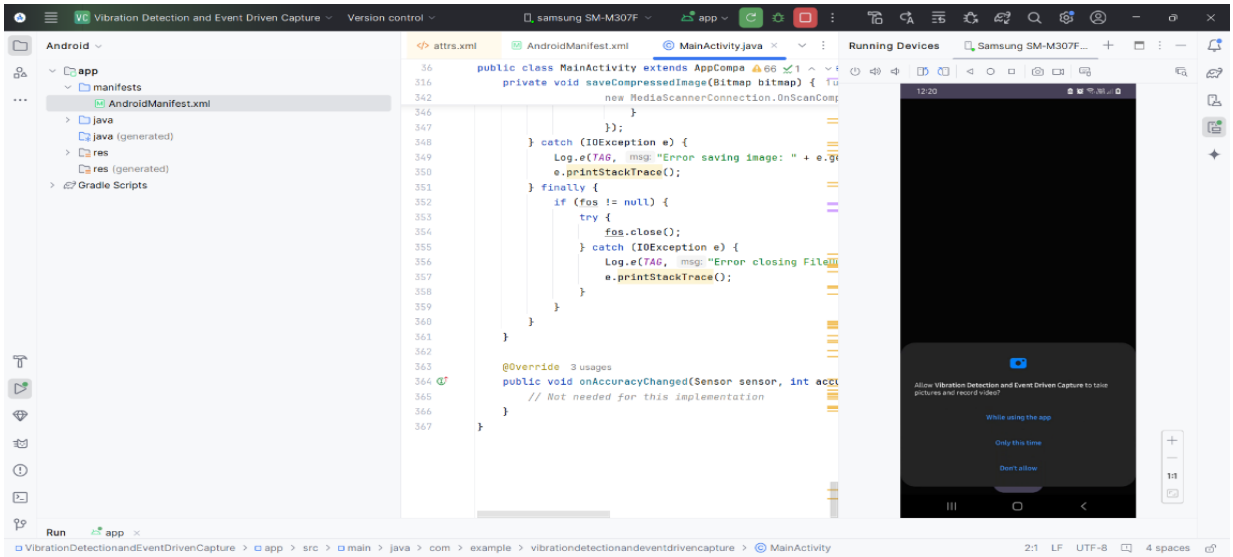
Update the activity_main.xml

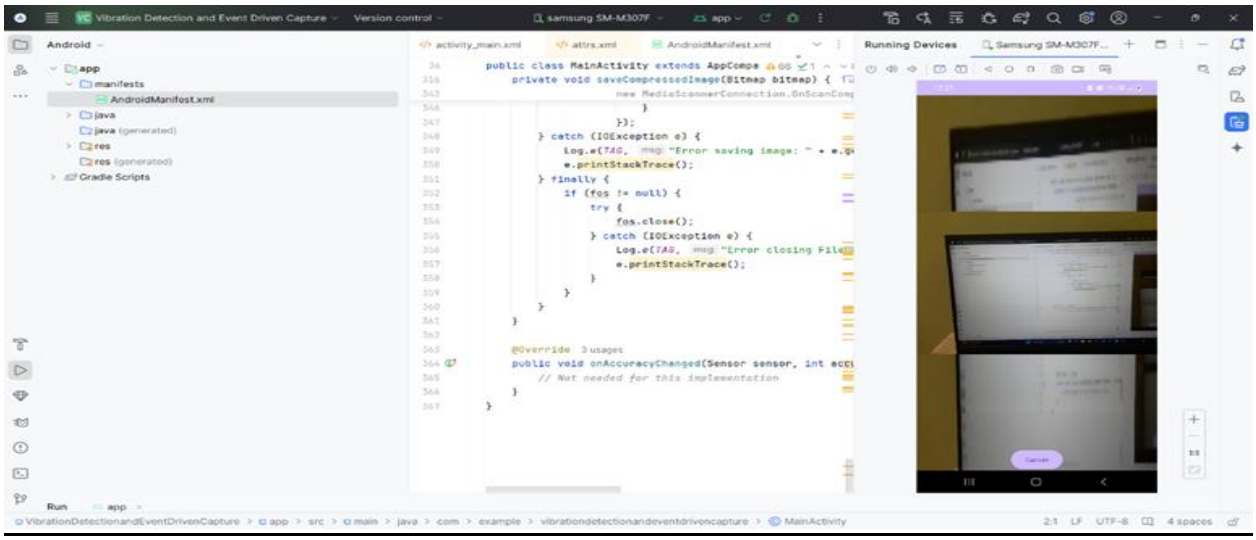


Update the AndroidManifest.xml



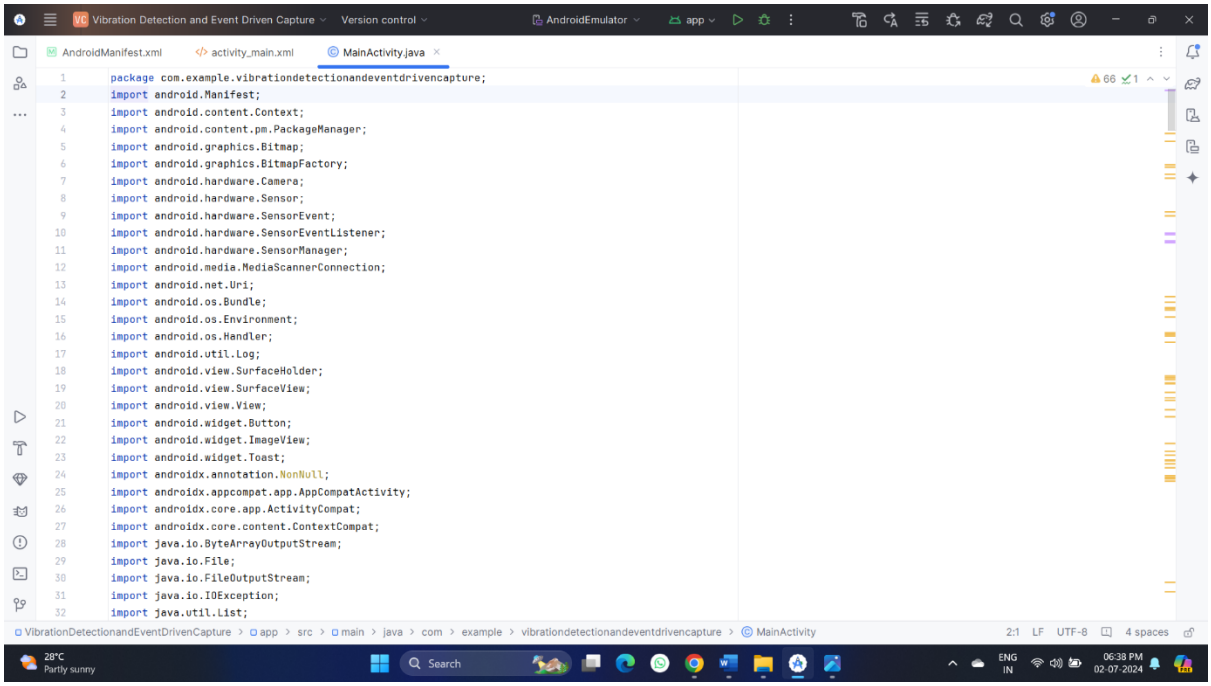
DAY 2 TASK 3 OUTPUT SCREENSHOTS

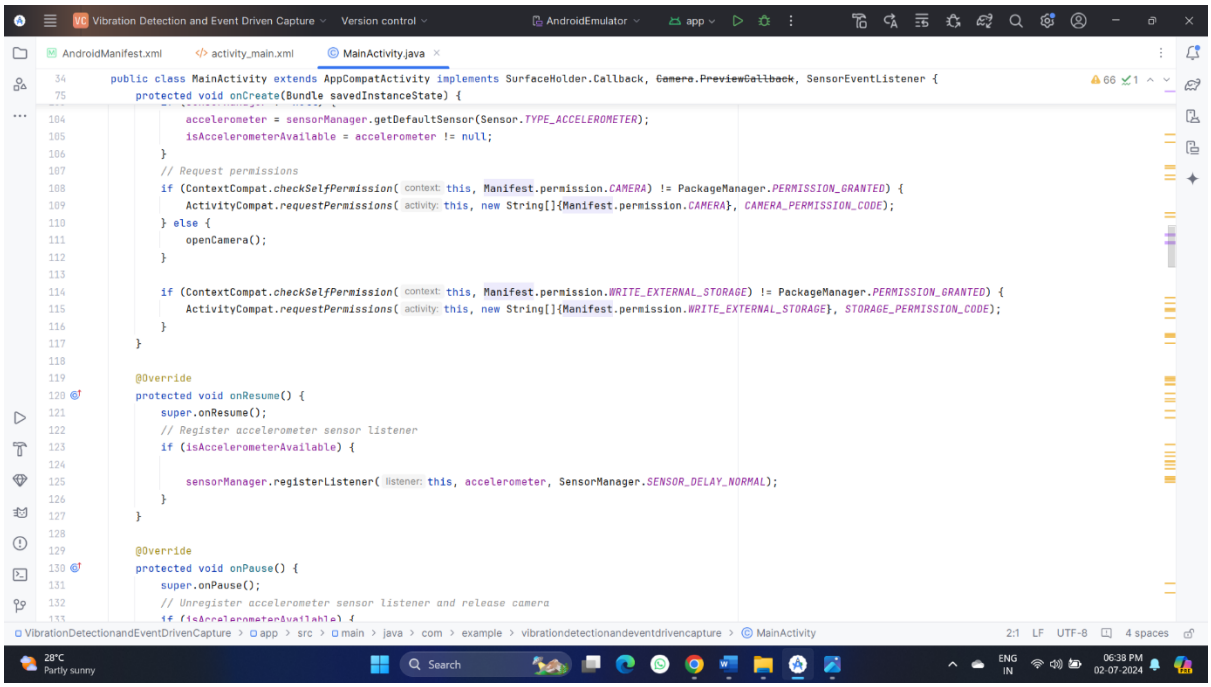




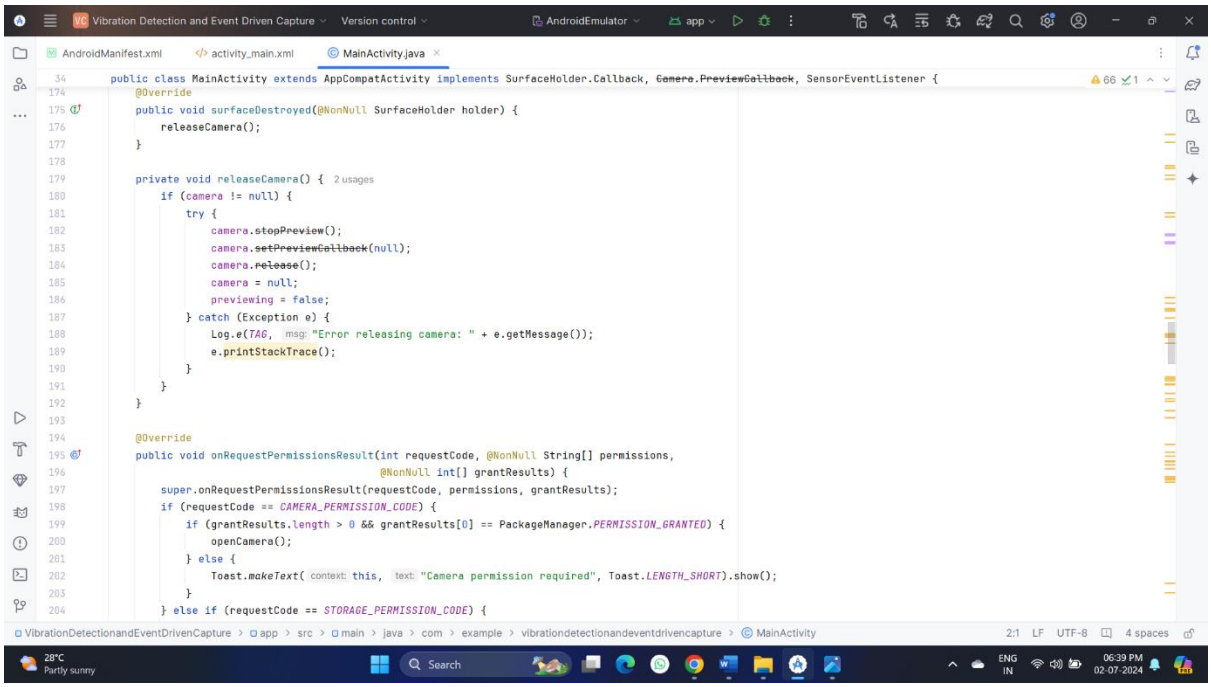
Task 4: Efficiently store the captured images and videos locally with a timestamp and geo-tagging information.

Update the MainActivity.Java

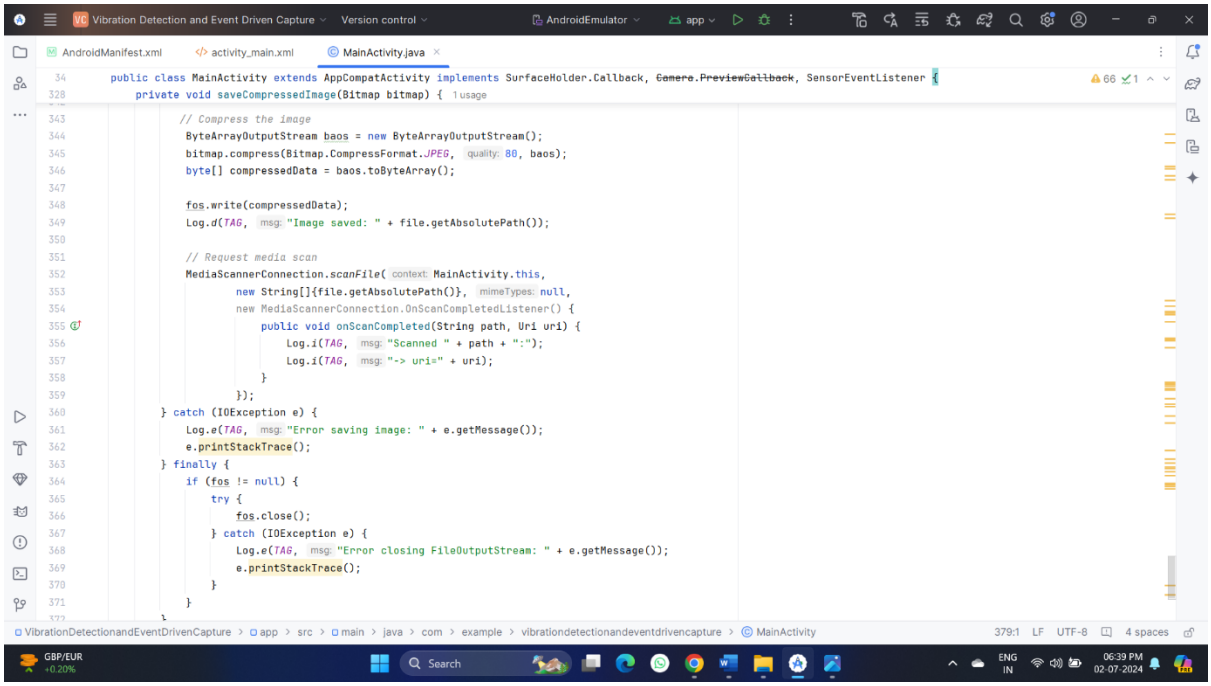




```
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {
    protected void onCreate(Bundle savedInstanceState) {
            accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
            isAccelerometerAvailable = accelerometer != null;
        }
        // Request permissions
        if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.CAMERA}, CAMERA_PERMISSION_CODE);
        } else {
            openCamera();
        }

        if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, STORAGE_PERMISSION_CODE);
        }
    }

    @Override
    protected void onResume() {
        super.onResume();
        // Register accelerometer sensor listener
        if (isAccelerometerAvailable) {

            sensorManager.registerListener( listener: this, accelerometer, SensorManager.SENSOR_DELAY_NORMAL);
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        // Unregister accelerometer sensor listener and release camera
```

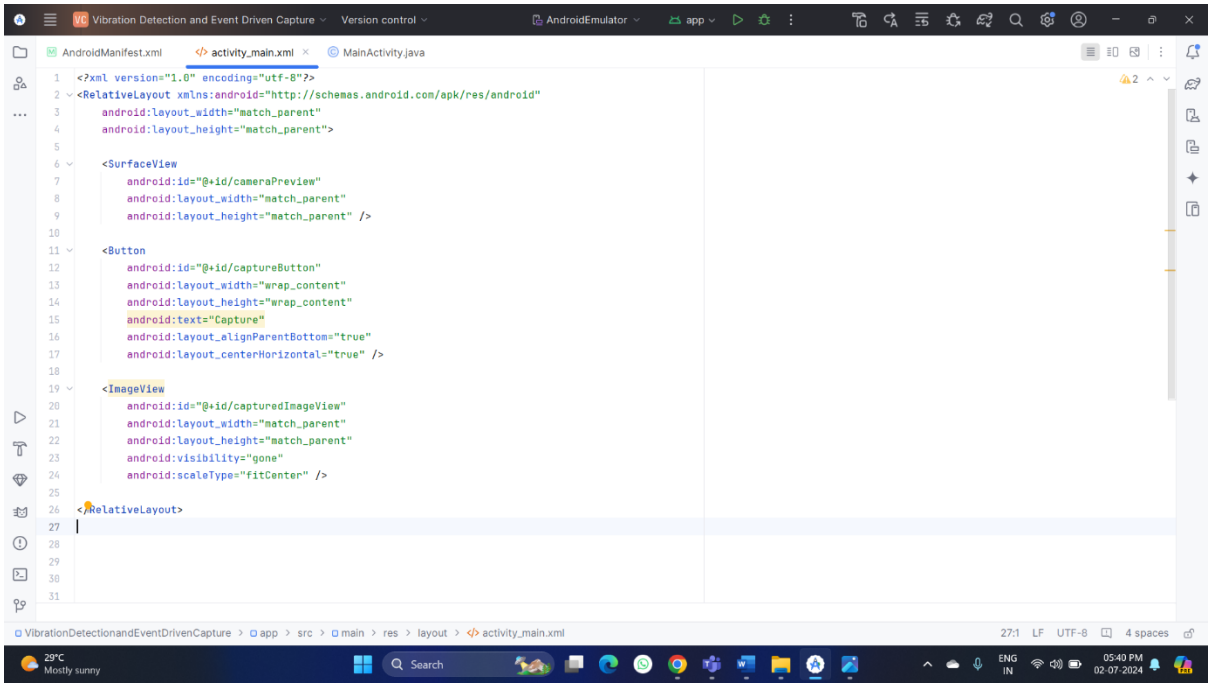


```
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {
    @Override
    public void surfaceDestroyed(@NonNull SurfaceHolder holder) {
        releaseCamera();
    }

    private void releaseCamera() {  2 usages
        if (camera != null) {
            try {
                camera.stopPreview();
                camera.setPreviewCallback(null);
                camera.release();
                camera = null;
                previewing = false;
            } catch (Exception e) {
                Log.e(TAG,  msg: "Error releasing camera: " + e.getMessage());
                e.printStackTrace();
            }
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
                                           @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (requestCode == CAMERA_PERMISSION_CODE) {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                openCamera();
            } else {
                Toast.makeText( context: this,  text: "Camera permission required", Toast.LENGTH_SHORT).show();
            }
        } else if (requestCode == STORAGE_PERMISSION_CODE) {
```



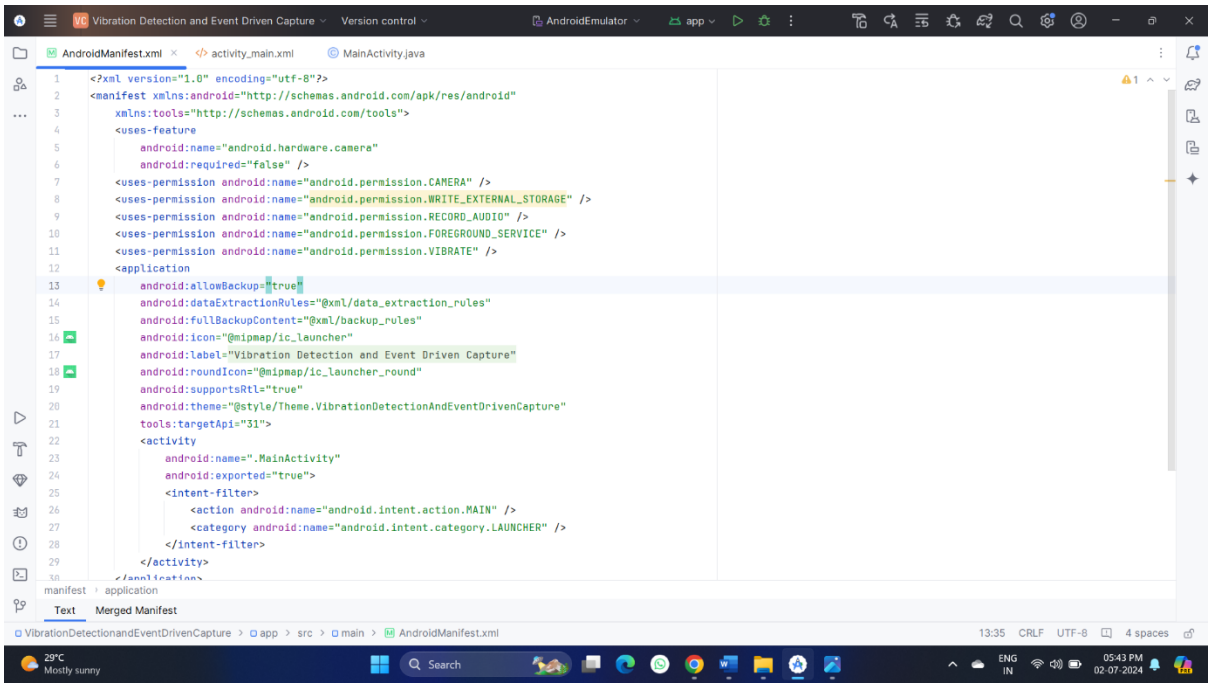
```
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {
    private void saveCompressedImage(Bitmap bitmap) {
        // Compress the image
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        bitmap.compress(Bitmap.CompressFormat.JPEG, quality: 80, baos);
        byte[] compressedData = baos.toByteArray();

        fos.write(compressedData);
        Log.d(TAG, msg: "Image saved: " + file.getAbsolutePath());

        // Request media scan
        MediaScannerConnection.scanFile( context: MainActivity.this,
                new String[]{file.getAbsolutePath()}, mimeTypes: null,
                new MediaScannerConnection.OnScanCompletedListener() {
                    public void onScanCompleted(String path, Uri uri) {
                        Log.i(TAG, msg: "Scanned " + path + ":");
                        Log.i(TAG, msg: "-> uri=" + uri);
                    }
                });
    } catch (IOException e) {
        Log.e(TAG, msg: "Error saving image: " + e.getMessage());
        e.printStackTrace();
    } finally {
        if (fos != null) {
            try {
                fos.close();
            } catch (IOException e) {
                Log.e(TAG, msg: "Error closing FileOutputStream: " + e.getMessage());
                e.printStackTrace();
            }
        }
    }
```

