

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.

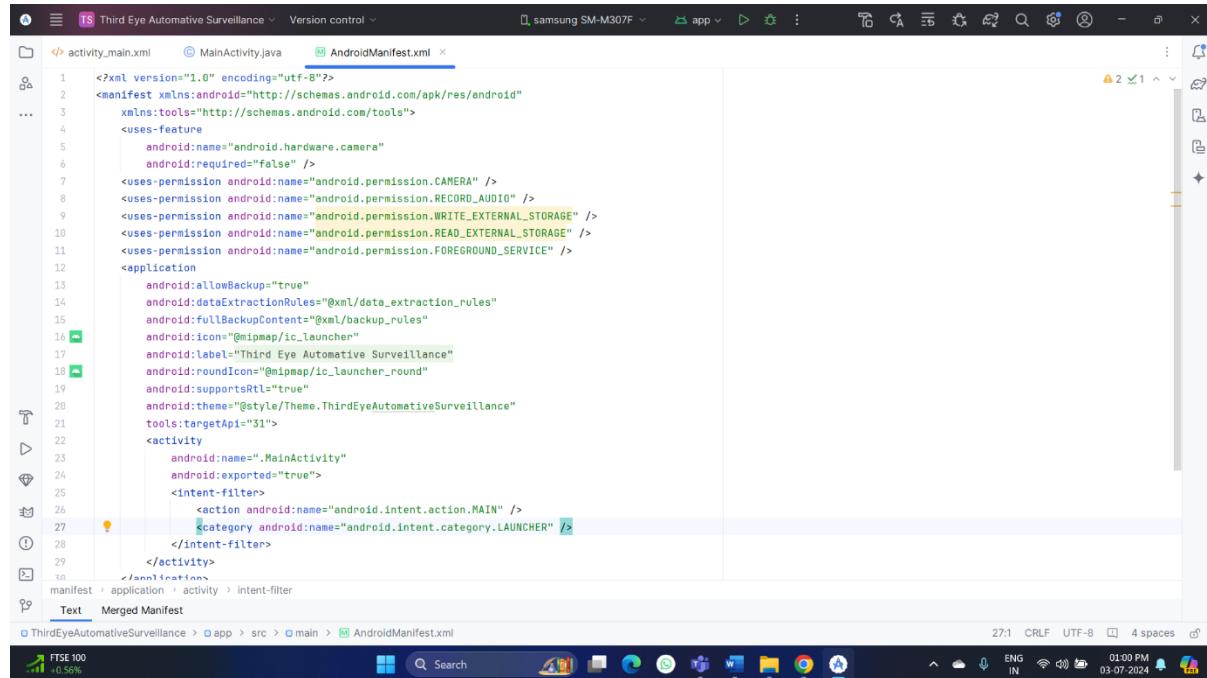
Conclusion: Camera-based surveillance systems for automotive security are integral in modern vehicles, offering enhanced protection, safety features, and peace of mind to vehicle owners and operators. Understanding these systems' capabilities and functionalities is essential for developing effective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

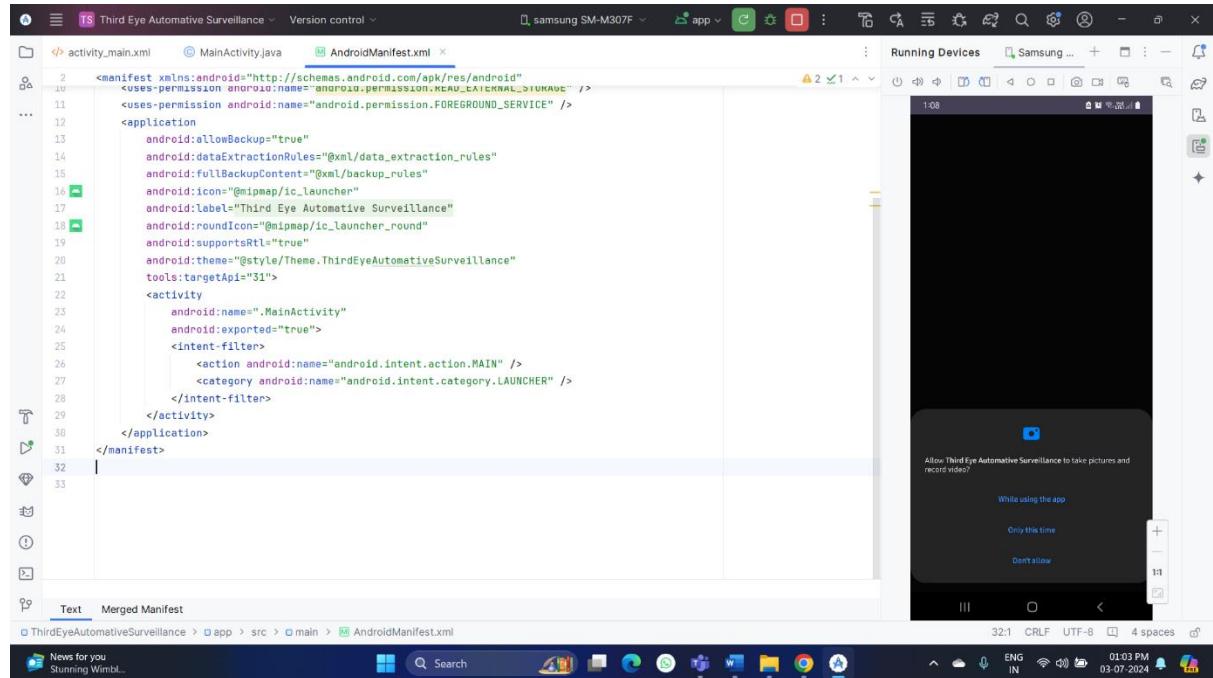
1. Install Android Studio: Download and install the latest version of Android Studio from the [official website](#).
2. Create a New Project:
 - Open Android Studio.
 - Click on "Start a new Android Studio project."
 - Select "Empty Activity" and click "Next."
 - Set the project name to ThirdAutomativeEyeSurveillance.
 - 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:



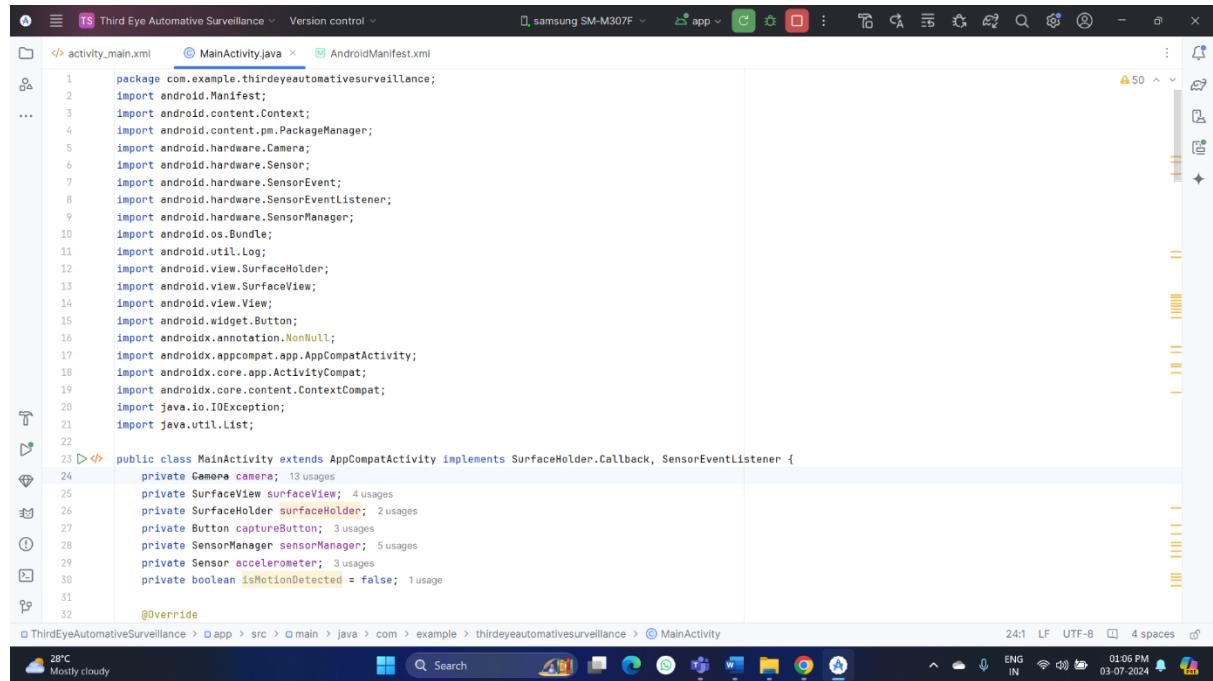
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Third Eye Automatic Surveillance"
        android:roundIcons="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ThirdEyeAutomotiveSurveillance"
        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>
```

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 Automotive Surveillance

```
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, Manifest.permission.CAMERA)
            != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(activity, 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
                        public void onPictureTaken(byte[] data, Camera camera) {
                            // Handle the image data
                            Log.d(tag: "CameraCapture", msg: "Picture taken");
                        }
                    });
                }
            }
        });
    }
}
```

28°C Mostly cloudy

```
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, SensorEventListener {
    private Camera.Size getOptimalPreviewSize(List<Camera.Size> sizes, int width, int height) {
        if (sizes == null) return null;

        Camera.Size optimalSize = null;
        double minDiff = Double.MAX_VALUE;

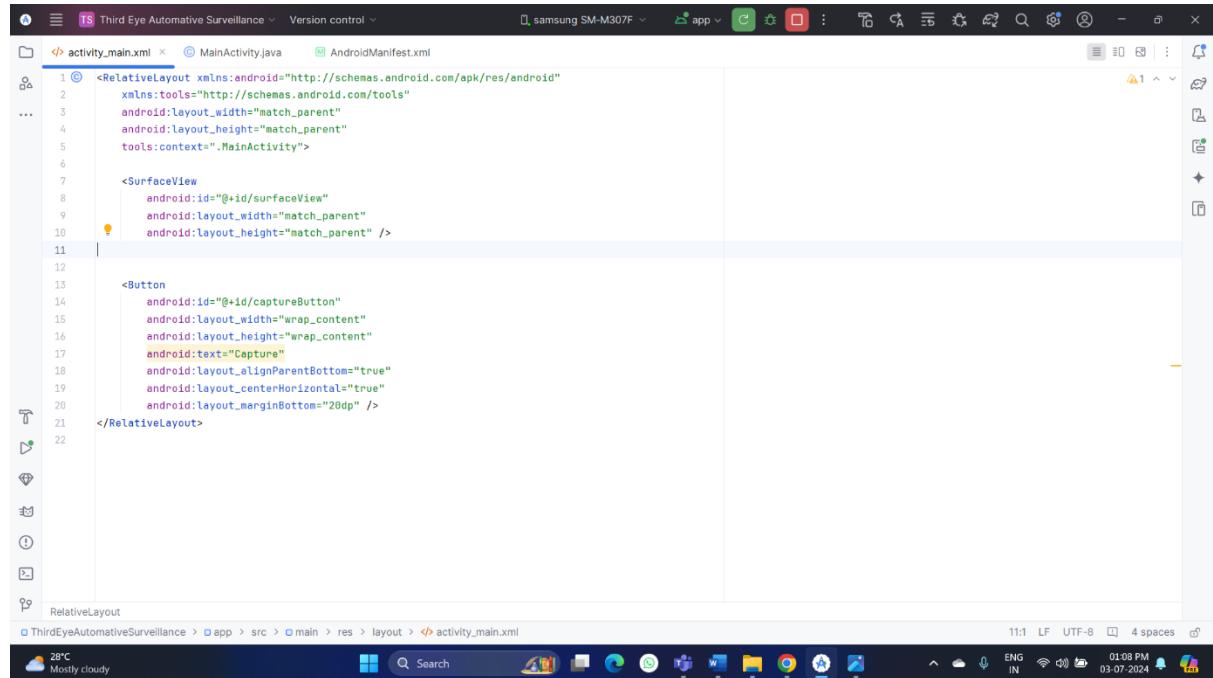
        int targetHeight = height;