Update the activity_main.xml

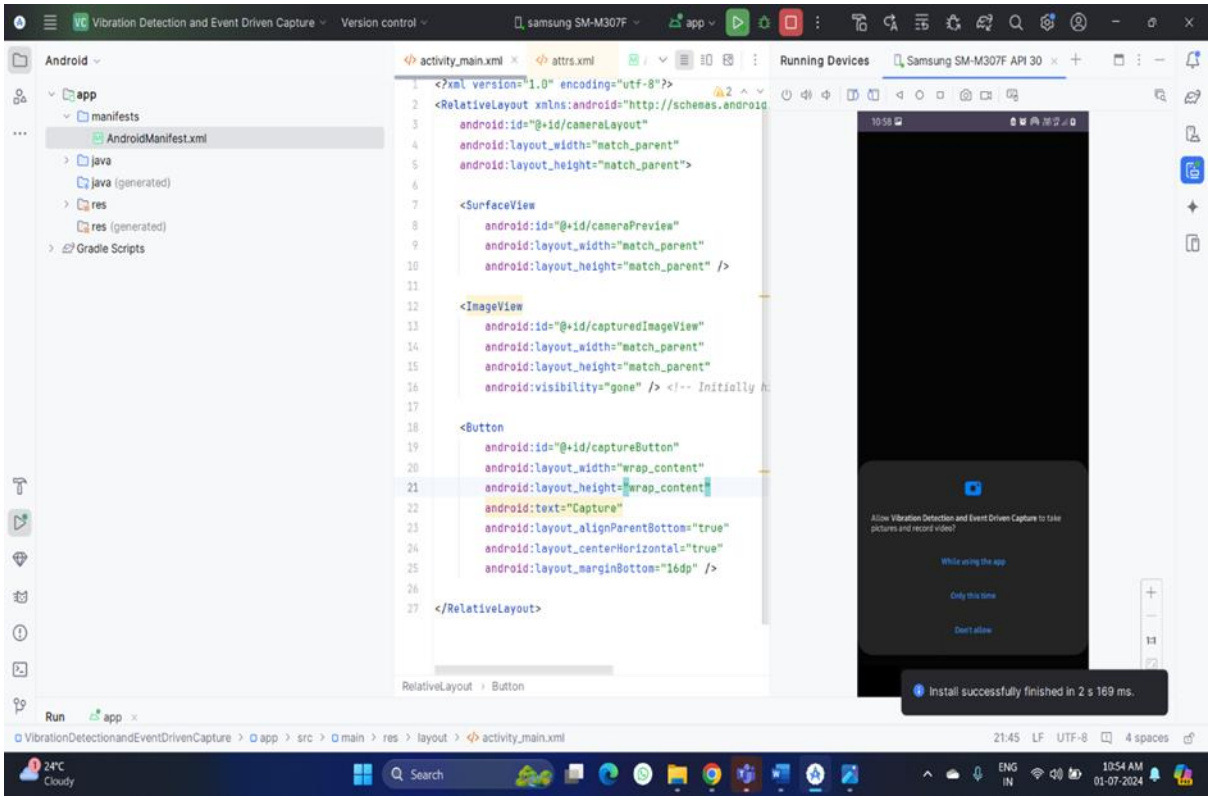


```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <SurfaceView
        android:id="@+id/cameraPreview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <Button
        android:id="@+id/captureButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Capture"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true" />

    <ImageView
        android:id="@+id/capturedImageView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="gone"
        android:scaleType="fitCenter" />

</RelativeLayout>
```

Update the AndroidManifest.xml



DAY 2 TASK 4 OUTPUT SCREENSHOTS



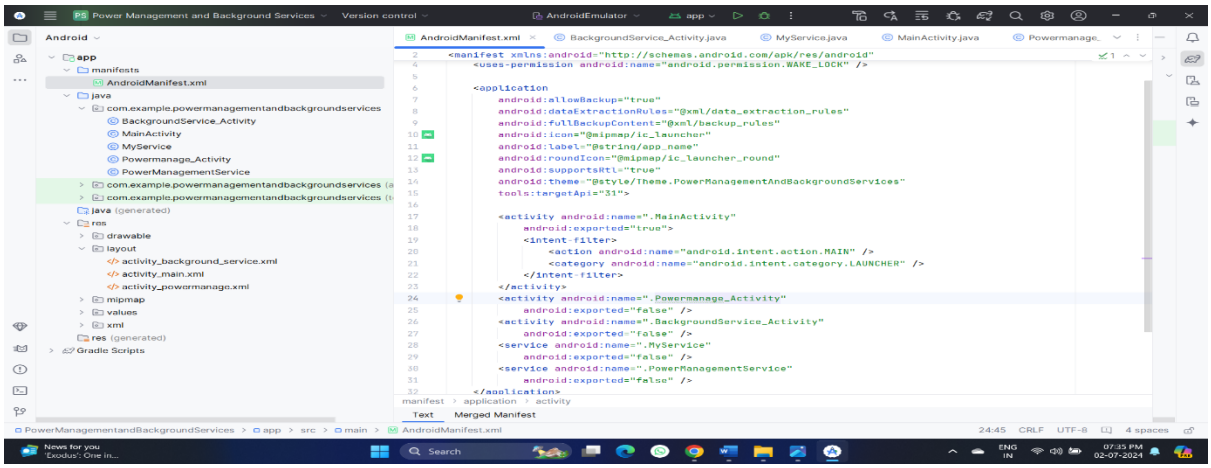
Day 3: Power Management and Background Services

It aims to enhance vehicle security and efficiency through an Android application. The application utilizes background services to manage power consumption and initiate surveillance when the vehicle is parked and the engine is off.

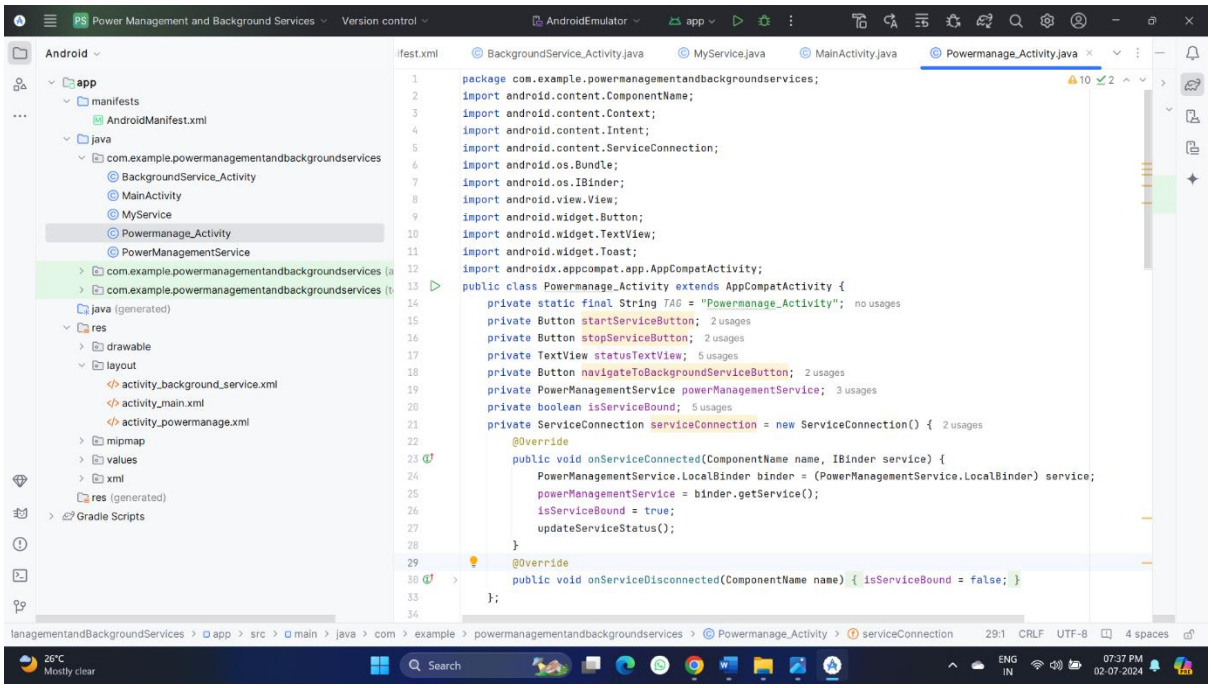
Task 1: Design a power management system that minimizes battery usage when the vehicle is parked.

The PowerManagementService was implemented to handle power management logic. This service monitors the vehicle's parked status and implements strategies to minimize battery usage, such as reducing CPU usage and optimizing network activities

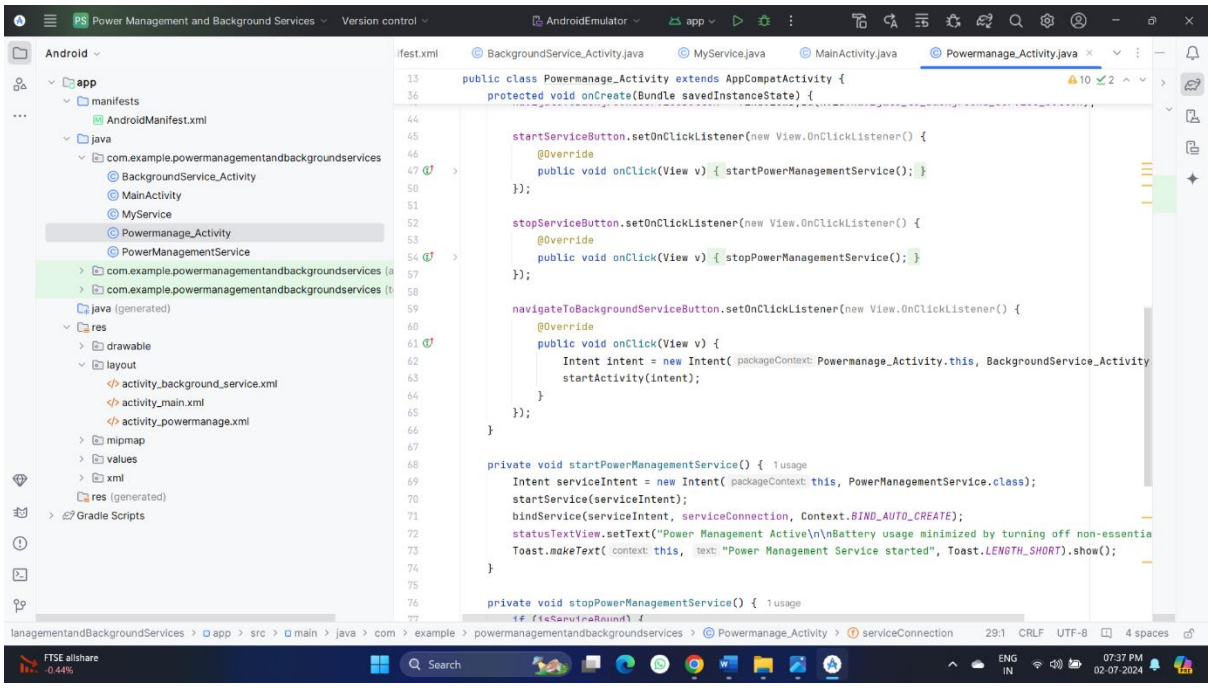
AndroidManifest.xml



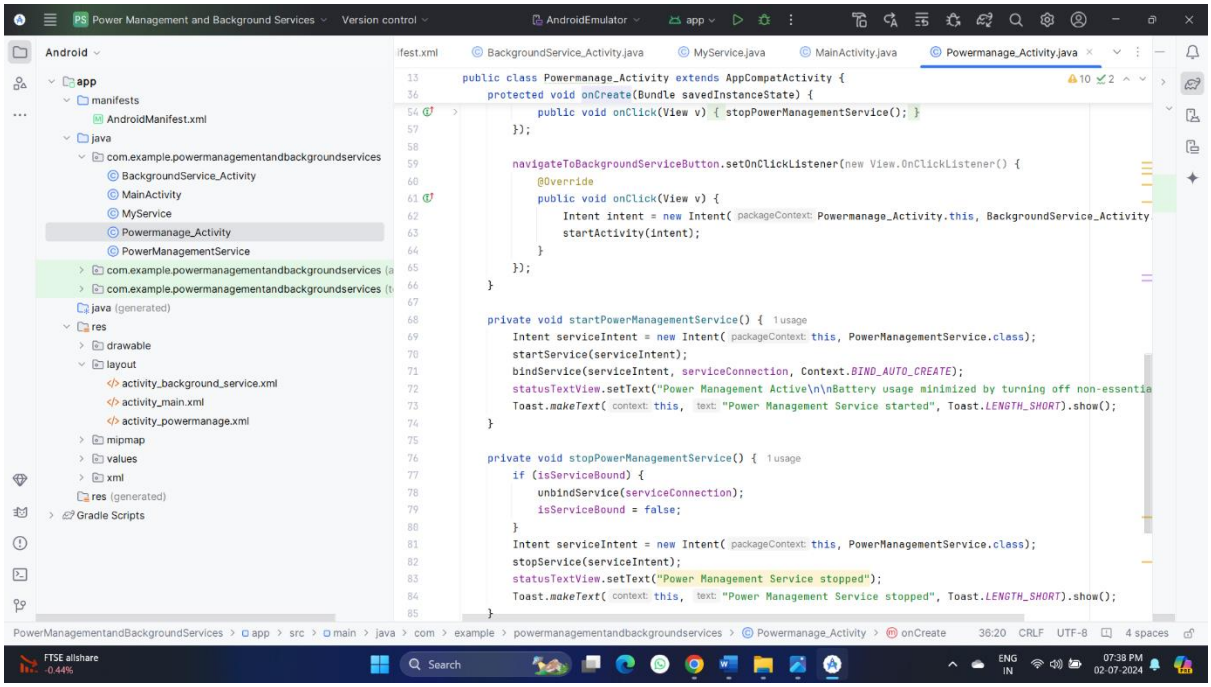
Powermanage_Activity.java



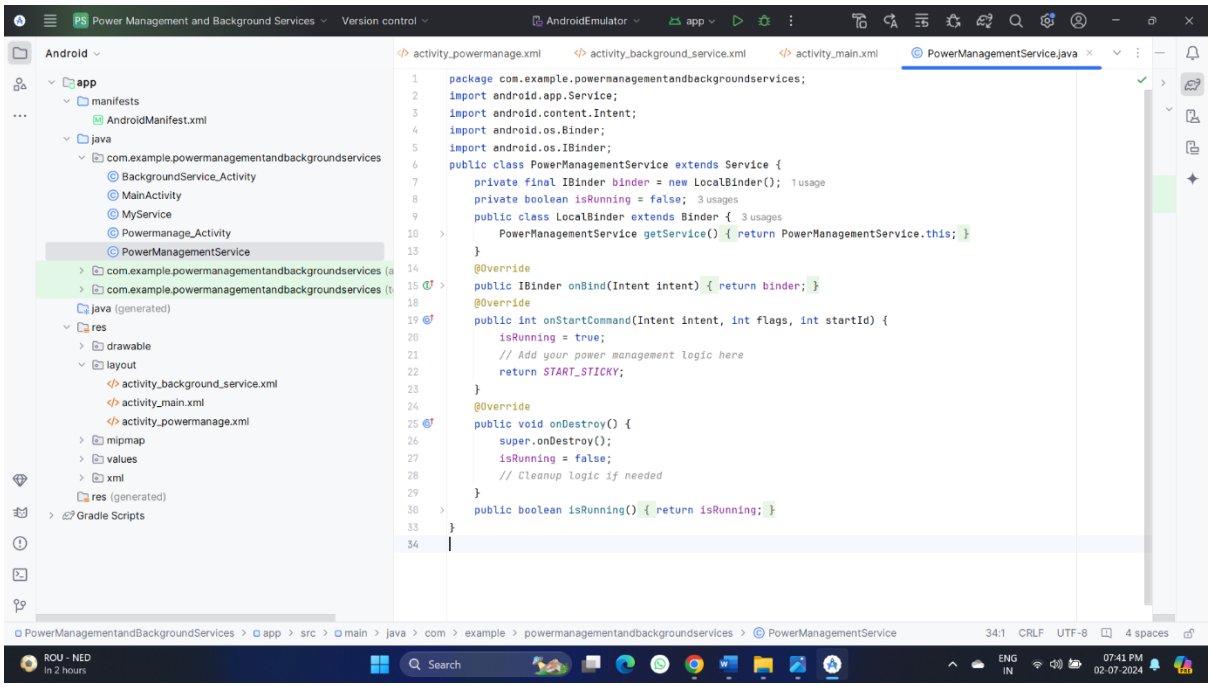
```
package com.example.powermanagementandbackgroundservices;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
public class Powermanage_Activity extends AppCompatActivity {
    private static final String TAG = "Powermanage_Activity";  no usages
    private Button startServiceButton;  2 usages
    private Button stopServiceButton;  2 usages
    private TextView statusTextView;  5 usages
    private Button navigateToBackgroundServiceButton;  2 usages
    private PowerManagementService powerManagementService;  3 usages
    private boolean isServiceBound;  5 usages
    private ServiceConnection serviceConnection = new ServiceConnection() {  2 usages
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            PowerManagementService.LocalBinder binder = (PowerManagementService.LocalBinder) service;
            powerManagementService = binder.getService();
            isServiceBound = true;
            updateServiceStatus();
        }
        @Override
        public void onServiceDisconnected(ComponentName name) { isServiceBound = false; }
    };
```



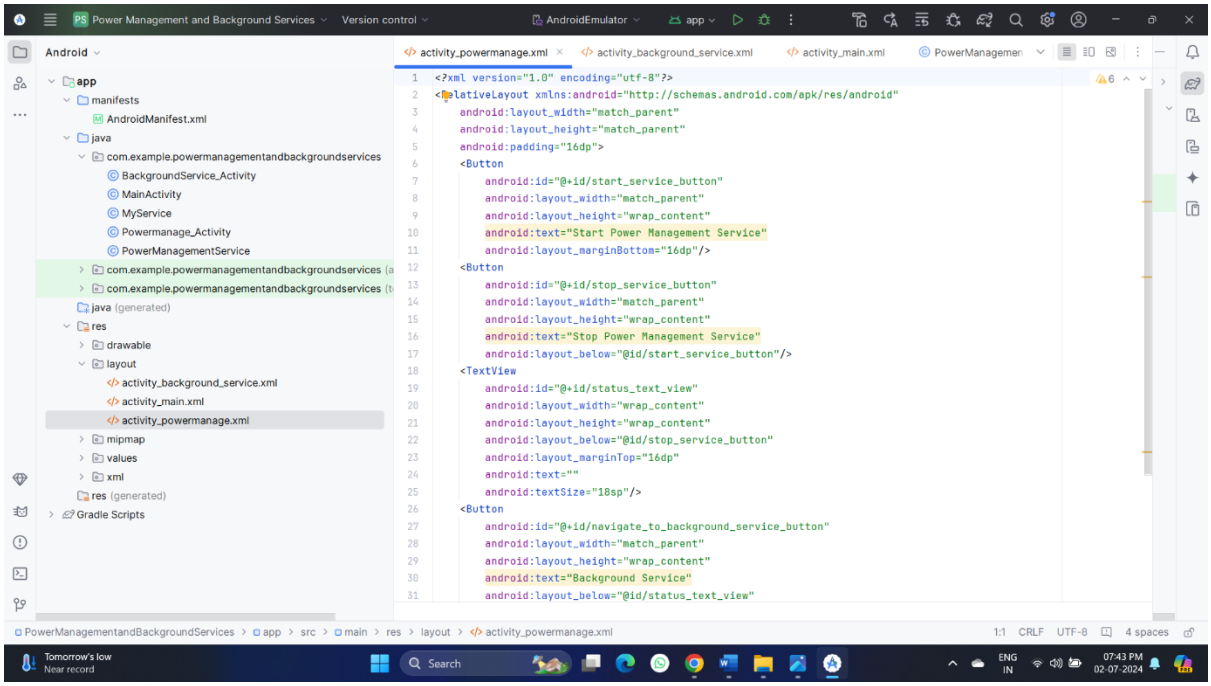
```
public class Powermanage_Activity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {

        startServiceButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { startPowerManagementService(); }
        });

        stopServiceButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { stopPowerManagementService(); }
        });

        navigateToBackgroundServiceButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent( packageContext: Powermanage_Activity.this, BackgroundService_Activity
                startActivity(intent);
            }
        });
    }

    private void startPowerManagementService() {  1 usage
        Intent serviceIntent = new Intent( packageContext: this, PowerManagementService.class);
        startService(serviceIntent);
        bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
        statusTextView.setText("Power Management Active\n\nBattery usage minimized by turning off non-essentia
        Toast.makeText( context: this,  text: "Power Management Service started", Toast.LENGTH_SHORT).show();
    }

    private void stopPowerManagementService() {  1 usage
```