        // 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;
    }
}
```

28°C Mostly cloudy

activity_main.xml



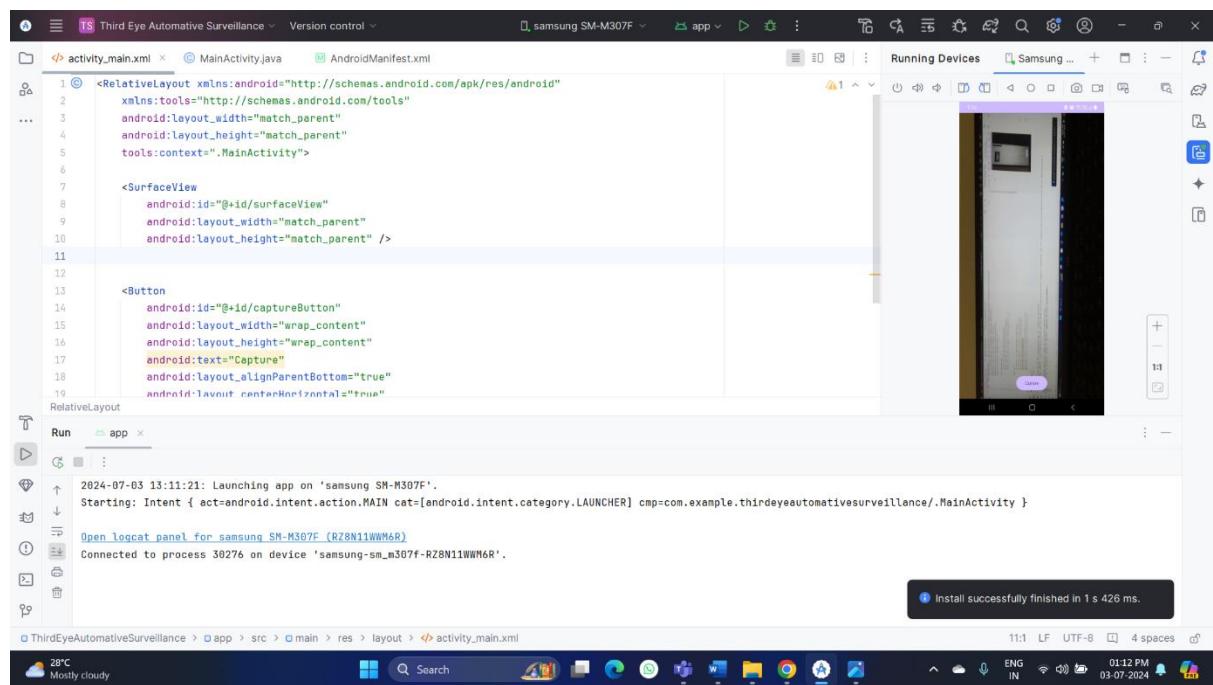
The screenshot shows the Android Studio interface with the XML code for `activity_main.xml`. The code defines a `RelativeLayout` containing a `SurfaceView` and a `Button`.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"

    <SurfaceView
        android:id="@+id/surfaceView"
        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"
        android:layout_marginBottom="20dp" />
</RelativeLayout>
```

DAY 1 TASK 3 OUTPUT SCREENSHOT



The screenshot shows the Android Studio interface with the XML code for `activity_main.xml` and the running application on a Samsung SM-M307F device. The app displays a camera preview with a capture button.

Log output:

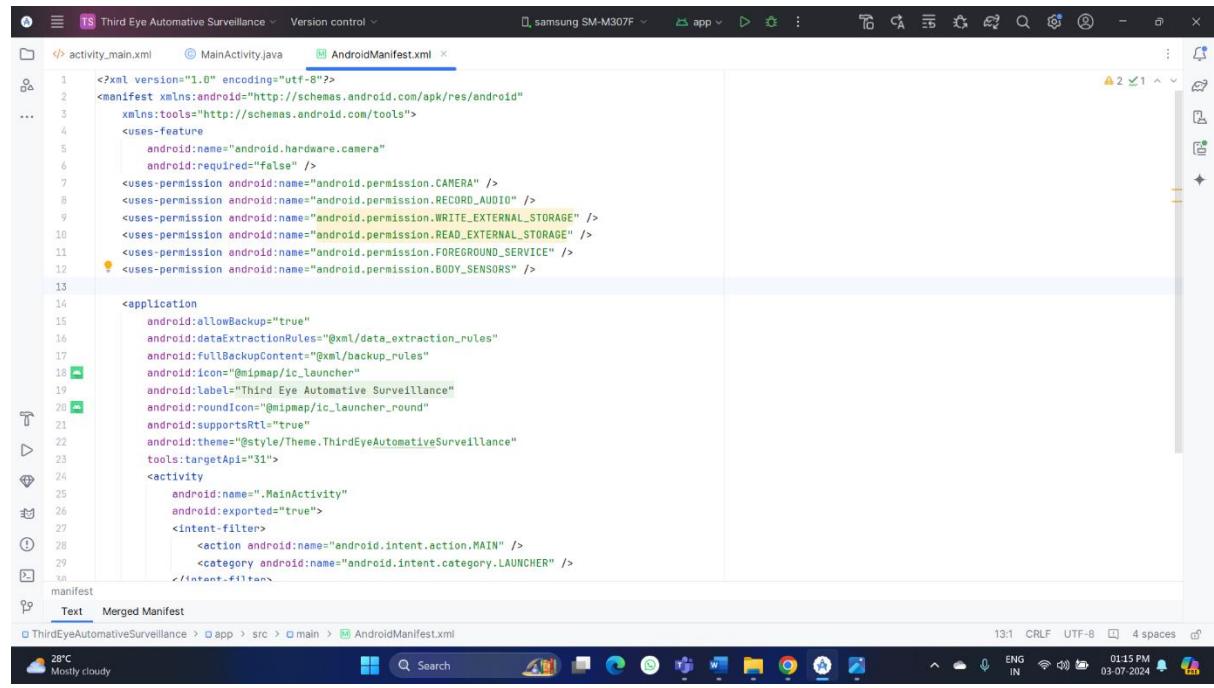
```
2024-07-03 13:11:21: Launching app on 'samsung SM-M307F'.
Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.example.thirdeyeautomativesurveillance/.MainActivity }
Open logcat panel for samsung SM-M307F (R28N11WW6R)
Connected to process 30276 on device 'samsung-sm_m307f-R28N11WW6R'.
```

Message bar:

```
Install successfully finished in 1 s 426 ms.
```

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

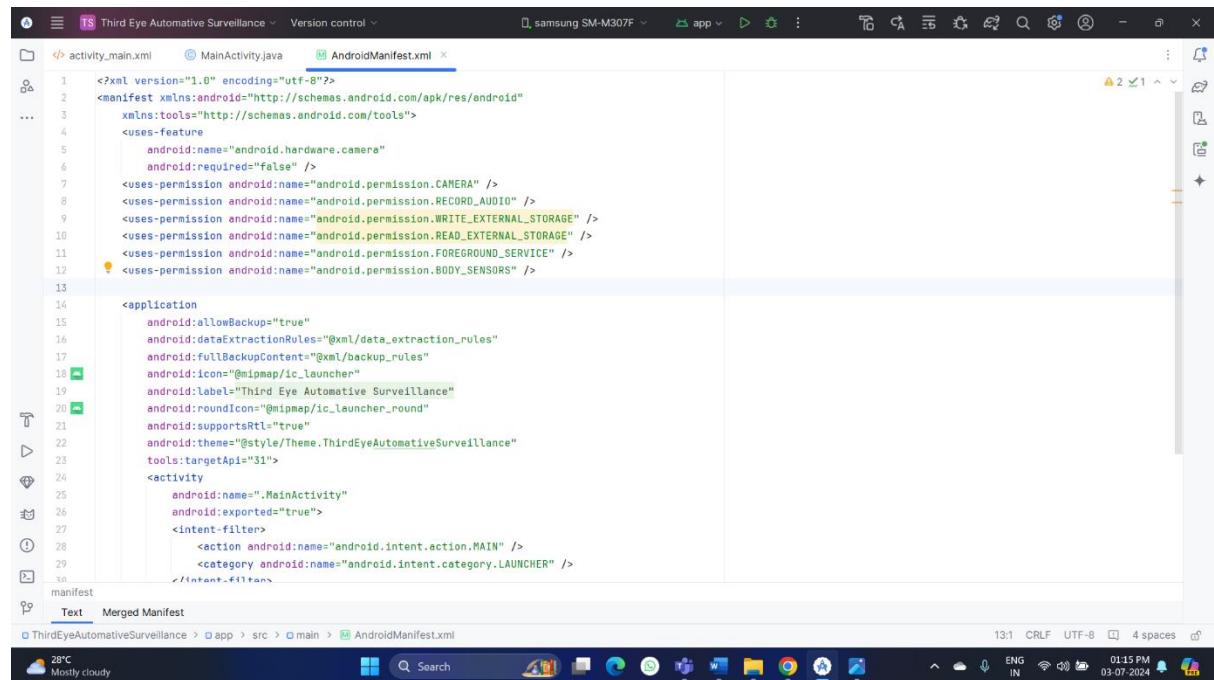
Update AndroidManifest.xml



The screenshot shows the Android Studio interface with the 'AndroidManifest.xml' file selected in the project tree. The code editor displays the XML manifest file, which includes permissions for camera, audio recording, and external storage, along with basic application metadata like icon and theme. The status bar at the bottom shows the date, time, and battery level.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
    <uses-permission android:name="android.permission.BODY_SENSORS" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Third Eye Automotive Surveillance"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ThirdEyeAutomotiveSurveillance"
        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>
```

Update MainActivity.java (Extended with Motion Detection):



The screenshot shows the Android Studio interface with the 'MainActivity.java' file selected in the project tree. The code editor displays Java code that extends the 'Activity' class and overrides the 'onSensorChanged' method to handle motion sensor data. The status bar at the bottom shows the date, time, and battery level.