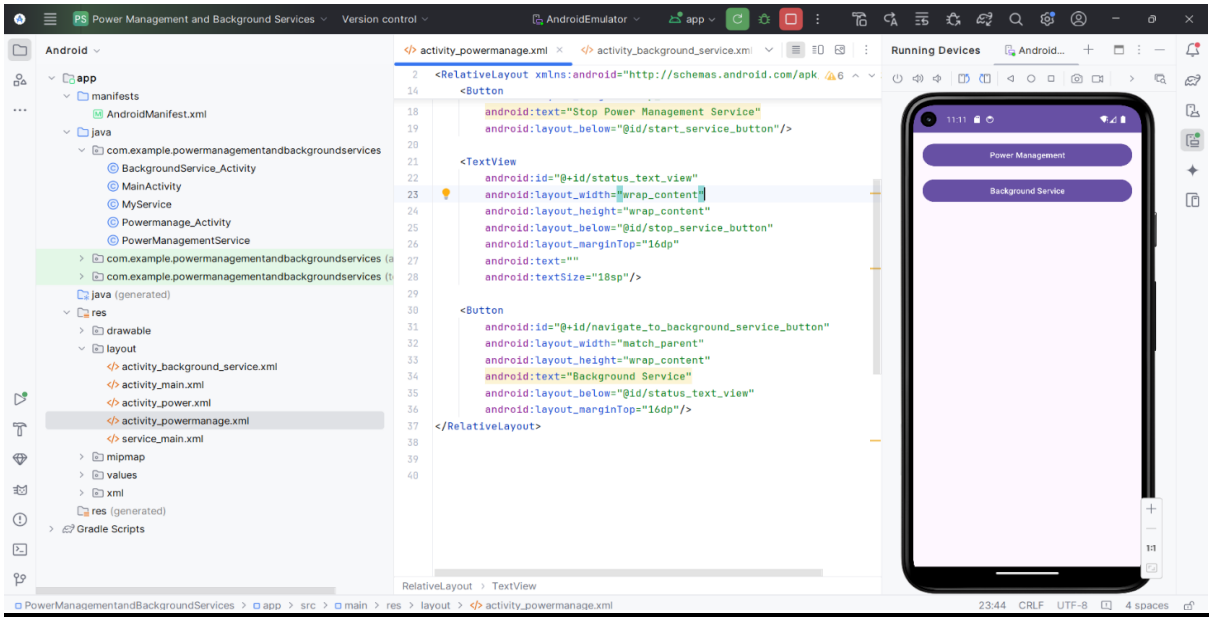
PowerManagementService.java

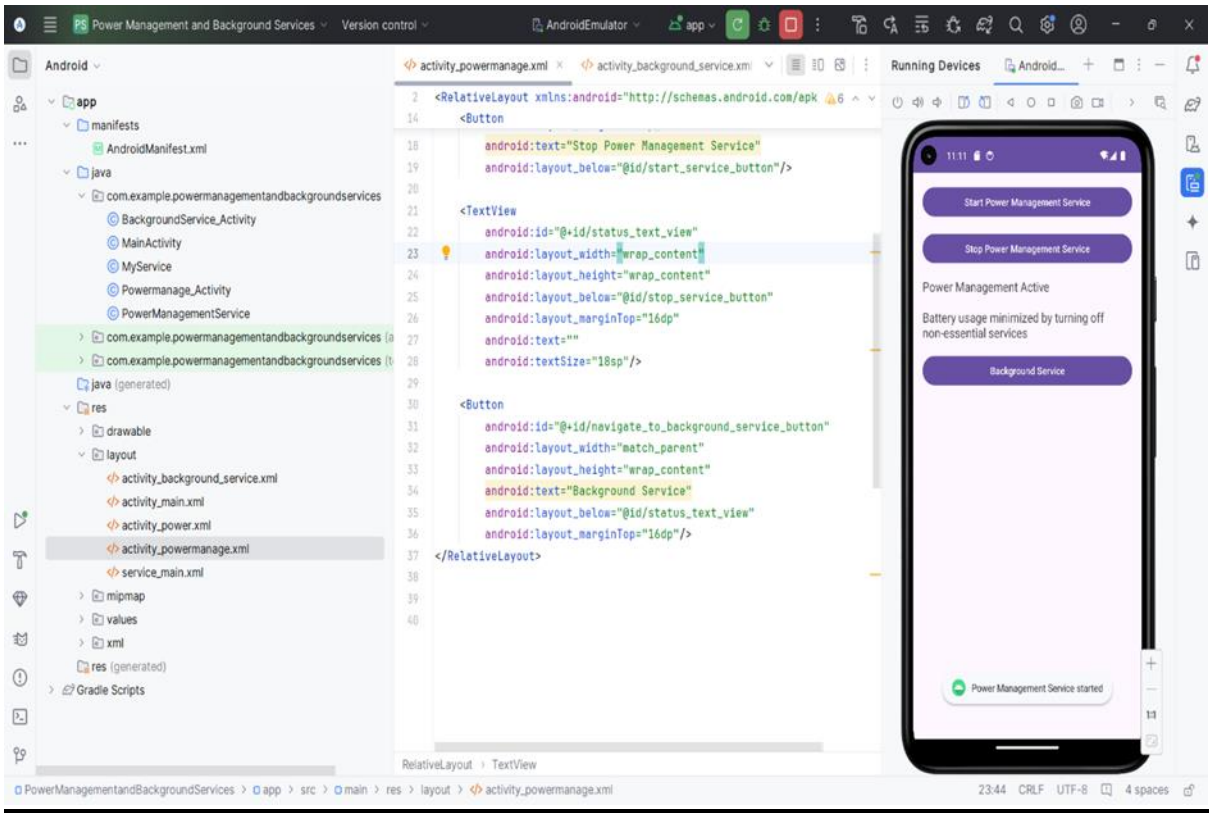


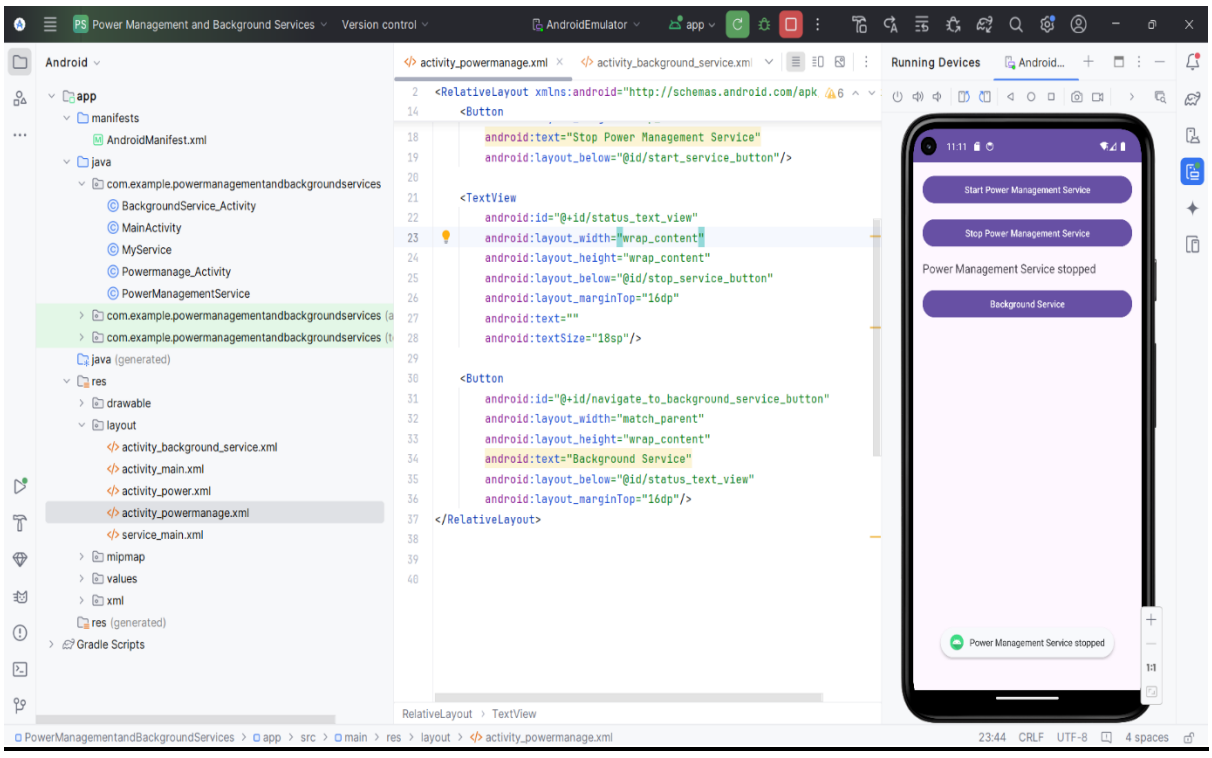
activity_powermanage.xml



DAY 3 TASK 1 OUTPUT SCREENSHOTS



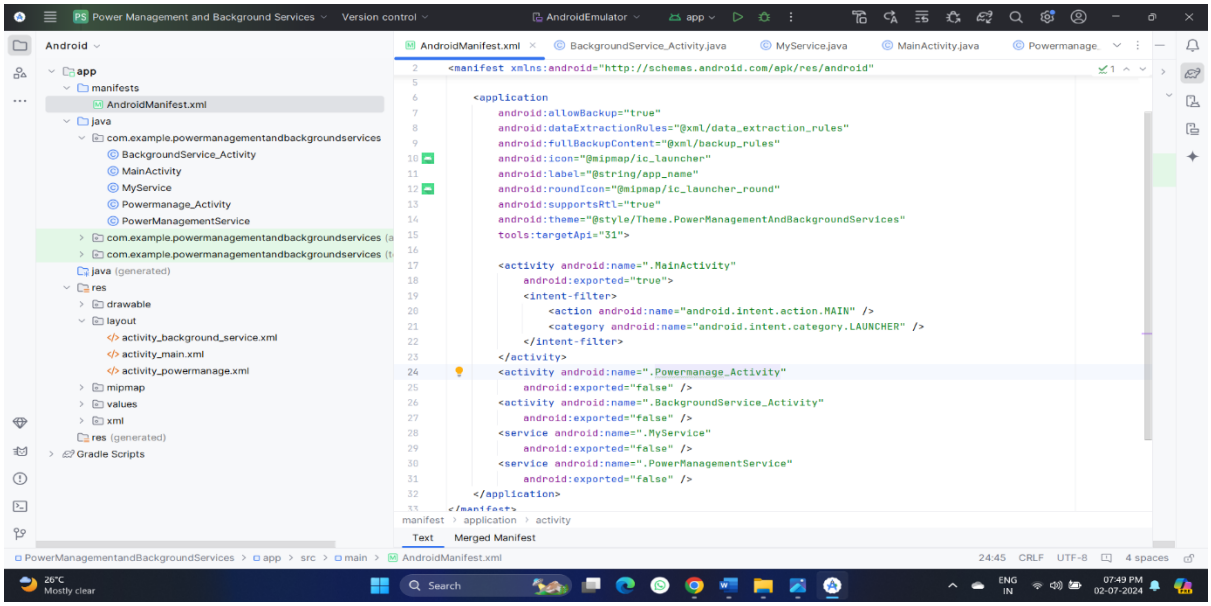

Power Management and Background Services
Version control
AndroidEmulator
app
Android
activity_powermanage.xml
activity_background_service.xml
Running Devices
Android...
app
manifests
AndroidManifest.xml
java
com.example.powermanagementandbackgroundservices
BackgroundService_Activity
MainActivity
MyService
Powermanage_Activity
PowerManagementService
java (generated)
res
drawable
layout
activity_background_service.xml
activity_main.xml
activity_power.xml
activity_powermanage.xml
service_main.xml
mipmap
values
xml
res (generated)
Gradle Scripts
<RelativeLayout xmlns:android="http://schemas.android.com/apk
<Button
android:text="Stop Power Management Service"
android:layout_below="@id/start_service_button"/>
<TextView
android:id="@+id/status_text_view"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/stop_service_button"
android:layout_marginTop="16dp"
android:text=""
android:textSize="18sp"/>
<Button
android:id="@+id/navigate_to_background_service_button"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Background Service"
android:layout_below="@id/status_text_view"
android:layout_marginTop="16dp"/>
</RelativeLayout>
RelativeLayout
TextView
Start Power Management Service
Stop Power Management Service
Power Management Active
Battery usage minimized by turning off non-essential services
Background Service
Power Management Service started
PowerManagementandBackgroundServices
app
src
main
res
layout
activity_powermanage.xml
23:44
CRLF
UTF-8
4 spaces


Power Management and Background Services
Version control
AndroidEmulator
app
Android
activity_powermanage.xml
activity_background_service.xml
Running Devices
Android...
app
manifests
AndroidManifest.xml
java
com.example.powermanagementandbackgroundservices
BackgroundService_Activity
MainActivity
MyService
Powermanage_Activity
PowerManagementService
java (generated)
res
drawable
layout
activity_background_service.xml
activity_main.xml
activity_power.xml
activity_powermanage.xml
service_main.xml
mipmap
values
xml
res (generated)
Gradle Scripts
<RelativeLayout xmlns:android="http://schemas.android.com/apk
<Button
android:text="Stop Power Management Service"
android:layout_below="@id/start_service_button"/>
<TextView
android:id="@+id/status_text_view"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/stop_service_button"
android:layout_marginTop="16dp"
android:text=""
android:textSize="18sp"/>
<Button
android:id="@+id/navigate_to_background_service_button"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Background Service"
android:layout_below="@id/status_text_view"
android:layout_marginTop="16dp"/>
</RelativeLayout>
RelativeLayout
TextView
Start Power Management Service
Stop Power Management Service
Power Management Service stopped
Background Service
Power Management Service stopped
PowerManagementandBackgroundServices
app
src
main
res
layout
activity_powermanage.xml
23:44
CRLF
UTF-8
4 spaces

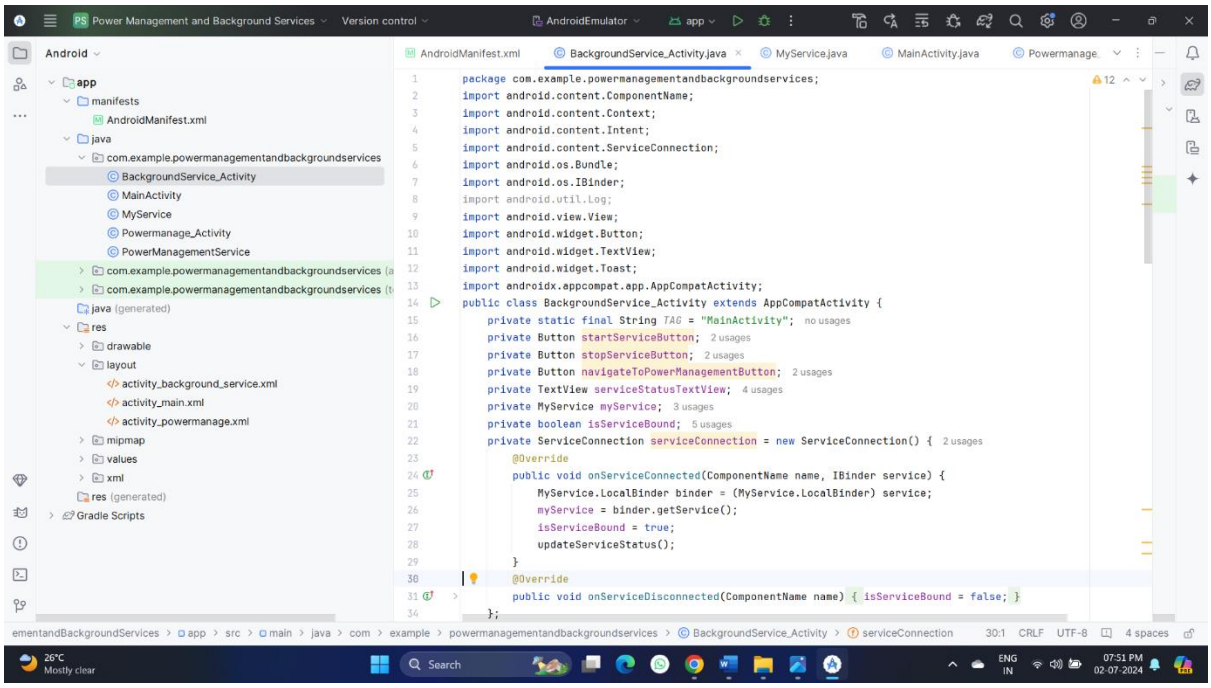
Task 2: Implement a background service that runs the surveillance system only when the vehicle is locked and the engine is off.

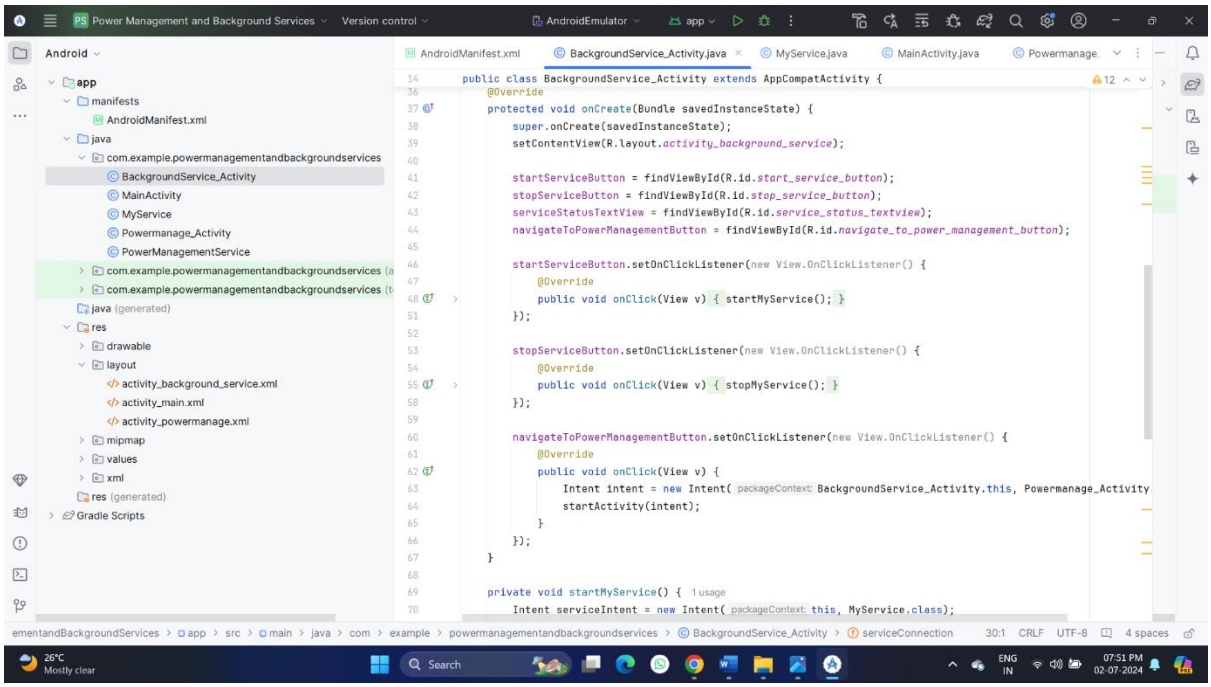
The BackgroundService runs surveillance activities when triggered by specific conditions, such as the vehicle being locked and the engine being off. It initiates camera operations or sensor data collection to monitor surroundings for potential threats**.**