```
public class MainActivity extends Activity {
    SensorManager sensorManager;
    Sensor accelerometer;
    Handler handler;
    long lastTime;
    float currentX, currentY, currentZ;
    float previousX, previousY, previousZ;
    float rateX, rateY, rateZ;
    float lowX, lowY, lowZ;
    float highX, highY, highZ;
    int count = 0;
    int maxCount = 10;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        handler = new Handler();
        lastTime = System.currentTimeMillis();
    }

    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            currentX = event.values[0];
            currentY = event.values[1];
            currentZ = event.values[2];
            if (System.currentTimeMillis() - lastTime > 100) {
                count++;
                if (count > maxCount) {
                    Intent intent = new Intent("android.intent.action.MAIN");
                    intent.addCategory("android.intent.category.LAUNCHER");
                    sendBroadcast(intent);
                }
                lastTime = System.currentTimeMillis();
            }
        }
    }
}
```

```
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, SensorEventListener {
    public void onSensorChanged(SensorEvent event) {
        float x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];

        double acceleration = Math.sqrt(x * x + y * y + z * z) - SensorManager.GRAVITY_EARTH;
        if (acceleration > 2) { // threshold for motion detection
            isMotionDetected = true;
            captureButton.performClick();
        }
    }

    @Override 3 usages
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Not used
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (accelerometer != null) {
            sensorManager.registerListener(listener, this, SensorManager.SENSOR_DELAY_NORMAL);
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }
}
```

DAY 1 TASK 4 OUTPUT SCREENSHOTS

Android

app

manifests

AndroidManifest.xml

Java

res

Gradle Scripts

Running Devices

Samsung ...

```
package com.example.thirdeyeautomativesurveillance;
import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
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.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Button;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
```

2024-07-03 13:20:59: Launching app on 'samsung SM-M307F'.

Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.example.thirdeyeautomativesurveillance/.MainActivity }

Open locate panel for samsung SM-M307F (RZBN11WWH6R)

Connected to process 31056 on device 'samsung-sm_m307f-RZBN11WWH6R'.

News for you...
Stunning Wimbl...

Day 2: Vibration Detection and Event-Driven Capture

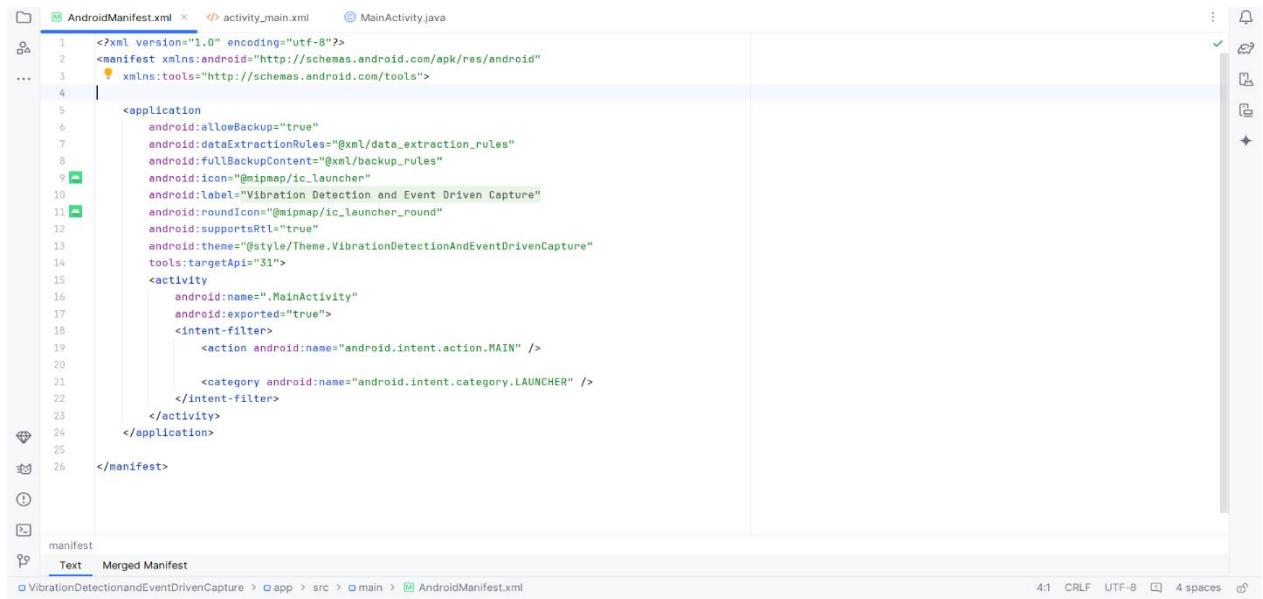
Objective: Implement a robust system to detect vehicle vibrations, capture images during vibrations, process and compress images for efficient storage, and store images with essential metadata.

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

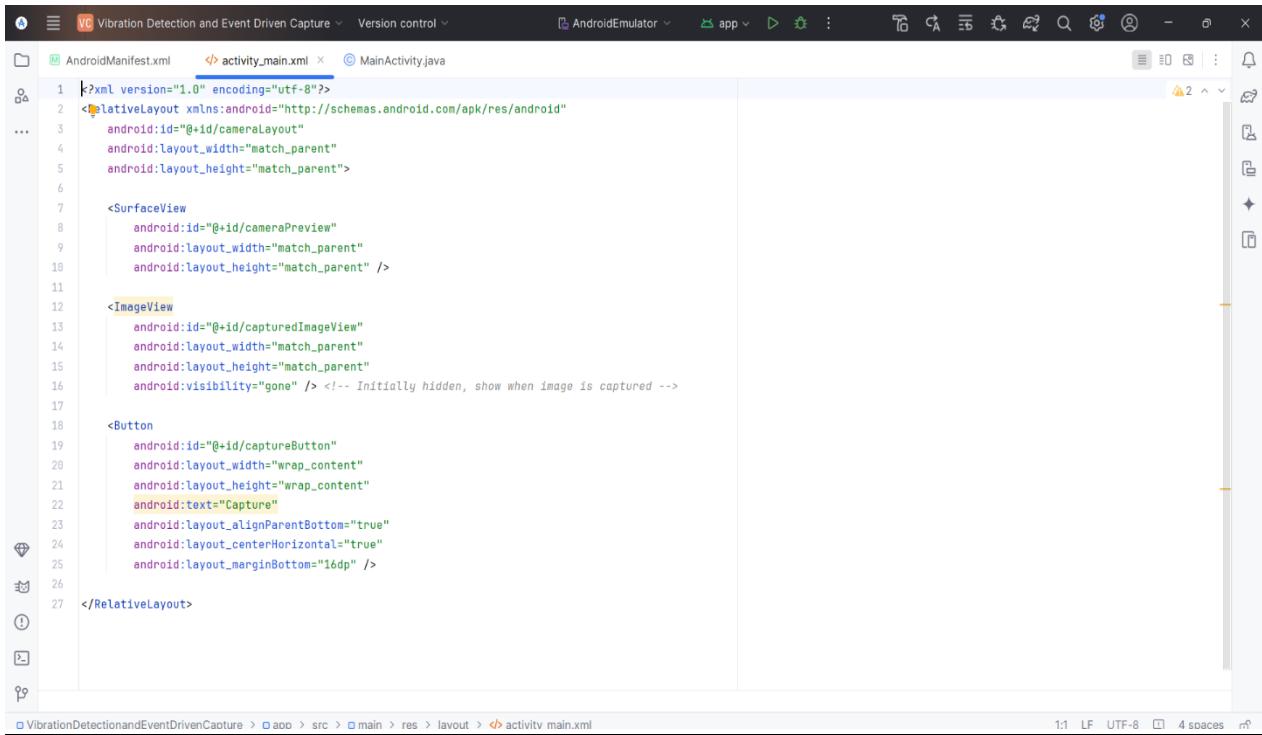
Vibration detection utilizes the accelerometer sensor available in Android devices. This sensor measures acceleration along three axes (x, y, z), enabling the detection of significant vibrations beyond a predefined threshold. The system continuously monitors accelerometer data to identify potential security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

AndroidManifest.xml