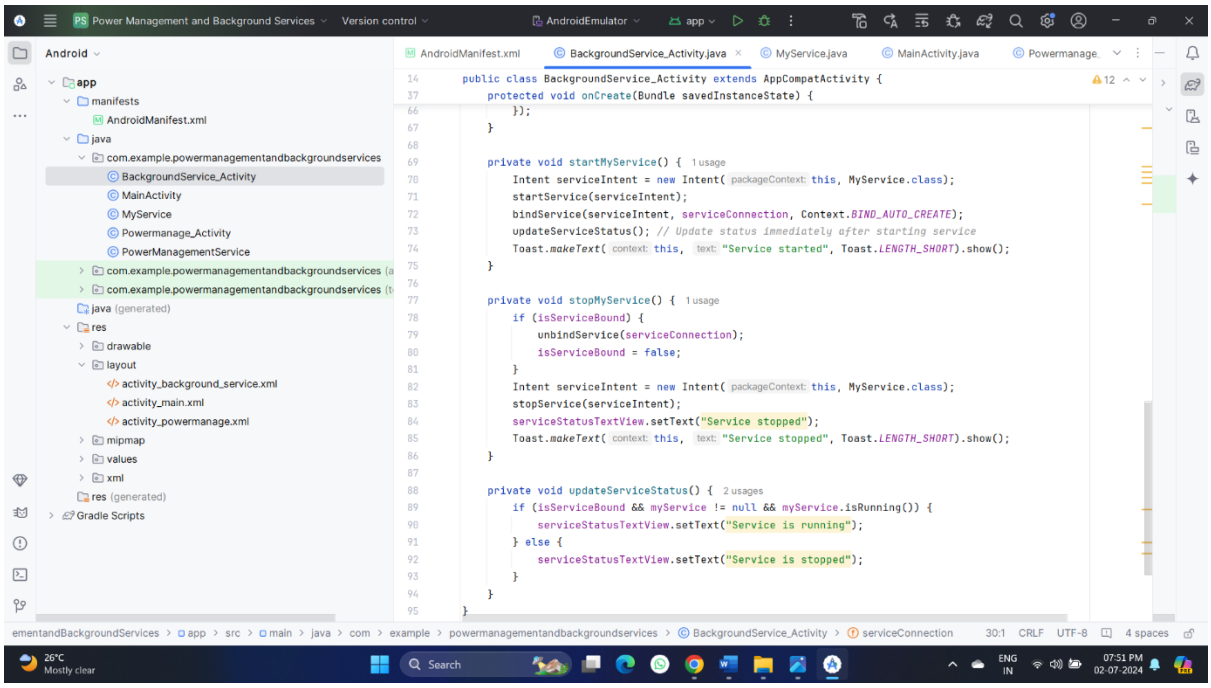
AndroidManifest.xml



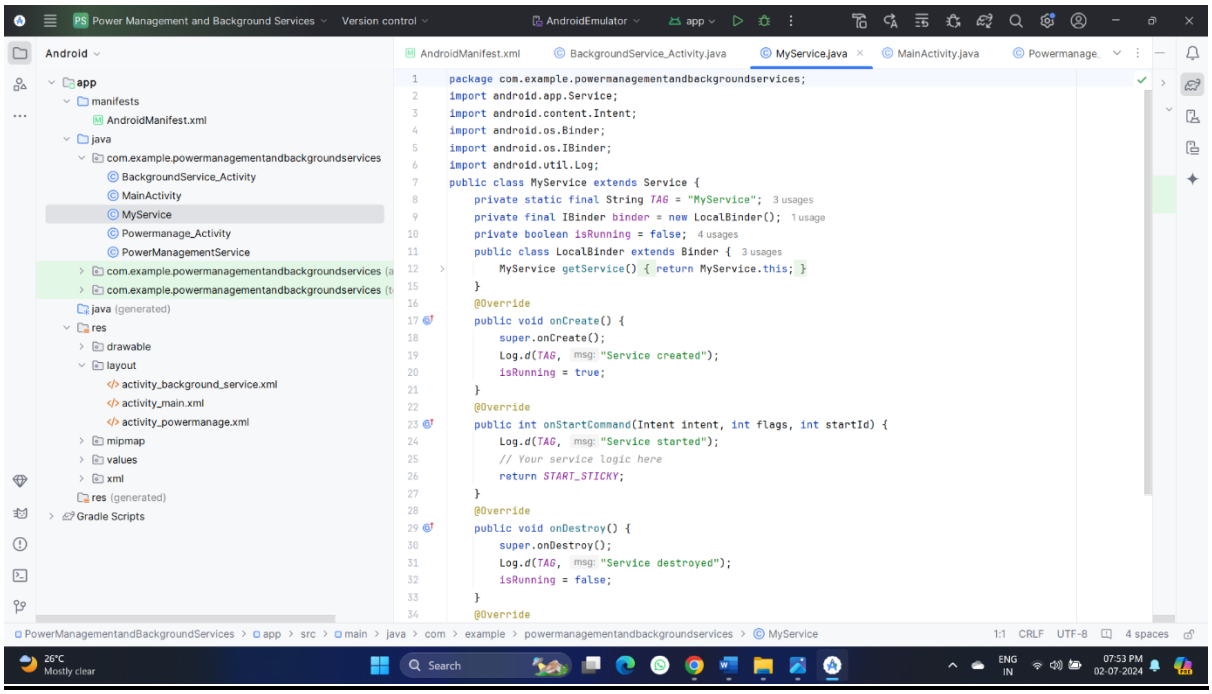
BackgroundService_Activity.java



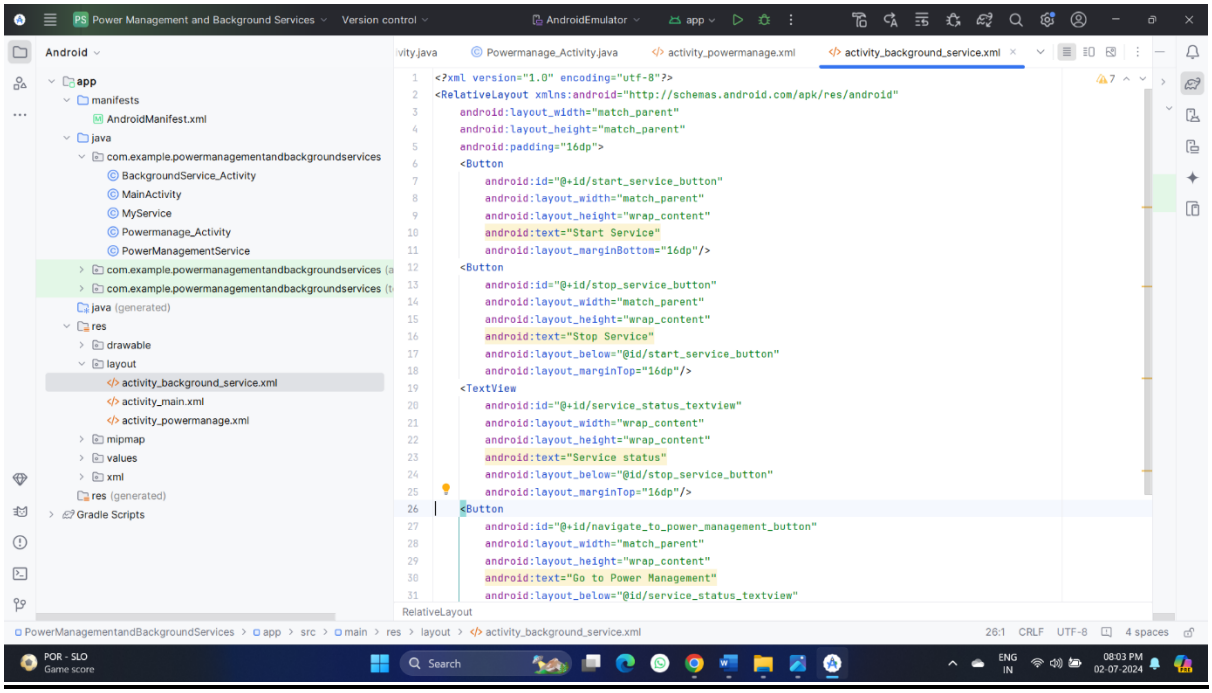

Power Management and Background Services
Version control
AndroidEmulator
app
Android
AndroidManifest.xml
BackgroundService_Activity.java
MyService.java
MainActivity.java
Powermanage
app
manifests
AndroidManifest.xml
java
com.example.powermanagementandbackgroundservices
BackgroundService_Activity
MainActivity
MyService
Powermanage_Activity
PowerManagementService
java (generated)
res
drawable
layout
activity_background_service.xml
activity_main.xml
activity_powermanage.xml
mipmap
values
xml
res (generated)
Gradle Scripts
public class BackgroundService_Activity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_background_service);
startServiceButton = findViewById(R.id.start_service_button);
stopServiceButton = findViewById(R.id.stop_service_button);
serviceStatusTextView = findViewById(R.id.service_status_textview);
navigateToPowerManagementButton = findViewById(R.id.navigate_to_power_management_button);
startServiceButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) { startMyService(); }
});
stopServiceButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) { stopMyService(); }
});
navigateToPowerManagementButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(packageContext: BackgroundService_Activity.this, Powermanage_Activity
startActivity(intent);
}
});
}
private void startMyService() { 1 usage
Intent serviceIntent = new Intent(packageContext: this, MyService.class);
ementandBackgroundServices › app › src › main › java › com › example › powermanagementandbackgroundservices › BackgroundService_Activity › serviceConnection
30:1
CRLF
UTF-8
4 spaces
26°C
Mostly clear
Search
ENG
IN
07:51 PM
02-07-2024


Power Management and Background Services
Version control
AndroidEmulator
app
Android
AndroidManifest.xml
BackgroundService_Activity.java
MyService.java
MainActivity.java
Powermanage
app
manifests
AndroidManifest.xml
java
com.example.powermanagementandbackgroundservices
BackgroundService_Activity
MainActivity
MyService
Powermanage_Activity
PowerManagementService
java (generated)
res
drawable
layout
activity_background_service.xml
activity_main.xml
activity_powermanage.xml
mipmap
values
xml
res (generated)
Gradle Scripts
public class BackgroundService_Activity extends AppCompatActivity {
protected void onCreate(Bundle savedInstanceState) {
});
}
private void startMyService() { 1 usage
Intent serviceIntent = new Intent(packageContext: this, MyService.class);
startService(serviceIntent);
bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
updateServiceStatus(); // Update status immediately after starting service
Toast.makeText(context: this, text: "Service started", Toast.LENGTH_SHORT).show();
}
private void stopMyService() { 1 usage
if (isServiceBound) {
unbindService(serviceConnection);
isServiceBound = false;
}
Intent serviceIntent = new Intent(packageContext: this, MyService.class);
stopService(serviceIntent);
serviceStatusTextView.setText("Service stopped");
Toast.makeText(context: this, text: "Service stopped", Toast.LENGTH_SHORT).show();
}
private void updateServiceStatus() { 2 usages
if (isServiceBound && myService != null && myService.isRunning()) {
serviceStatusTextView.setText("Service is running");
} else {
serviceStatusTextView.setText("Service is stopped");
}
}
}
ementandBackgroundServices › app › src › main › java › com › example › powermanagementandbackgroundservices › BackgroundService_Activity › serviceConnection
30:1
CRLF
UTF-8
4 spaces
26°C
Mostly clear
Search
ENG
IN
07:51 PM
02-07-2024

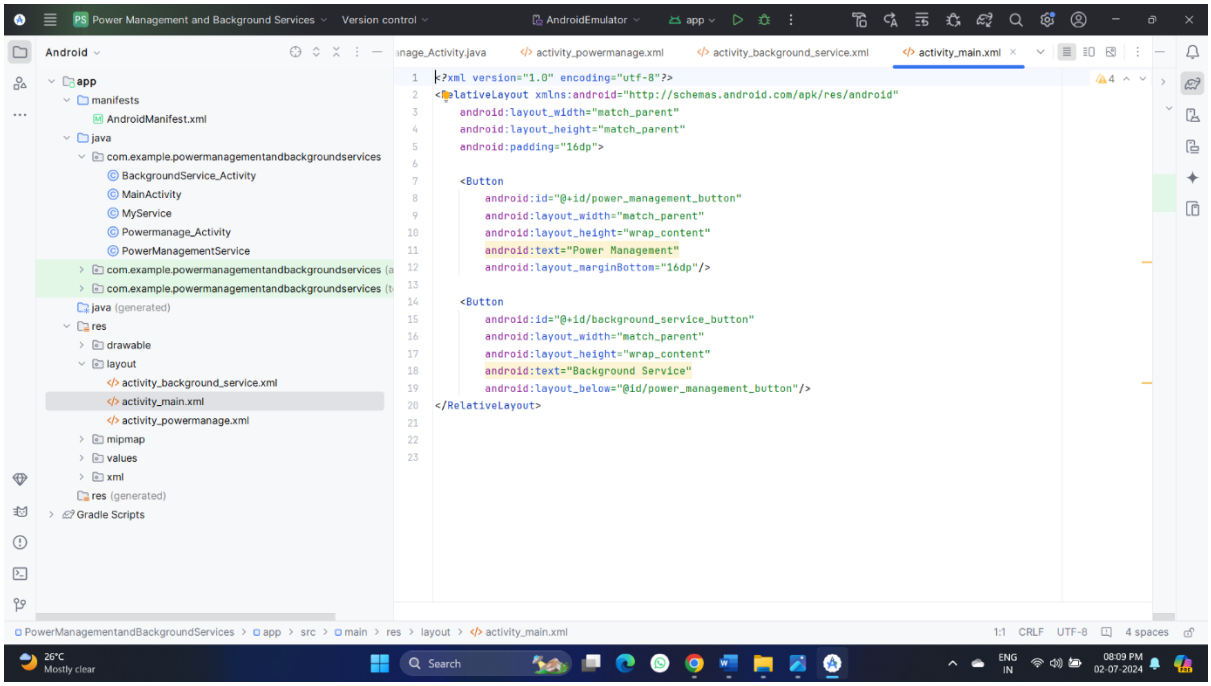
MyService.java.



activity_background_service.xml

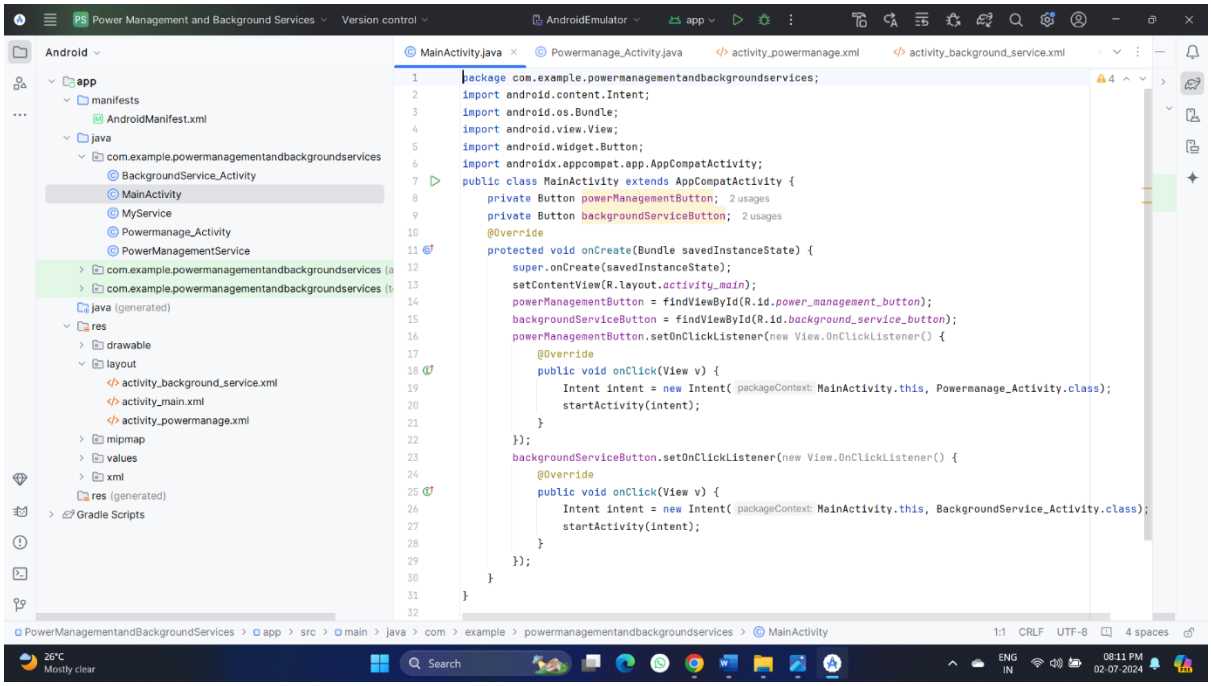


activity_main.xml



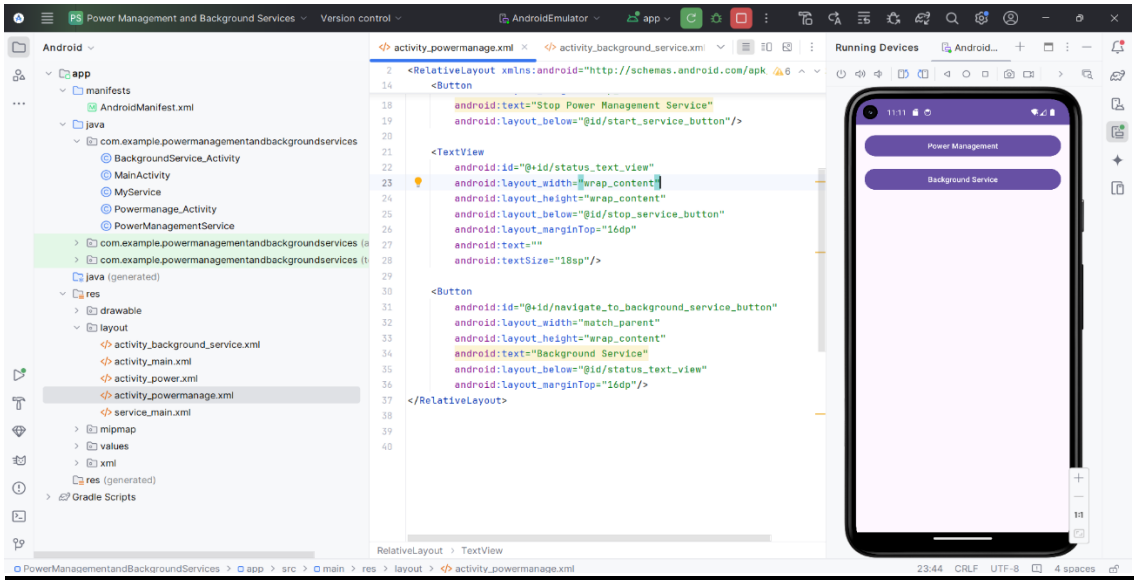
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <Button
        android:id="@+id/power_management_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Power Management"
        android:layout_marginBottom="16dp"/>

    <Button
        android:id="@+id/background_service_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Background Service"
        android:layout_below="@id/power_management_button"/>
</RelativeLayout>
```

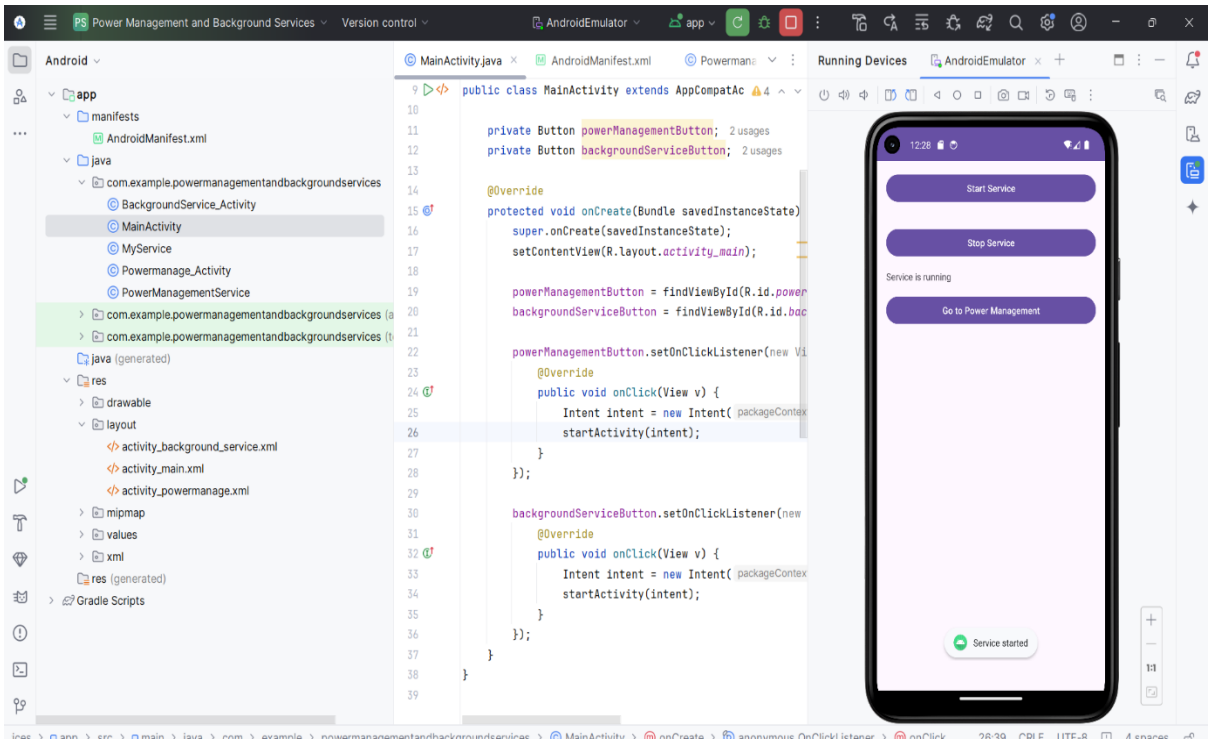
MainActivity.java

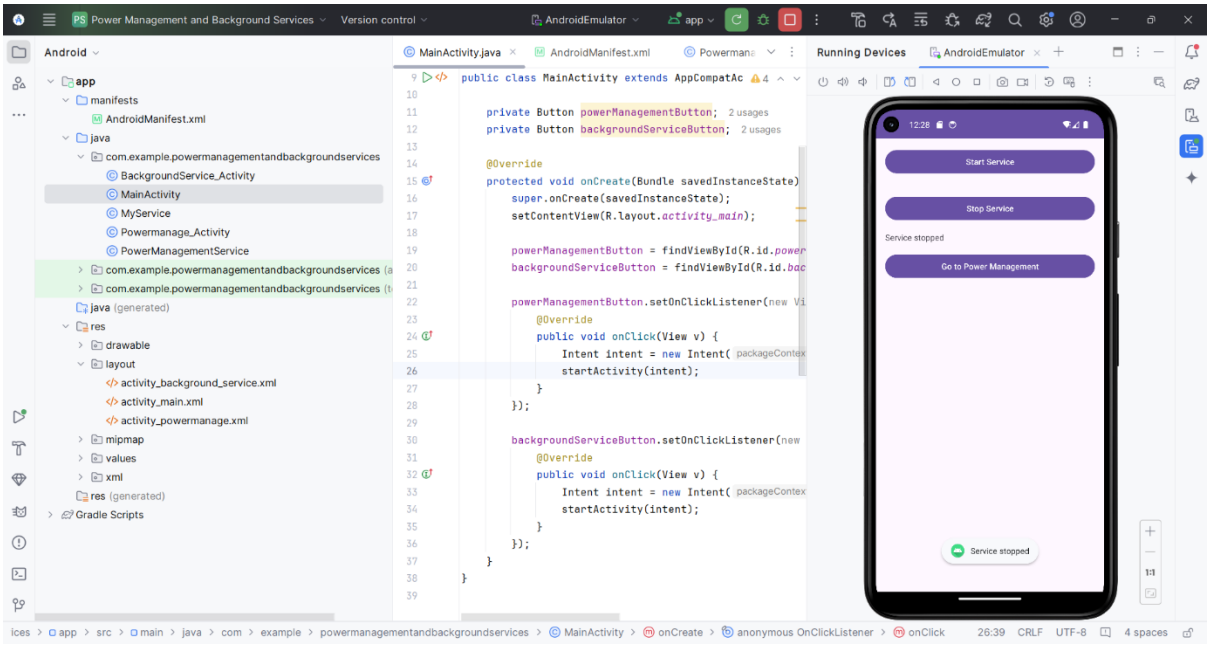


```
package com.example.powermanagementandbackgroundservices;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
    private Button powerManagementButton;  2 usages
    private Button backgroundServiceButton;  2 usages
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        powerManagementButton = findViewById(R.id.power_management_button);
        backgroundServiceButton = findViewById(R.id.background_service_button);
        powerManagementButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent( packageContext: MainActivity.this, Powermanage_Activity.class);
                startActivity(intent);
            }
        });
        backgroundServiceButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent( packageContext: MainActivity.this, BackgroundService_Activity.class);
                startActivity(intent);
            }
        });
    }
}
```

DAY 3 TASK 2 OUTPUT SCREENSHOTS





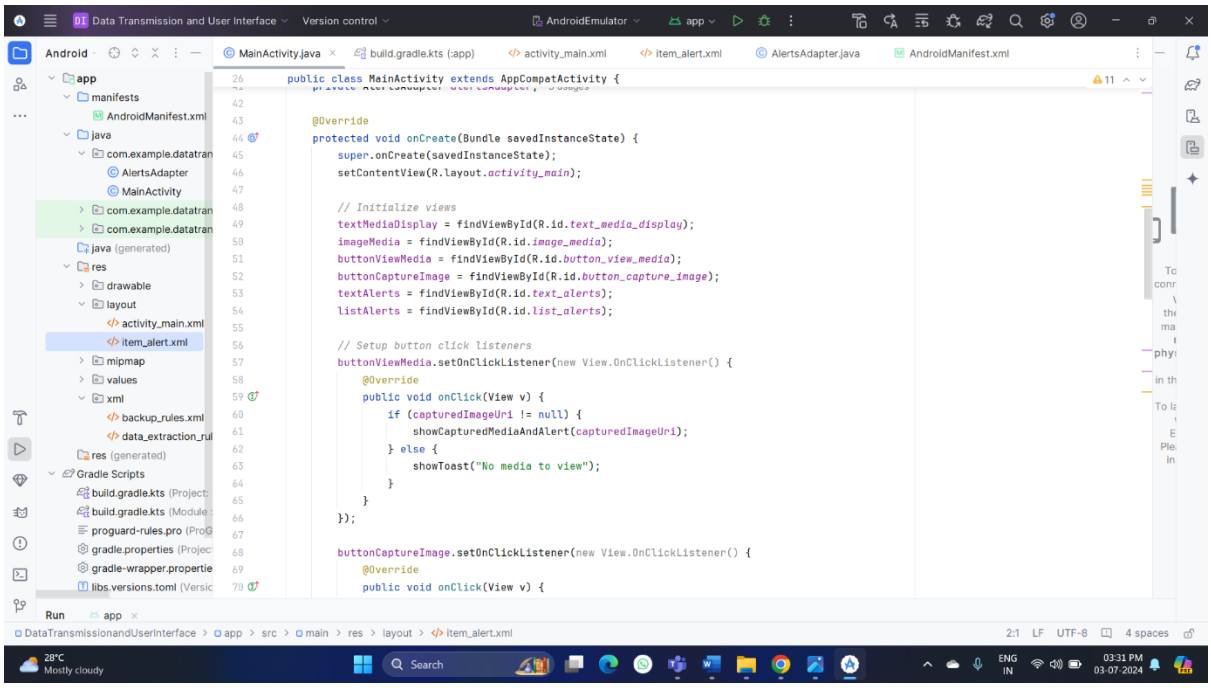


Day 4: Data Transmission and User Interface

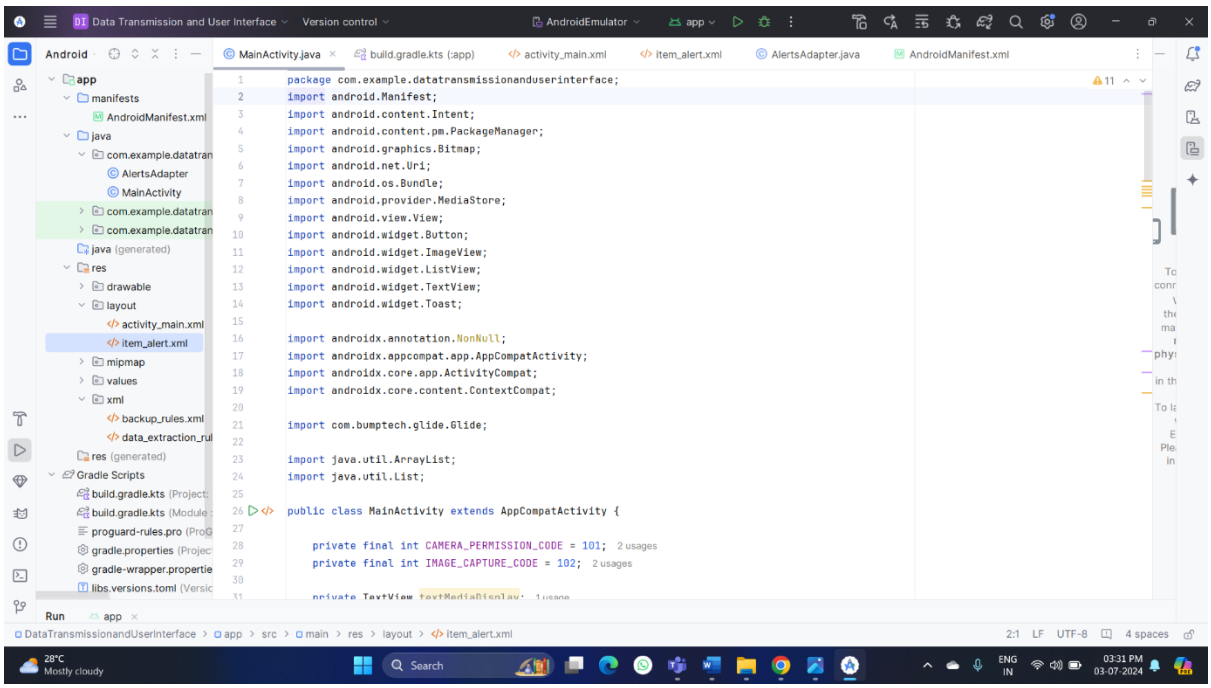
Task 1: Create a method for transmitting alerts and images to the vehicle owner's smartphone or designated device.

The goal is to create a method for transmitting alerts and images to the vehicle owner's smartphone or designated device. This involves setting up a service that captures the necessary data (alerts and images) and sends it over the network to the intended recipient. It involves setting up a robust mechanism for transmitting alerts and images from the surveillance system to the vehicle owner. By leveraging Android services and utility methods for network communication, we can ensure that alerts are sent efficiently and reliably, providing the vehicle owner with timely information about potential security threats.

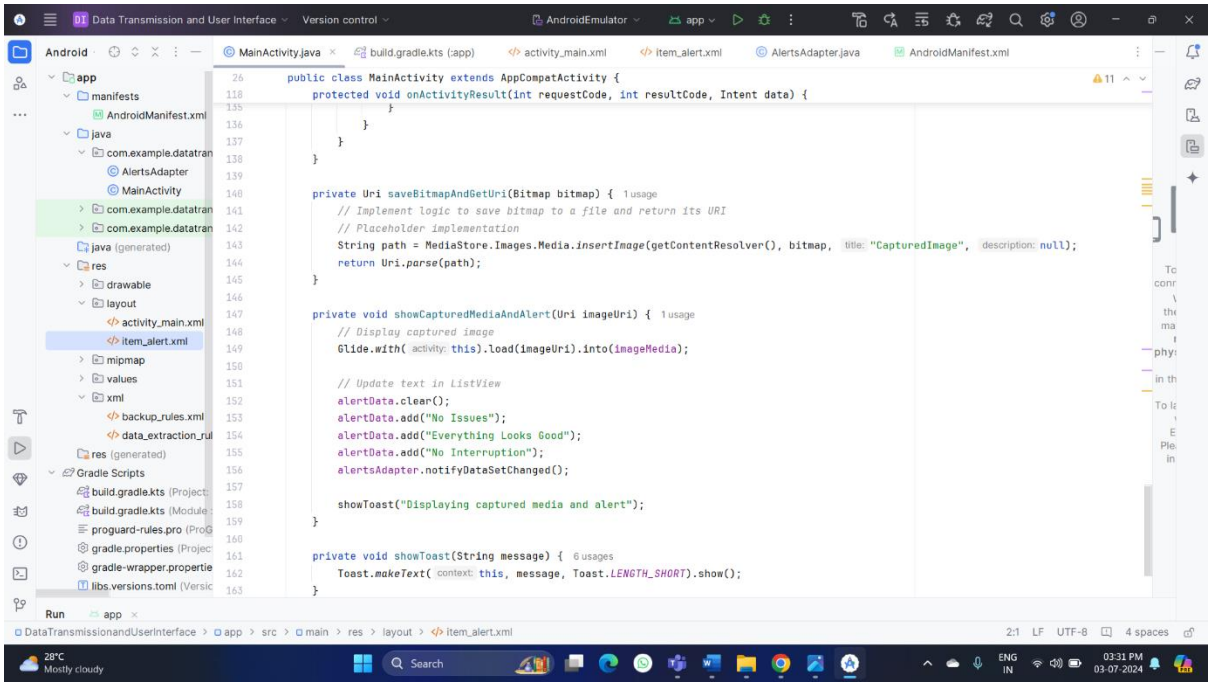
MainActivity.java



```
public class MainActivity extends AppCompatActivity {

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

        // Initialize views
        textMediaDisplay = findViewById(R.id.text_media_display);
        imageMedia = findViewById(R.id.image_media);
        buttonViewMedia = findViewById(R.id.button_view_media);
        buttonCaptureImage = findViewById(R.id.button_capture_image);
        textAlerts = findViewById(R.id.text_alerts);
        listAlerts = findViewById(R.id.list_alerts);

        // Setup button click listeners
        buttonViewMedia.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (capturedImageUri != null) {
                    showCapturedMediaAndAlert(capturedImageUri);
                } else {
                    showToast("No media to view");
                }
            }
        });

        buttonCaptureImage.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

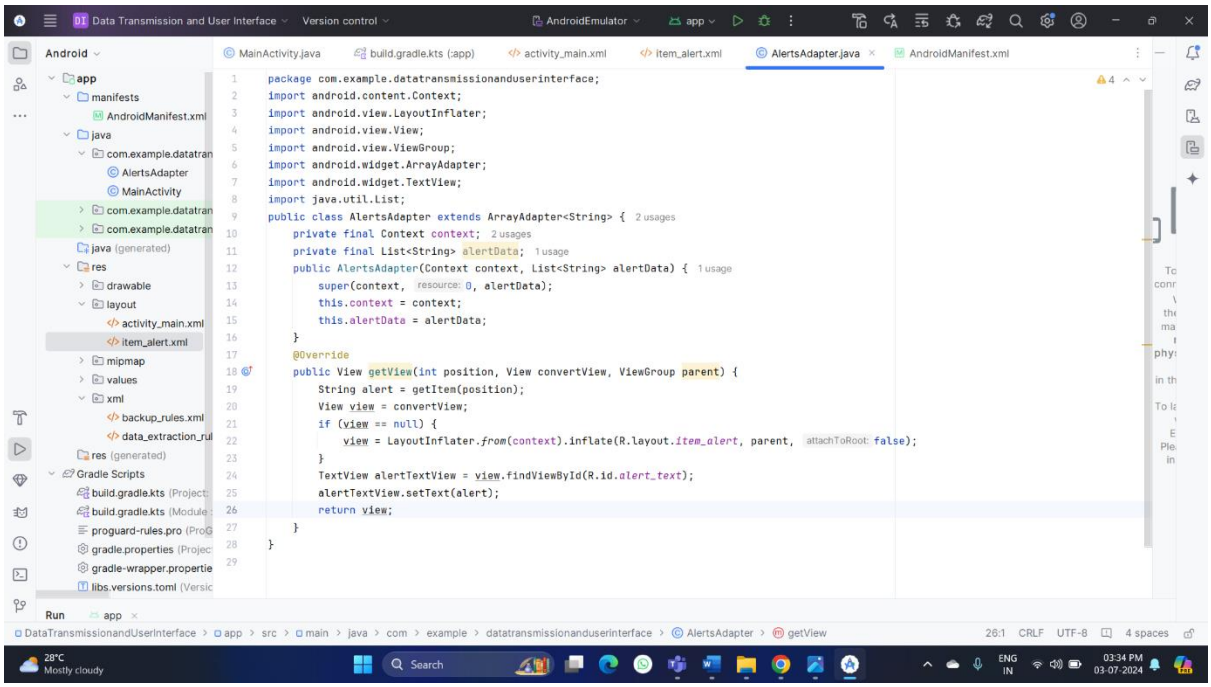


```
package com.example.datatransmissionanduserinterface;
import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

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

import com.bumptech.glide.Glide;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    private final int CAMERA_PERMISSION_CODE = 101;  2 usages
    private final int IMAGE_CAPTURE_CODE = 102;  2 usages

    private TextView textMediaDisplay;  1 usage
```



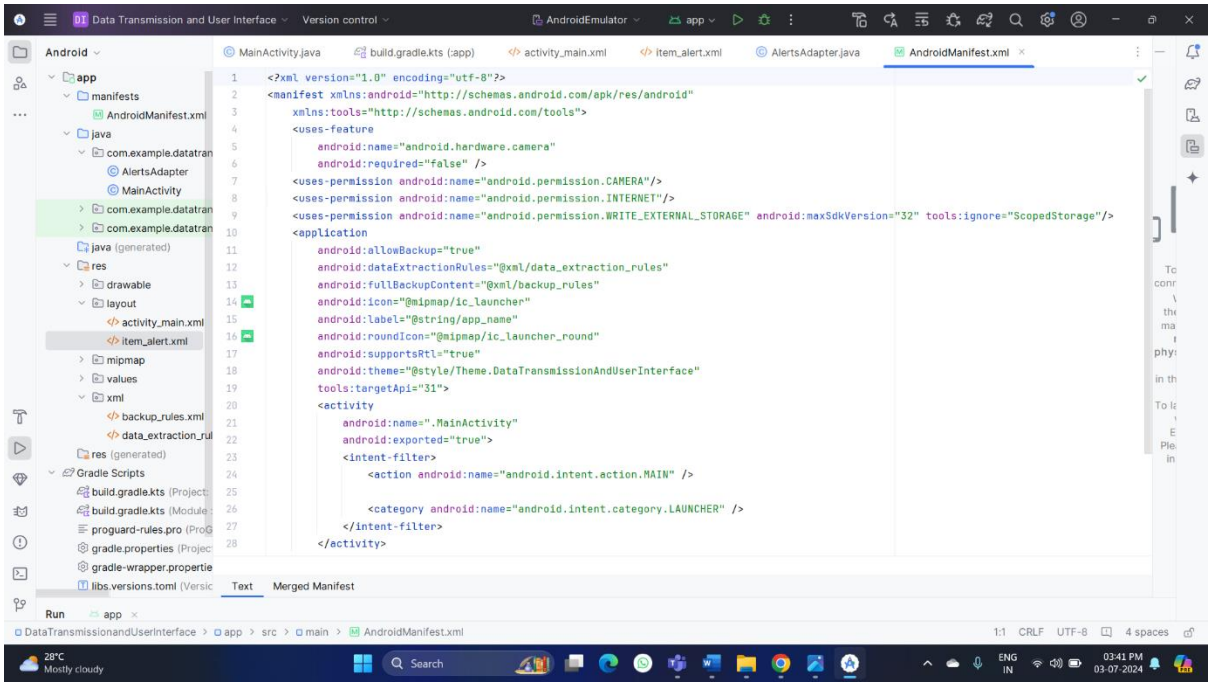
```
    public class MainActivity extends AppCompatActivity {
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
                    }
                }
            }
        }

        private Uri saveBitmapAndGetUri(Bitmap bitmap) {
            // Implement logic to save bitmap to a file and return its URI
            // Placeholder implementation
            String path = MediaStore.Images.Media.insertImage(getContentResolver(), bitmap, "CapturedImage", null);
            return Uri.parse(path);
        }

        private void showCapturedMediaAndAlert(Uri imageUri) {
            // Display captured image
            Glide.with(this).load(imageUri).into(imageMedia);

            // Update text in ListView
            alertData.clear();
            alertData.add("No Issues");
            alertData.add("Everything Looks Good");
            alertData.add("No Interruption");
            alertsAdapter.notifyDataSetChanged();

            showToast("Displaying captured media and alert");
        }

        private void showToast(String message) {
            Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
        }
```

AlertsAdapter.java

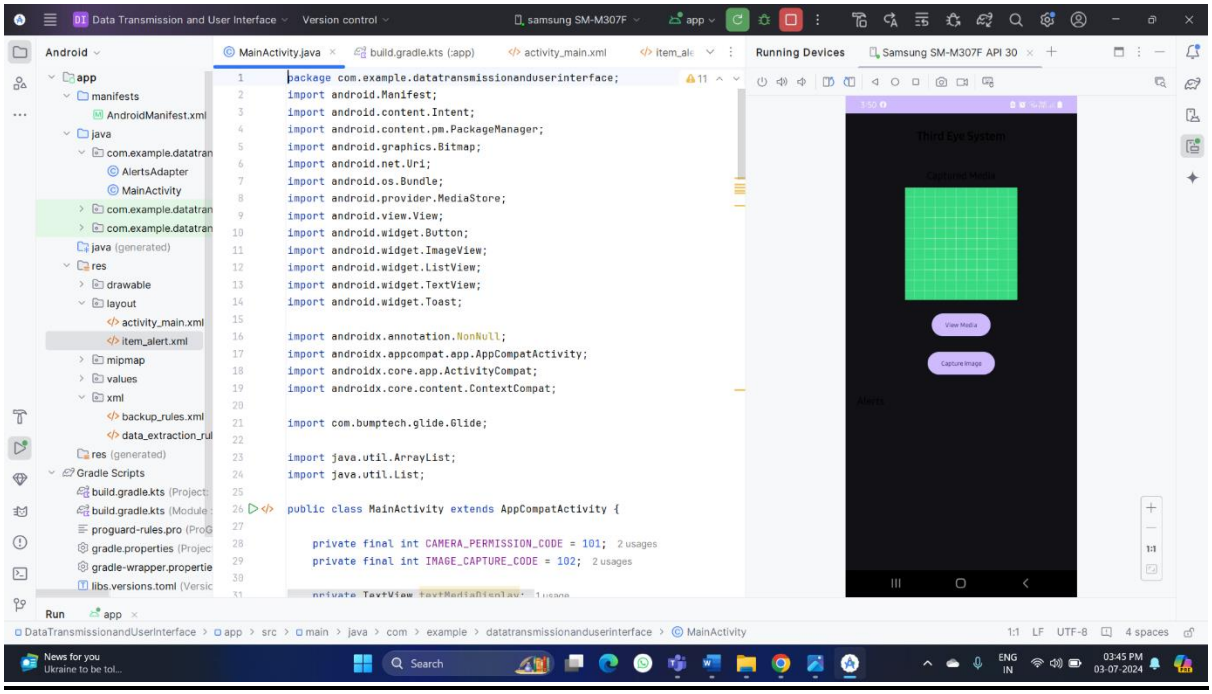


```
package com.example.datatransmissionanduserinterface;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;
import java.util.List;
public class AlertsAdapter extends ArrayAdapter<String> {
    private final Context context;
    private final List<String> alertData;
    public AlertsAdapter(Context context, List<String> alertData) {
        super(context, 0, alertData);
        this.context = context;
        this.alertData = alertData;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        String alert = getItem(position);
        View view = convertView;
        if (view == null) {
            view = LayoutInflater.from(context).inflate(R.layout.item_alert, parent, false);
        }
        TextView alertTextView = view.findViewById(R.id.alert_text);
        alertTextView.setText(alert);
        return view;
    }
}
```

AndroidManifest.xml

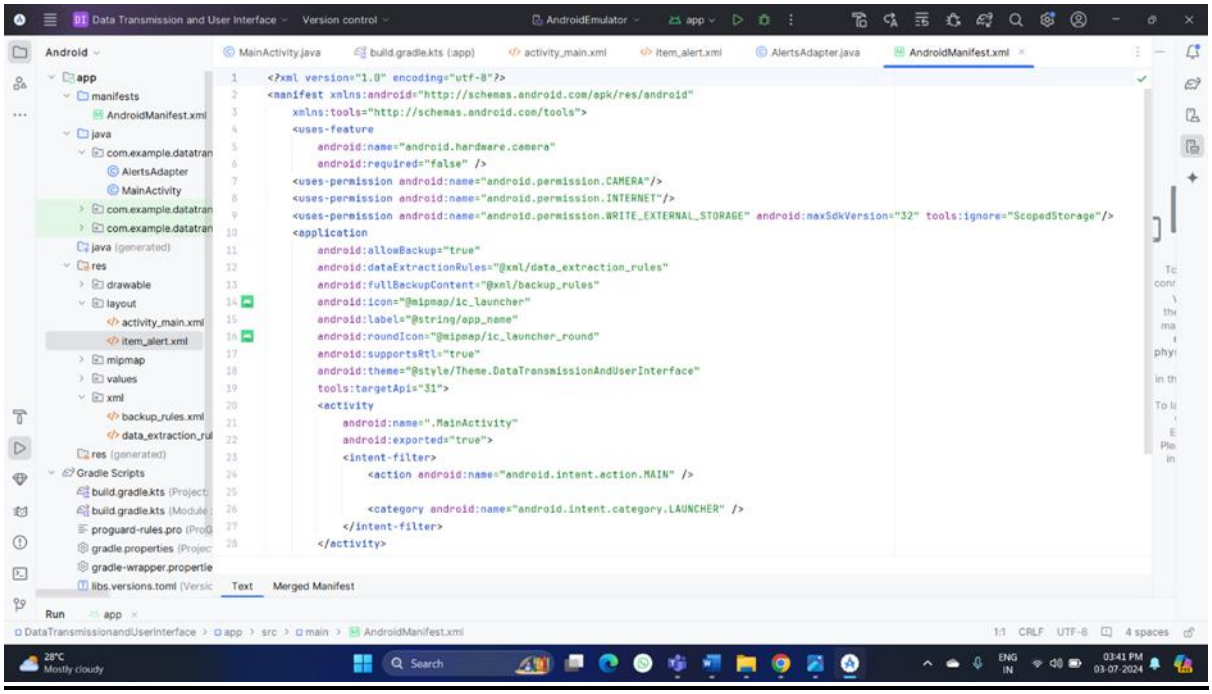


DAY 4 TASK 1 OUTPUT SCREENSHOTS

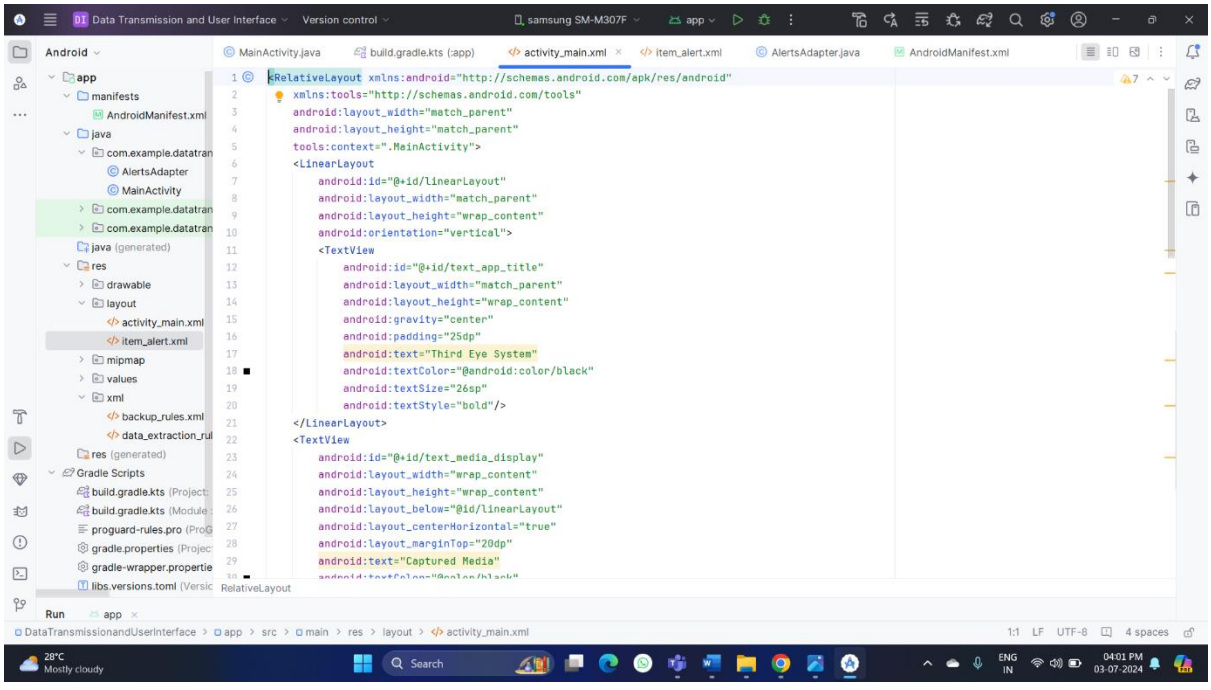


Task 2: Develop a user interface for the owner to interact with the Third Eye system, including viewing the captured media and receiving alerts.