```
<?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="Vibration Detection and Event Driven Capture"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.VibrationDetectionAndEventDrivenCapture"
        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>
```

activity_main.xml



The screenshot shows the Android Studio interface with the XML layout file `activity_main.xml` open. The code defines a `RelativeLayout` containing a `SurfaceView`, an `ImageView`, and a `Button`. The `SurfaceView` and `ImageView` have `android:layout_width="match_parent"` and `android:layout_height="match_parent"`. The `Button` has `android:text="Capture"`, `android:layout_alignParentBottom="true"`, and `android:layout_centerHorizontal="true"`.

```
<?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">

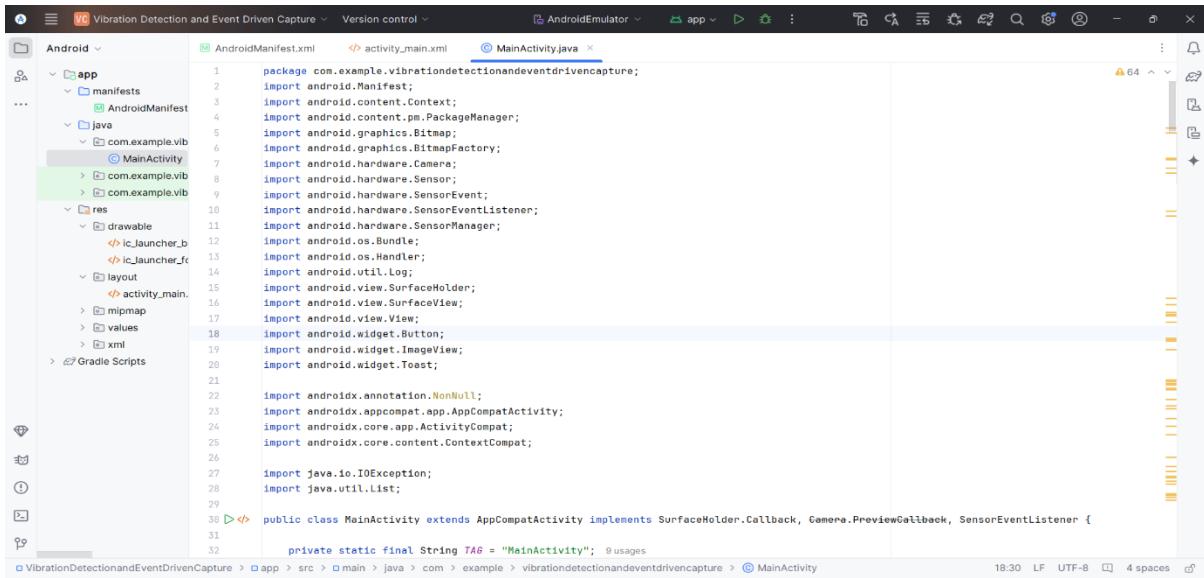
    <SurfaceView
        android:id="@+id/cameraPreview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <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



The screenshot shows the Android Studio interface with the Java code for `MainActivity.java` open. The code imports various Android classes and implements `SurfaceHolder.Callback`, `Camera.PreviewCallback`, and `SensorEventListener`. It defines a class `MainActivity` that extends `AppCompatActivity` and implements the required interfaces.

```
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
```

The screenshot shows the Android Studio interface with the project navigation bar at the top. The main area displays the `MainActivity.java` file. The code implements a `MainActivity` class that extends `AppCompatActivity` . It initializes a `Camera` , `SurfaceView` , and `ImageView` . It also handles sensor events and captures images every 2 seconds using a `Handler` and `Runnable` .

```
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {  
    private static final String TAG = "MainActivity";  
    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  
            }  
        }  
    }  
}
```

This screenshot shows the same `MainActivity.java` file with additional code added. The `onCreate` method is now fully implemented, setting up the `SurfaceView` , `SurfaceHolder` , and `Camera` . The `captureRunnable` is also fully implemented, handling the camera's `takePicture` method.

```
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
```

```

    public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {
        ...
        @Override
        public void onSensorChanged(SensorEvent event) {
            if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
                // Detect vibration based on accelerometer data
                float[] values = event.values;
                float x = values[0];
                float y = values[1];
                float z = values[2];

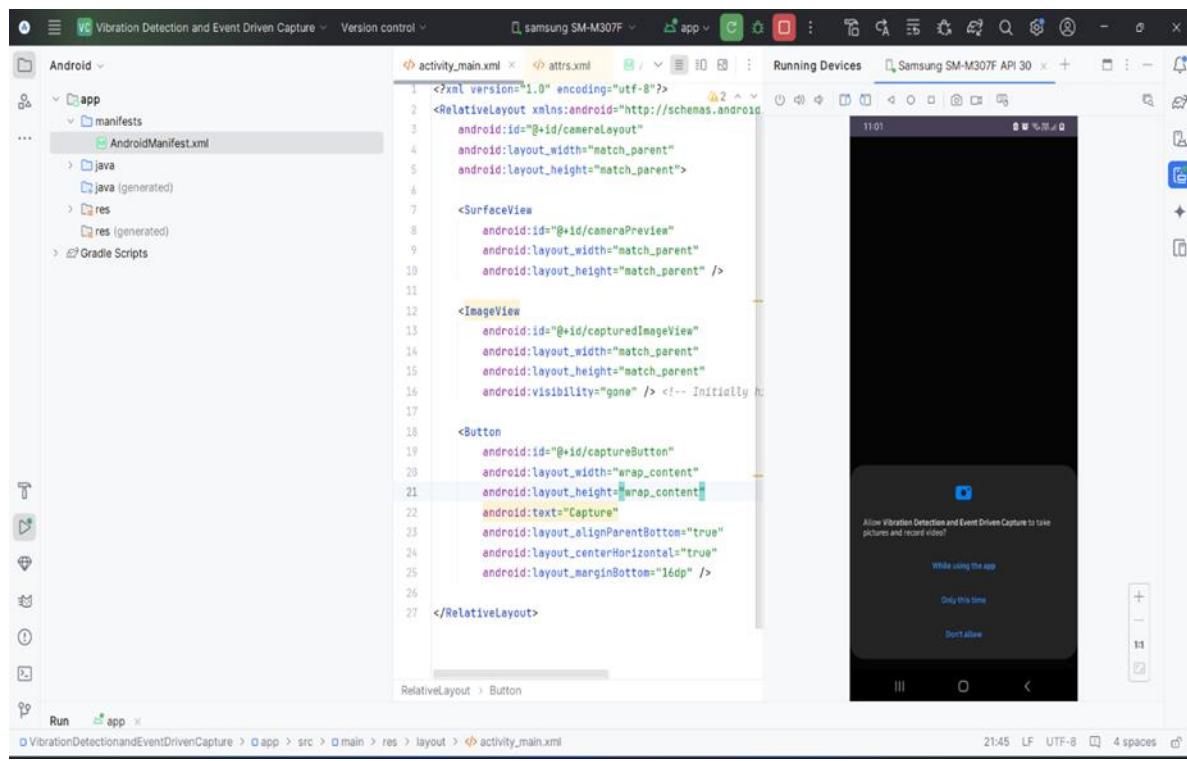
                double acceleration = Math.sqrt(x * x + y * y + z * z);

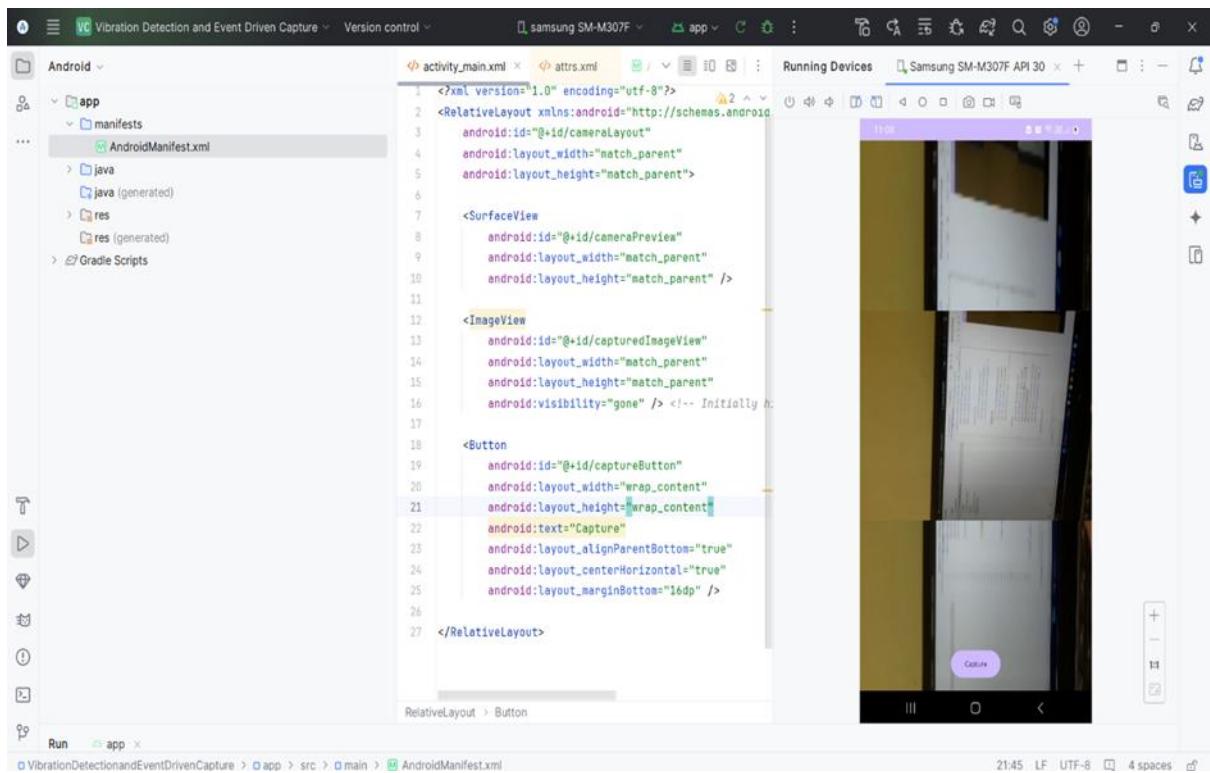
                // Adjust threshold as per your testing and requirements
                if (acceleration > ACCELERATION_THRESHOLD) {
                    Log.d(TAG, msg: "Vibration detected. Starting capture.");
                    handler.post(captureRunnable); // Start capturing every 2 seconds
                }
            }
        }

        @Override 3 usages
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
            // Not used in this example
        }
    }