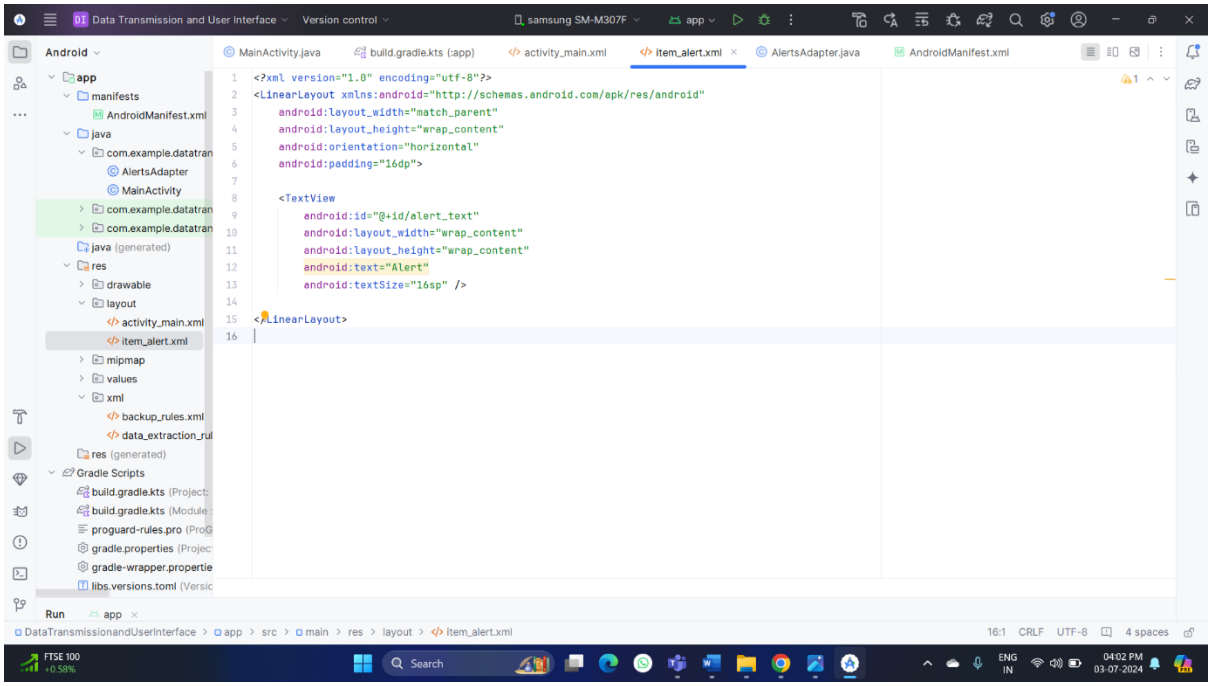
AndroidManifest.xml



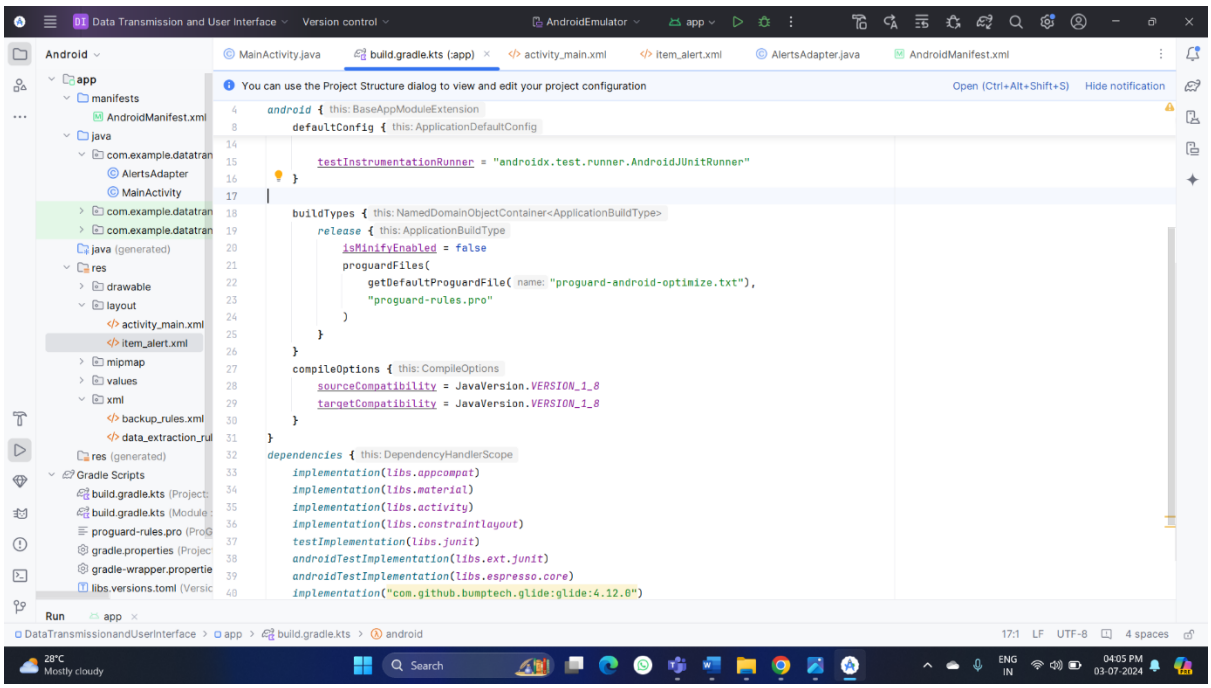
activity_main.xml



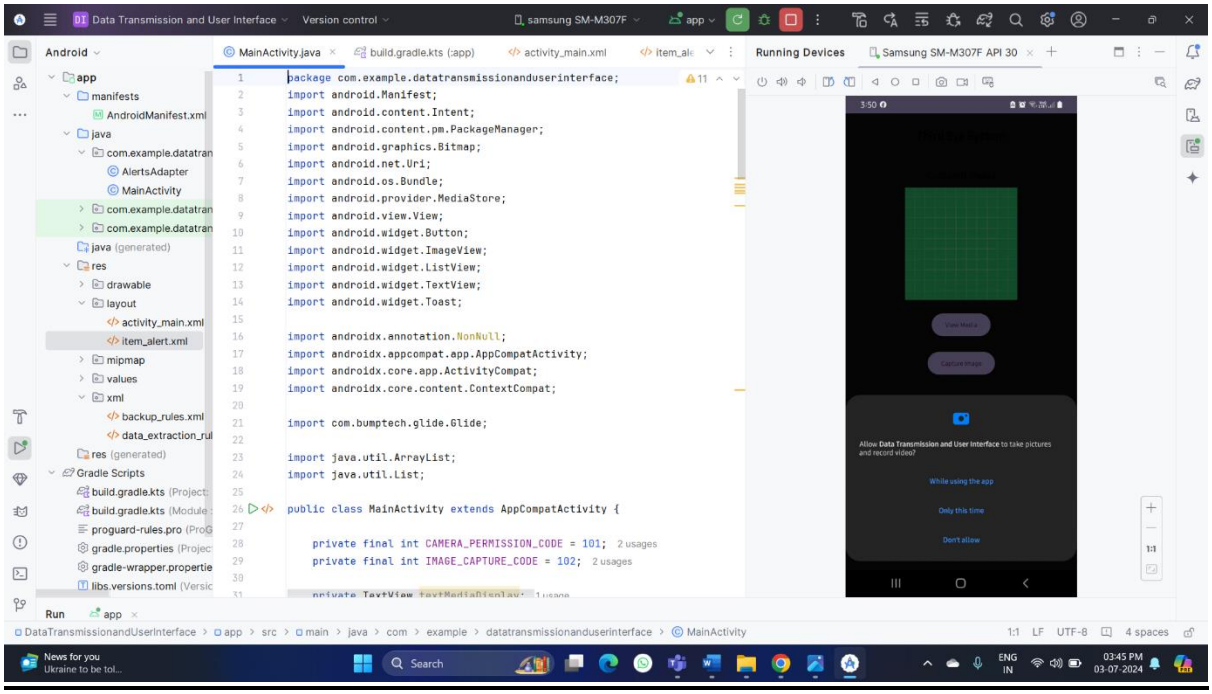
item_alert.xml

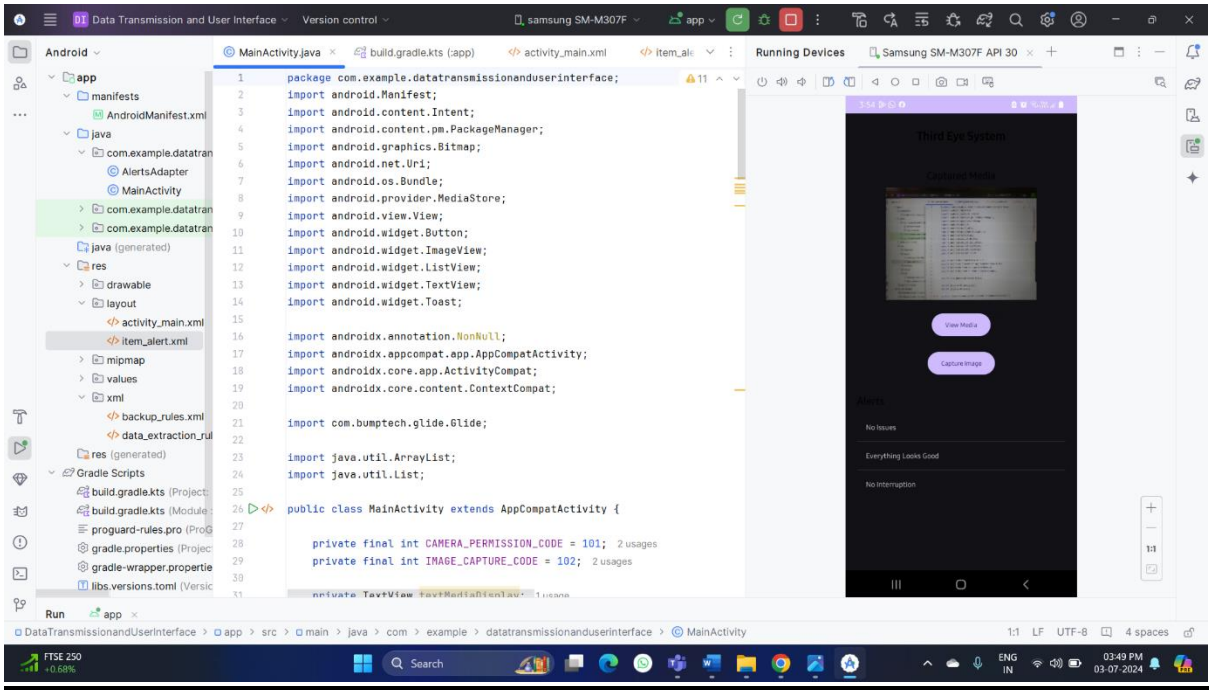


build.gradle.kts(:app)



DAY 4 TASK 2 OUTPUT SCREENSHOTS



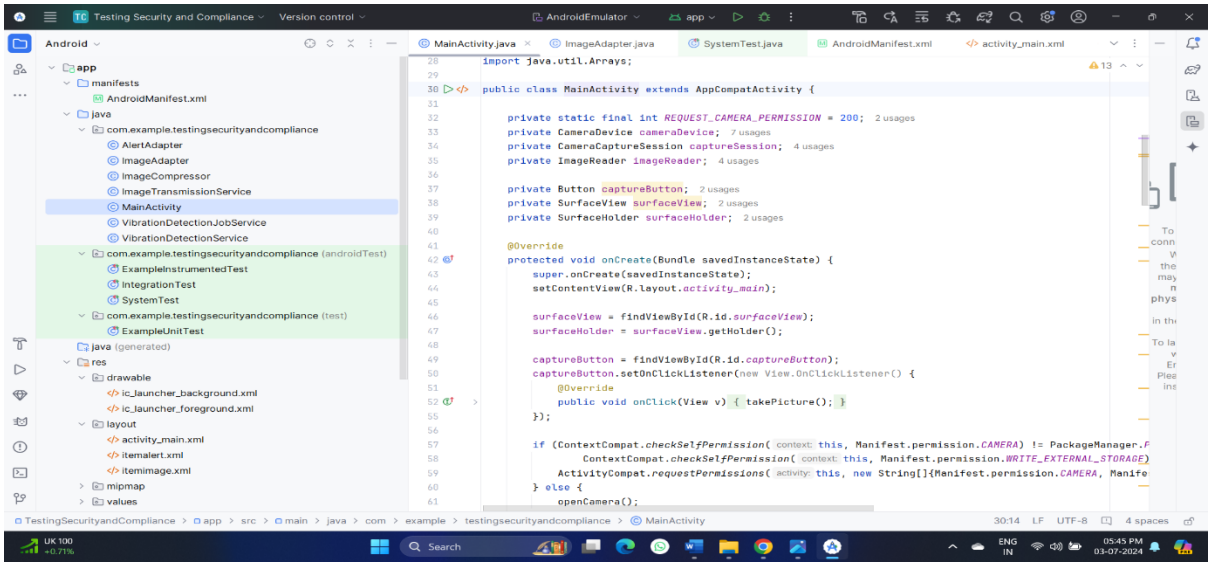


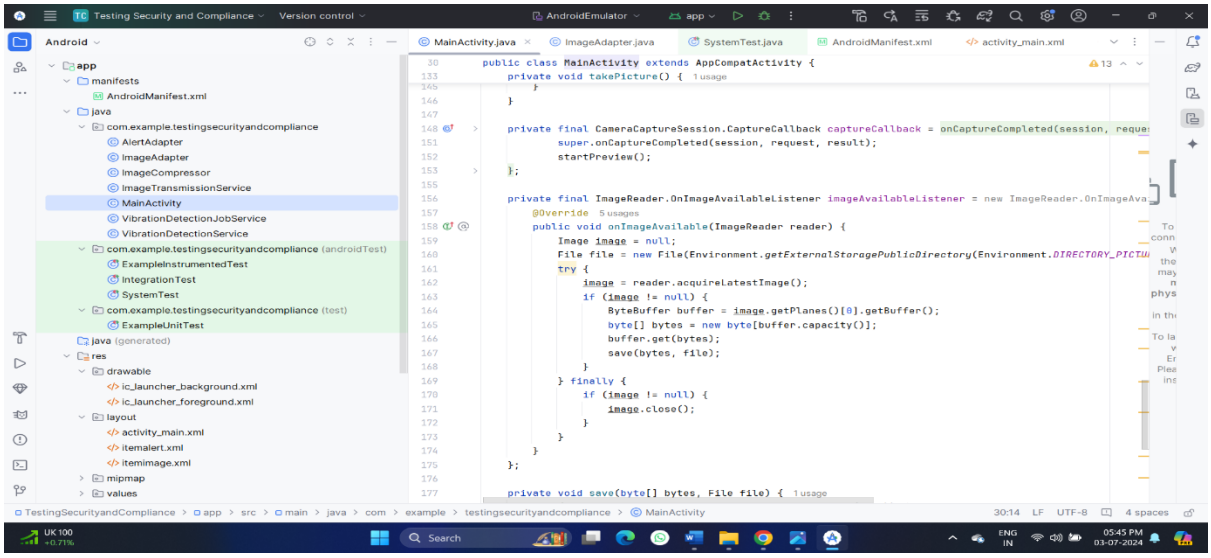
Day 5: Testing, Security, and Compliance

Task 1: Perform rigorous testing in various scenarios, including different lighting and weather conditions, to ensure the system's reliability.

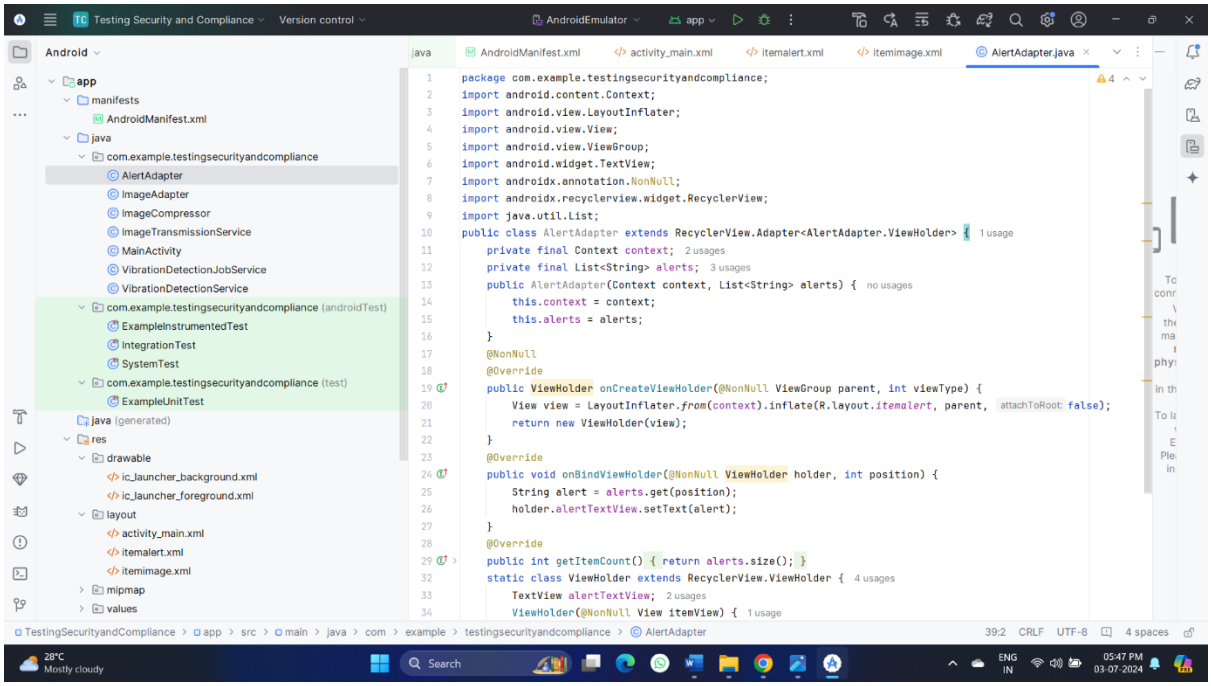
The Third Eye Automotive Surveillance system is designed to enhance vehicle security by leveraging the capabilities of an Android device. The system focuses on power management, background services, vibration detection, event-driven camera capture, image processing, storage, data transmission, and compliance with security and privacy regulations. The following document outlines the architecture, components, and implementation details of the system.

MainActivity.java



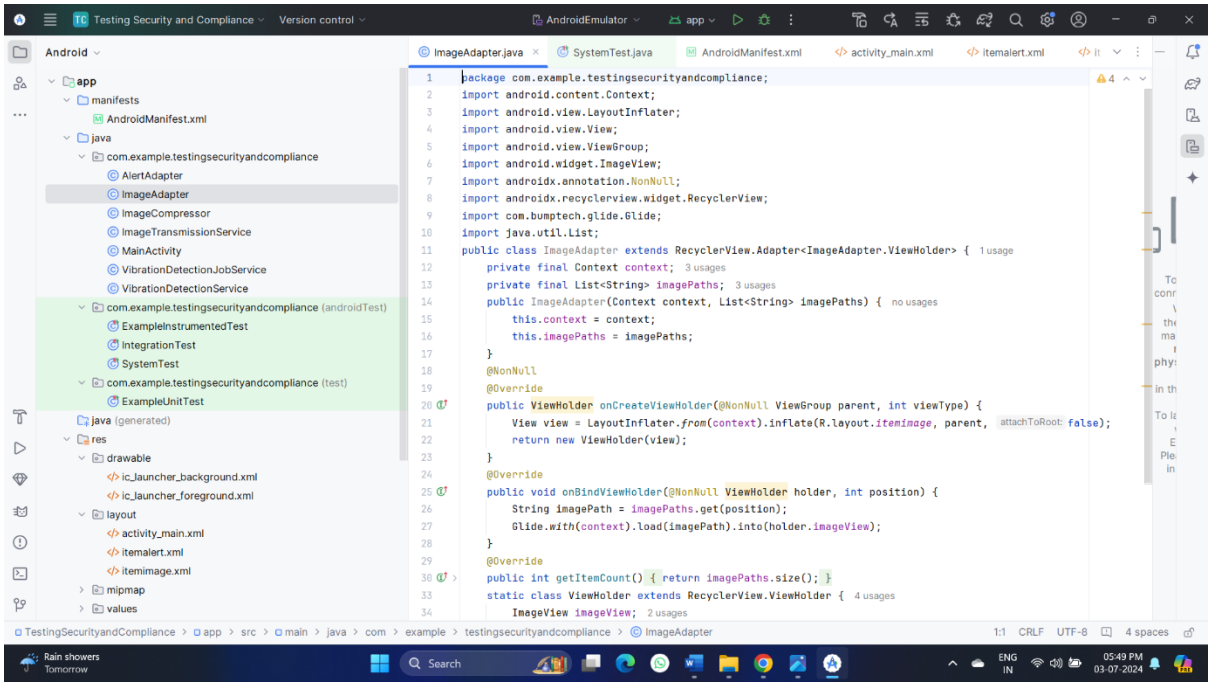


AlertAdapter.java



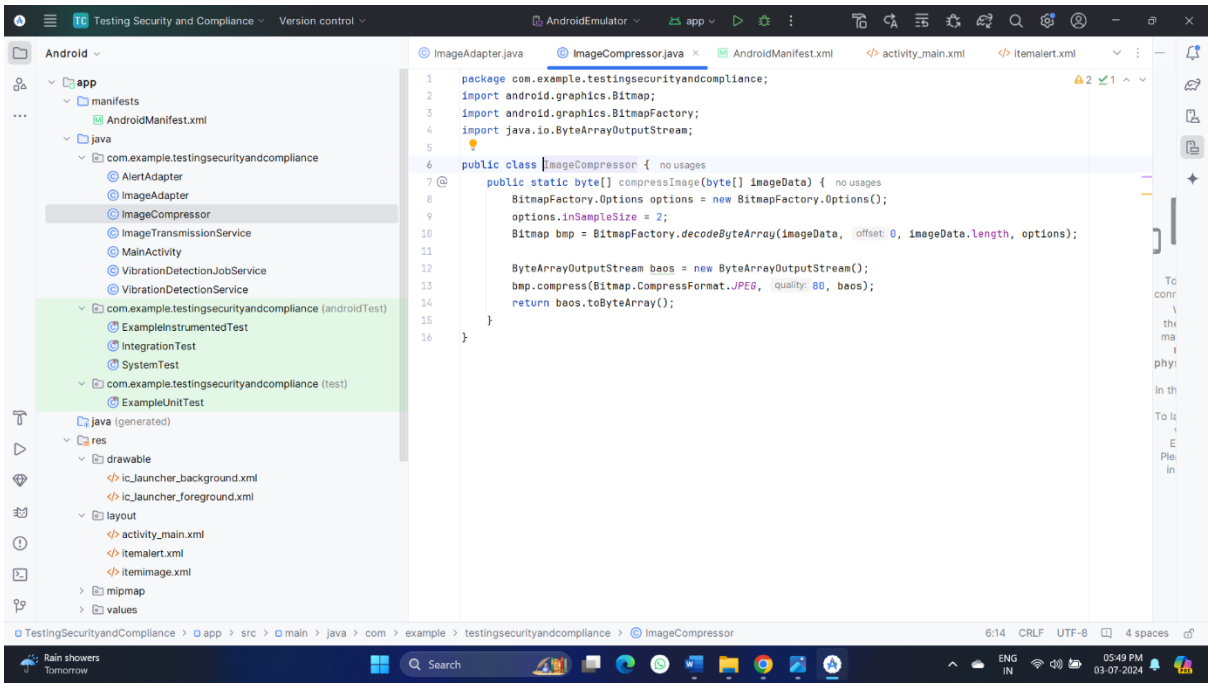
```
package com.example.testingsecurityandcompliance;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;
public class AlertAdapter extends RecyclerView.Adapter<AlertAdapter.ViewHolder> { 1 usage
    private final Context context; 2 usages
    private final List<String> alerts; 3 usages
    public AlertAdapter(Context context, List<String> alerts) { no usages
        this.context = context;
        this.alerts = alerts;
    }
    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(context).inflate(R.layout.itemalert, parent, attachToRoot: false);
        return new ViewHolder(view);
    }
    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        String alert = alerts.get(position);
        holder.alertTextView.setText(alert);
    }
    @Override
    public int getItemCount() { return alerts.size(); }
    static class ViewHolder extends RecyclerView.ViewHolder { 4 usages
        TextView alertTextView; 2 usages
        ViewHolder(@NonNull View itemView) { 1 usage
```

ImageAdapter.java

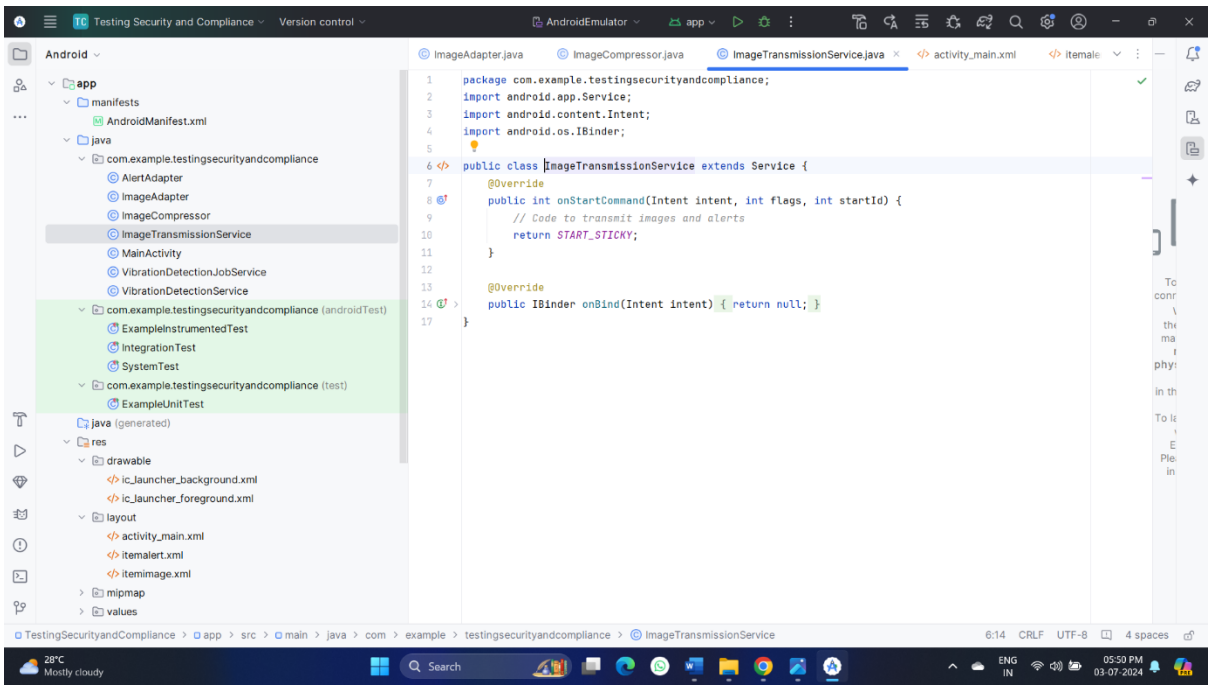


```
package com.example.testingsecurityandcompliance;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import com.bumptech.glide.Glide;
import java.util.List;
public class ImageAdapter extends RecyclerView.Adapter<ImageAdapter.ViewHolder> { 1 usage
    private final Context context; 3 usages
    private final List<String> imagePaths; 3 usages
    public ImageAdapter(Context context, List<String> imagePaths) { no usages
        this.context = context;
        this.imagePaths = imagePaths;
    }
    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(context).inflate(R.layout.itemimage, parent, attachToRoot: false);
        return new ViewHolder(view);
    }
    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        String imagePath = imagePaths.get(position);
        Glide.with(context).load(imagePath).into(holder.imageView);
    }
    @Override
    public int getItemCount() { return imagePaths.size(); }
    static class ViewHolder extends RecyclerView.ViewHolder { 4 usages
        ImageView imageView; 2 usages
```

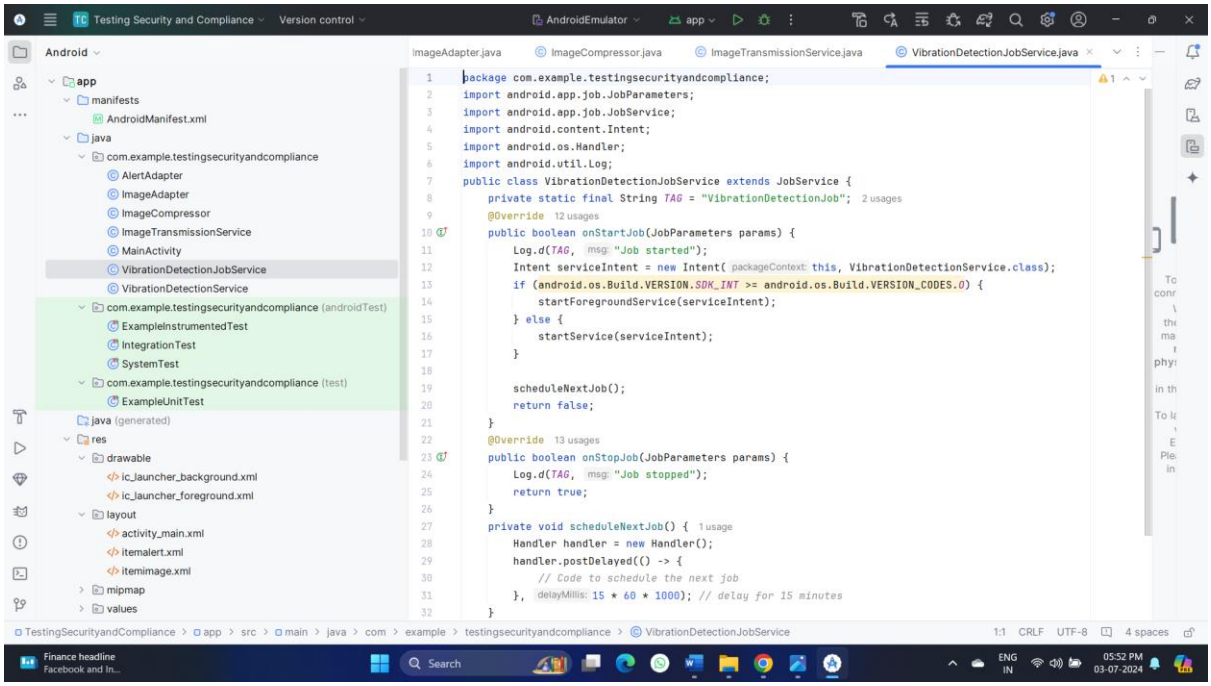
ImageCompressor.java



ImageTransmissionService.java

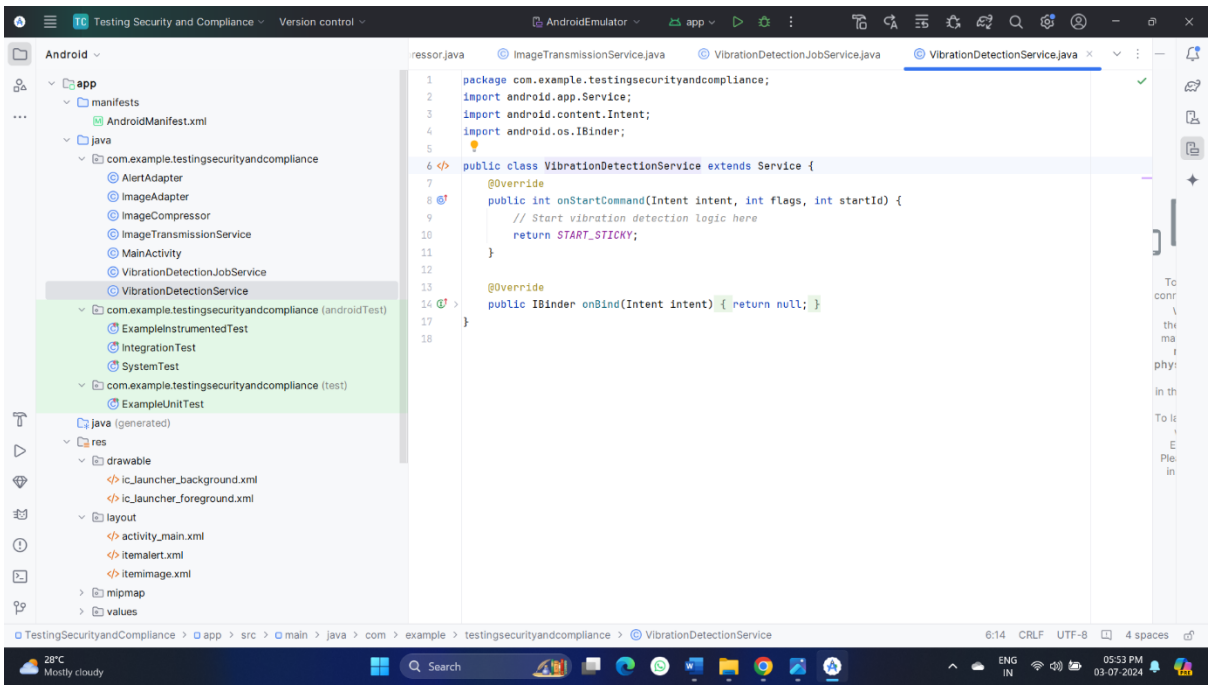


VibrationDetectionJobService.java



```
package com.example.testingsecurityandcompliance;
import android.app.job.JobParameters;
import android.app.job.JobService;
import android.content.Intent;
import android.os.Handler;
import android.util.Log;
public class VibrationDetectionJobService extends JobService {
    private static final String TAG = "VibrationDetectionJob";  2 usages
    @Override  12 usages
    public boolean onStartJob(JobParameters params) {
        Log.d(TAG,  msg: "Job started");
        Intent serviceIntent = new Intent( packageContext: this, VibrationDetectionService.class);
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
            startForegroundService(serviceIntent);
        } else {
            startService(serviceIntent);
        }

        scheduleNextJob();
        return false;
    }
    @Override  13 usages
    public boolean onStopJob(JobParameters params) {
        Log.d(TAG,  msg: "Job stopped");
        return true;
    }
    private void scheduleNextJob() {  1 usage
        Handler handler = new Handler();
        handler.postDelayed(() -> {
            // Code to schedule the next job
        },  delayMillis: 15 * 60 * 1000); // delay for 15 minutes
    }
```

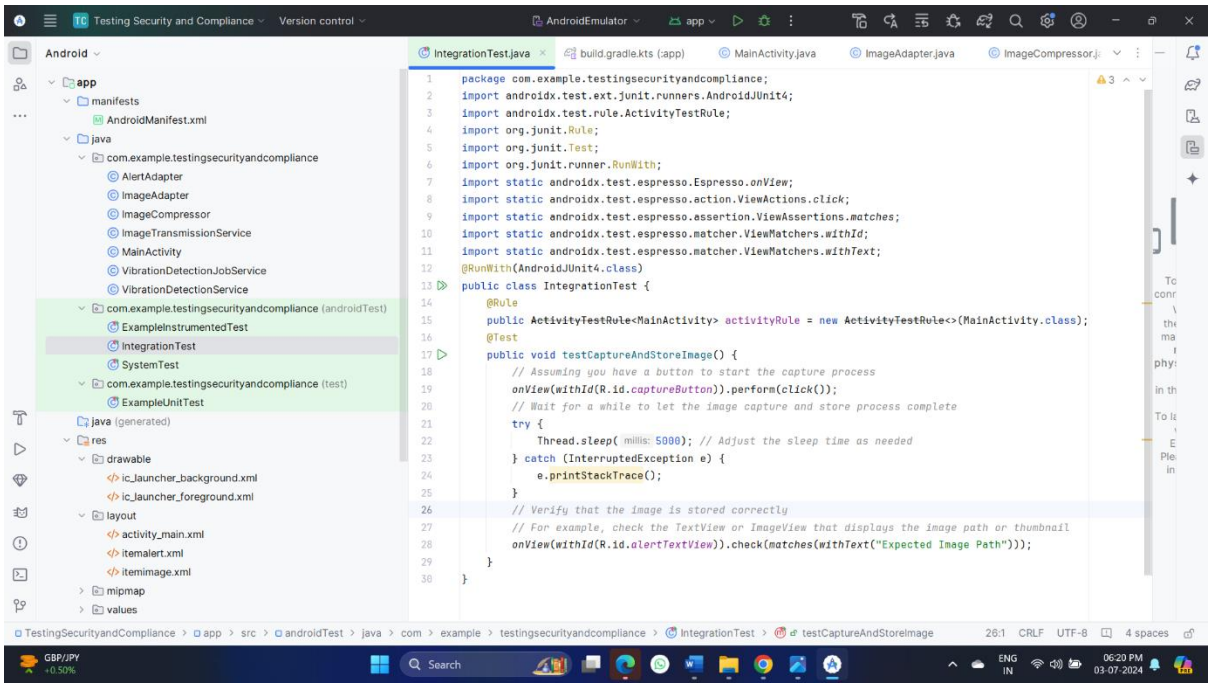
VibrationDetectionService.java



```
package com.example.testingsecurityandcompliance;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class VibrationDetectionService extends Service {
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // Start vibration detection logic here
        return START_STICKY;
    }

    @Override
    public IBinder onBind(Intent intent) { return null; }
}
```

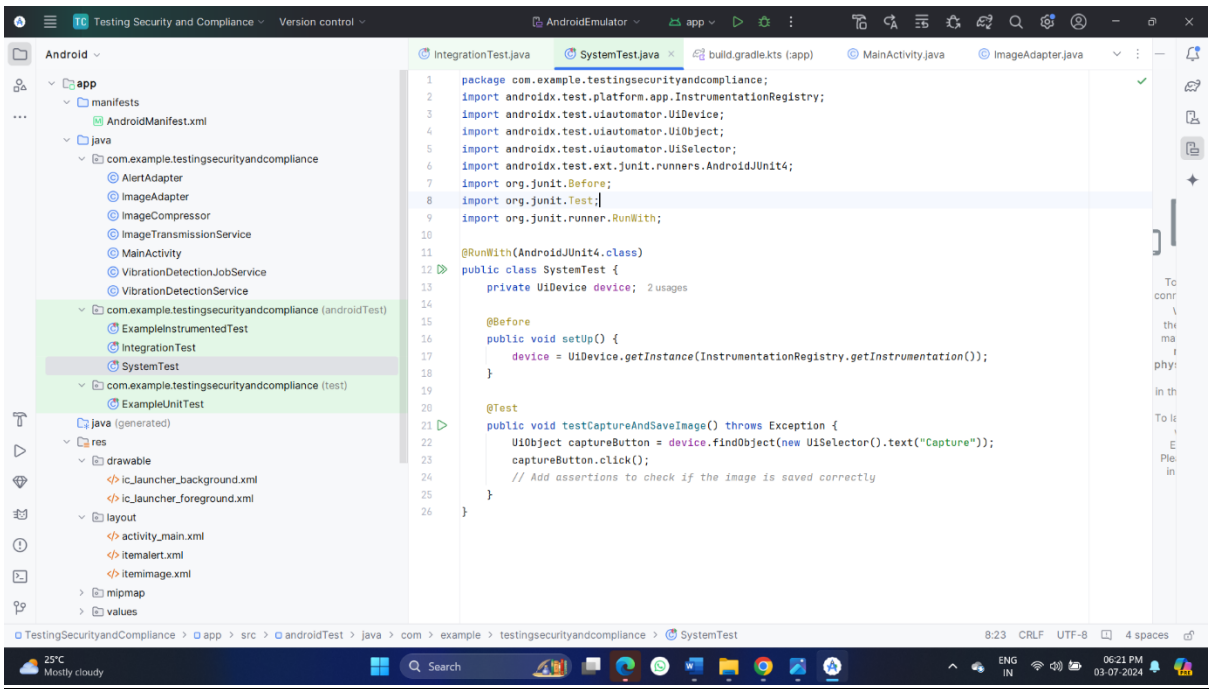
androidTest

IntegrationTest.java



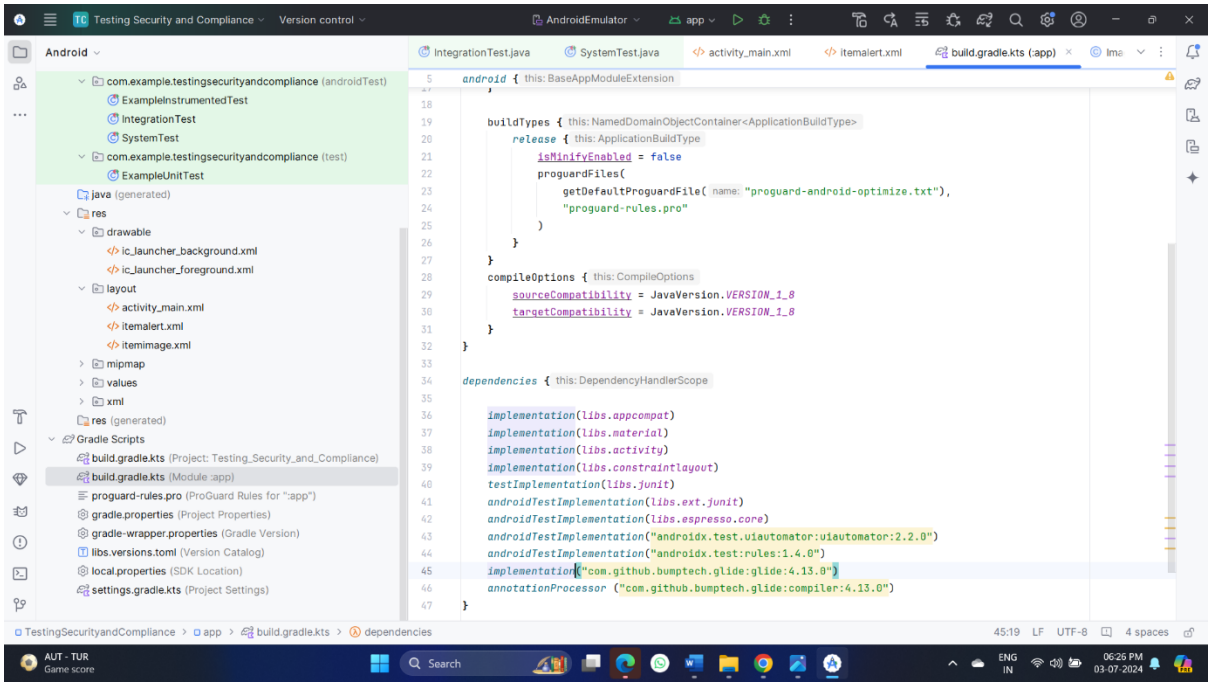
```java
package com.example.testingsecurityandcompliance;
import androidx.test.ext.junit.runners.AndroidJUnit4;
import androidx.test.rule.ActivityTestRule;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;
import static androidx.test.espresso.Espresso.onView;
import static androidx.test.espresso.action.ViewActions.click;
import static androidx.test.espresso.assertion.ViewAssertions.matches;
import static androidx.test.espresso.matcher.ViewMatchers.withId;
import static androidx.test.espresso.matcher.ViewMatchers.withText;
@RunWith(AndroidJUnit4.class)
public class IntegrationTest {
    @Rule
    public ActivityTestRule<MainActivity> activityRule = new ActivityTestRule<>(MainActivity.class);
    @Test
    public void testCaptureAndStoreImage() {
        // Assuming you have a button to start the capture process
        onView(withId(R.id.captureButton)).perform(click());
        // Wait for a while to let the image capture and store process complete
        try {
            Thread.sleep( millis: 5000); // Adjust the sleep time as needed
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        // Verify that the image is stored correctly
        // For example, check the TextView or ImageView that displays the image path or thumbnail
        onView(withId(R.id.alertTextView)).check(matches(withText("Expected Image Path")));
    }
}
```

SystemTest.java

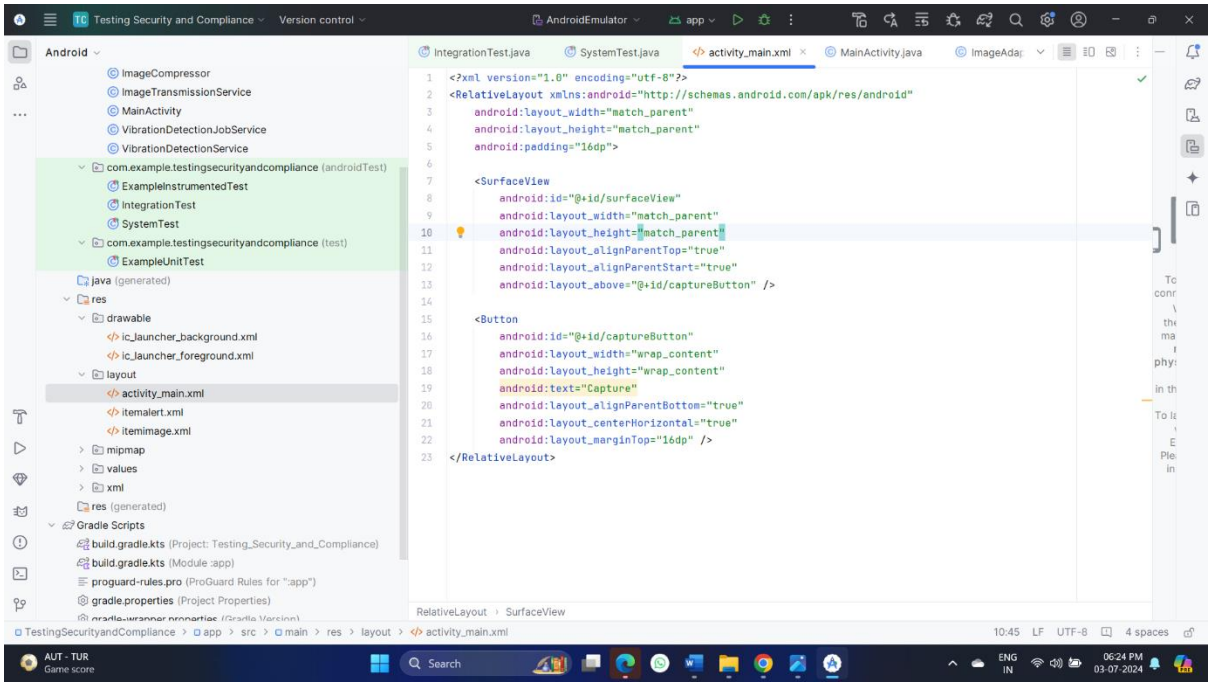


```java
package com.example.testingsecurityandcompliance;
import androidx.test.platform.app.InstrumentationRegistry;
import androidx.test.uiautomator.UiDevice;
import androidx.test.uiautomator.UiObject;
import androidx.test.uiautomator.UiSelector;
import androidx.test.ext.junit.runners.AndroidJUnit4;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

@RunWith(AndroidJUnit4.class)
public class SystemTest {
    private UiDevice device;  2 usages

    @Before
    public void setUp() {
        device = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation());
    }

    @Test
    public void testCaptureAndSaveImage() throws Exception {
        UiObject captureButton = device.findObject(new UiSelector().text("Capture"));
        captureButton.click();
        // Add assertions to check if the image is saved correctly
    }
}
```

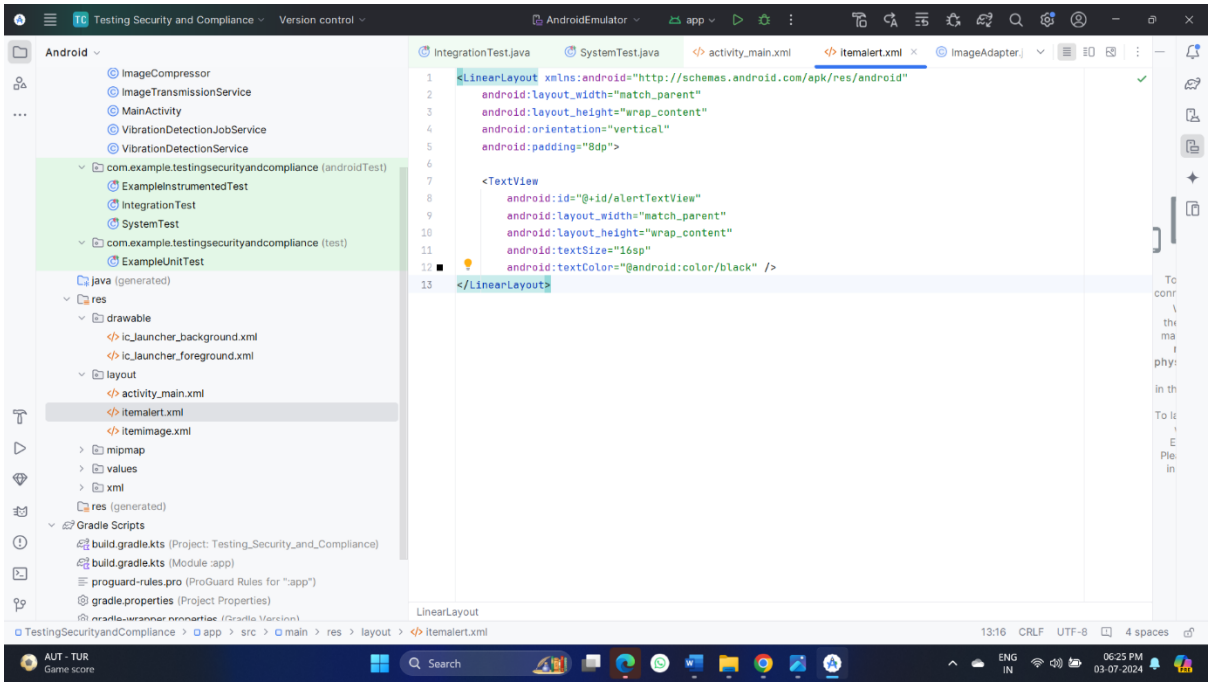
build.gradle.kts(Module:app)



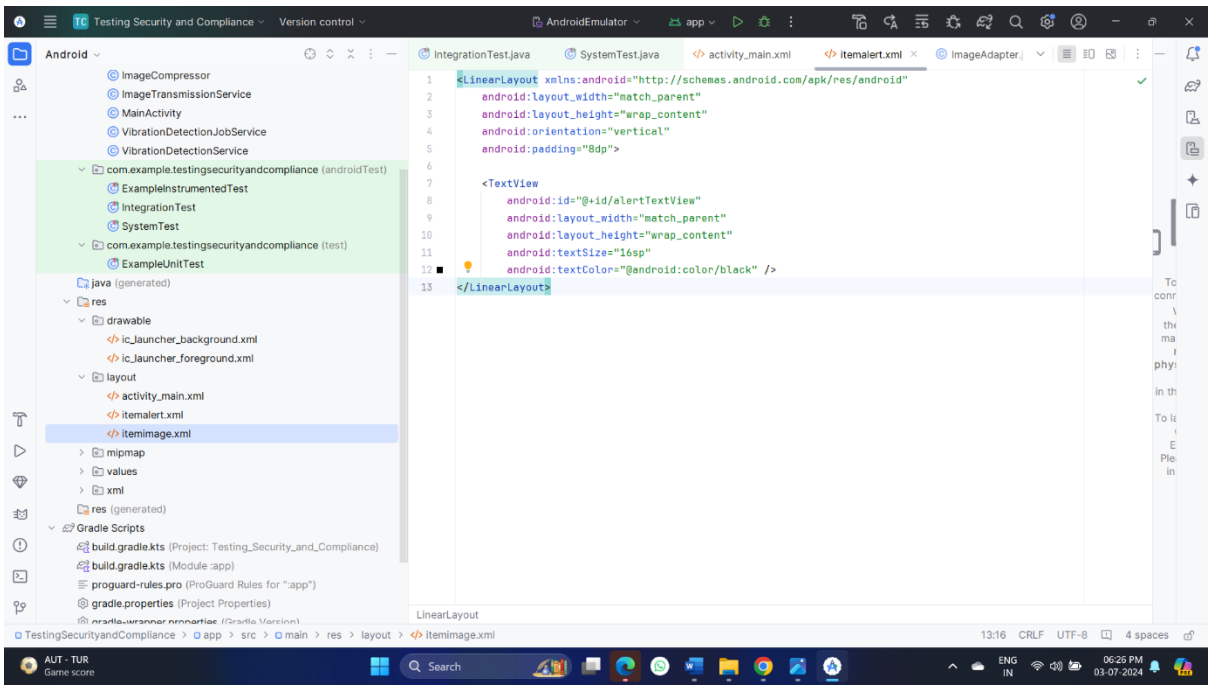
activity_main.xml



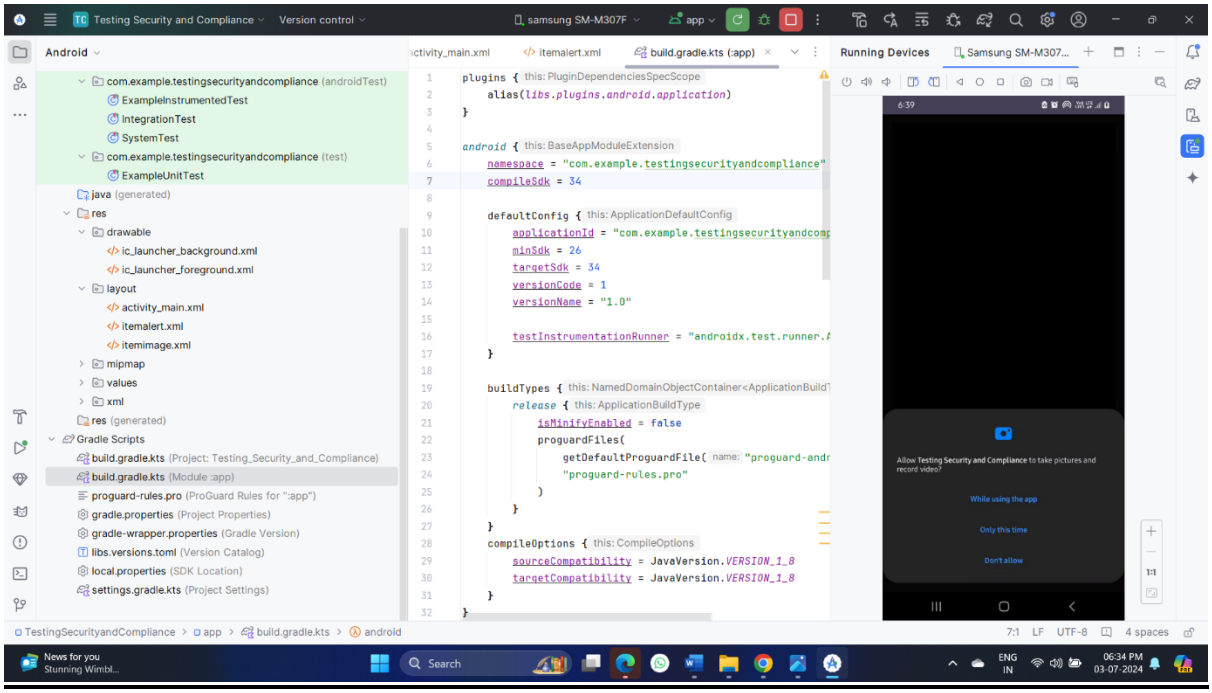
itemalert.xml

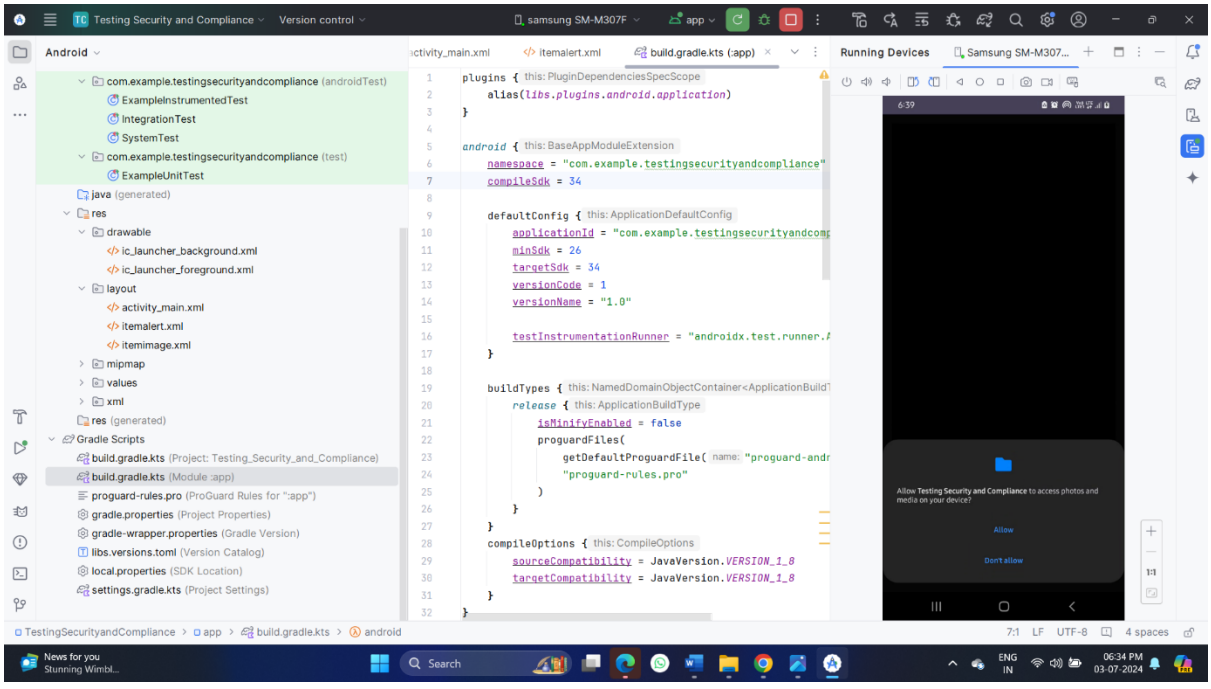


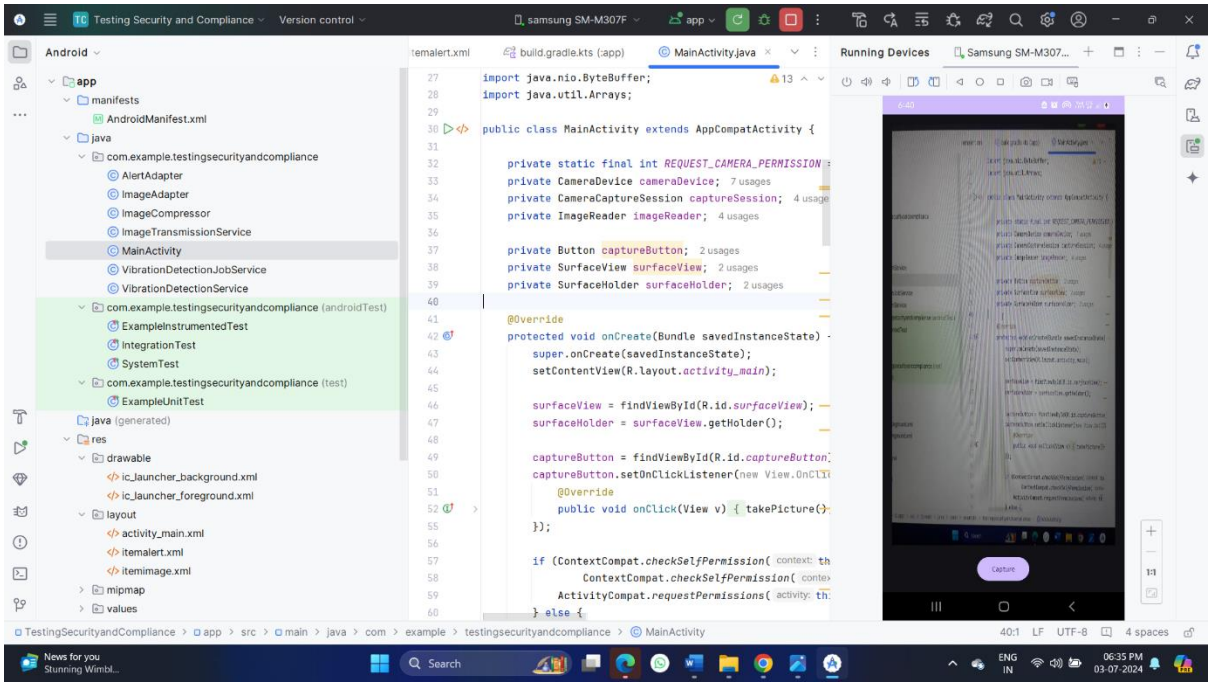
itemimage.xml



DAY 5 TASK 1 OUTPUT SCREENSHOTS







Task 2: Ensure that all data capture and transmission are secure and comply with privacy laws and automotive regulations.

Data security and compliance with privacy laws are critical components of the Third Eye Automotive Surveillance system. This ensures that user data is protected from unauthorized access and that the system adheres to relevant legal standards. Our approach encompasses data encryption, secure transmission, and compliance checks.

Data Security

Encryption

Data at Rest:

- All captured images and videos stored locally on the device are encrypted using Advanced Encryption Standard (AES) with a 256-bit key. This ensures that even if the data storage is compromised, the data remains unreadable without the decryption key.

Data in Transit:

- Data transmitted to the vehicle owner's smartphone or designated device is encrypted using Transport Layer Security (TLS). TLS provides end-to-end encryption, ensuring that data cannot be intercepted or tampered with during transmission.

Implementation:

- **Encryption Libraries:** Use well-established encryption libraries such as Bouncy Castle or the Android Keystore system for managing cryptographic keys and performing encryption/decryption operations.
- **Key Management:** Ensure that encryption keys are securely stored and managed. Use hardware-backed key storage where available.

Authentication and Authorization

- **User Authentication:** Implement strong user authentication mechanisms, such as multi-factor authentication (MFA), to ensure that only authorized users can access the surveillance system and its data.
- **Access Control:** Use role-based access control (RBAC) to define and enforce user permissions, ensuring that users can only access data and functions relevant to their roles.

Secure Coding Practices

- Follow secure coding practices to prevent common vulnerabilities such as SQL injection, cross-site scripting (XSS), and buffer overflows.
- Conduct regular code reviews and use static code analysis tools to identify and fix security vulnerabilities.

Compliance with Privacy Laws

Data Minimization

- **Principle:** Collect only the data that is necessary for the intended purpose and avoid excessive data collection.
- **Implementation:** Ensure that the system collects only the required data, such as images and timestamps, and avoids capturing unnecessary personal information.

Data Anonymization

- **Principle:** Where possible, anonymize data to protect user privacy.
- **Implementation:** Remove or obfuscate any personally identifiable information (PII) from the data before storage or transmission.

User Consent

- **Principle:** Obtain explicit consent from users before capturing and transmitting data.
- **Implementation:** Implement a user consent mechanism within the app, requiring users to agree to data capture and transmission policies before using the system.

Data Retention and Deletion

- **Principle:** Retain data only for as long as necessary and securely delete it when it is no longer needed.
- **Implementation:** Implement a data retention policy and automated mechanisms to delete data after a specified period or upon user request.

Compliance with Automotive Regulations

Regulatory Standards

- **Principle:** Adhere to relevant automotive industry standards and regulations, such as those from the National Highway Traffic Safety Administration (NHTSA) or the European Union General Data Protection Regulation (GDPR).
- **Implementation:** Regularly review and update the system to comply with evolving standards and regulations.

Safety Considerations

- **Principle:** Ensure that the surveillance system does not interfere with the vehicle's operation or safety features.
- **Implementation:** Conduct safety assessments and testing to verify that the system operates safely and does not introduce risks to the vehicle or its occupants.

Documentation and Auditing

- **Principle:** Maintain comprehensive documentation of data security and compliance measures.

- **Implementation:** Document all encryption methods, authentication mechanisms, compliance checks, and data retention policies. Conduct regular audits to verify compliance and address any gaps.

Conclusion

Ensuring data security and compliance with privacy laws and automotive regulations is essential for the Third Eye Automotive Surveillance system. By implementing strong encryption, authentication, and access control measures, adhering to privacy principles, and complying with regulatory standards, we can protect user data and ensure that the system operates within legal boundaries. Regular documentation and auditing further strengthen our commitment to data security and compliance.