```

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

```

public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEvent
{
    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(); 4 usages
    private boolean isCapturing = false; 5 usages
    private Runnable captureRunnable = new Runnable() { 3 usages
        @Override
        public void run() {
            if (camera != null && isCapturing) {
                try {
                    camera.takePicture(shutter: null, raw: null, pictureCallback);
                } catch (Exception e) {
                    Log.e(TAG, msg: "Error taking picture: " + e.getMessage());
                    e.printStackTrace();
                }
                handler.postDelayed(@this, delayMillis: 2000); // Capture every 2 seconds
            }
        }
    };
}

```

The screenshot shows the Android Studio interface with the project navigation bar at the top. The left sidebar displays the project structure under the 'Android' tab, showing the app module with its manifest, Java files, and resource folders. The main editor window is focused on the 'MainActivity.java' file, which contains Java code for an AppCompatActivity. The code imports various Android packages and defines a class that implements SurfaceHolder.Callback, Camera.PreviewCallback, and SensorEventListener. The code includes methods for surface creation, destruction, and preview changes, along with a releaseCamera method. The status bar at the bottom shows system information like battery level, signal strength, and the current date and time.

```
1 package com.example.vibrationdetectionandeventdrivencapture;
2 import android.Manifest;
3 import android.content.Context;
4 import android.content.pm.PackageManager;
5 import android.graphics.Bitmap;
6 import android.graphics.BitmapFactory;
7 import android.hardware.Camera;
8 import android.hardware.Sensor;
9 import android.hardware.SensorEvent;
10 import android.hardware.SensorEventListener;
11 import android.hardware.SensorManager;
12 import android.os.Bundle;
13 import android.os.Handler;
14 import android.util.Log;
15 import android.view.SurfaceHolder;
16 import android.view.SurfaceView;
17 import android.view.View;
18 import android.widget.Button;
19 import android.widget.ImageView;
20 import android.widget.Toast;
21
22 import androidx.annotation.NonNull;
23 import androidx.appcompat.app.AppCompatActivity;
24 import androidx.core.app.ActivityCompat;
25 import androidx.core.content.ContextCompat;
26
27 import java.io.IOException;
28 import java.util.List;
29
30 public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEventListener {
31
32     private static final String TAG = "MainActivity"; 0 usages
```

This screenshot continues from the previous one, showing the same Android Studio environment. The project structure and the beginning of the MainActivity.java code are identical. However, the code block has been extended further down, showing additional methods for camera preview setup and destruction, as well as the implementation of the SensorEventListener interface. The code includes try-catch blocks for handling exceptions during camera operations. The status bar at the bottom remains consistent with the first screenshot.

```
135     public void surfaceChanged(@NonNull SurfaceHolder holder, int format, int width, int height) {
136
137         try {
138             camera.stopPreview();
139             previewing = false;
140         } catch (Exception e) {
141             Log.e(TAG, msg: "Error stopping camera preview: " + e.getMessage());
142             e.printStackTrace();
143         }
144
145         try {
146             camera.setPreviewDisplay(surfaceHolder);
147             camera.setPreviewCallback(this);
148             camera.startPreview();
149             previewing = true;
150         } catch (IOException e) {
151             Log.e(TAG, msg: "Error setting camera preview: " + e.getMessage());
152             e.printStackTrace();
153         }
154
155     }
156
157     @Override
158     public void surfaceDestroyed(@NonNull SurfaceHolder holder) {
159         releaseCamera();
160     }
161
162     private void releaseCamera() { 2 usages
163         if (camera != null) {
164             try {
165                 camera.stopPreview();
166                 camera.setPreviewCallback(null);
```

The screenshot shows the Android Studio interface with the project 'Vibration Detection and Event Driven Capture' open. The left sidebar displays the project structure under the 'Android' tab, showing files like AndroidManifest.xml, activity_main.xml, and MainActivity.java. The right pane shows the code editor with the MainActivity.java file open. The code implements a vibration detection logic using SensorEvent values to calculate acceleration and start a capture process if it exceeds a threshold.

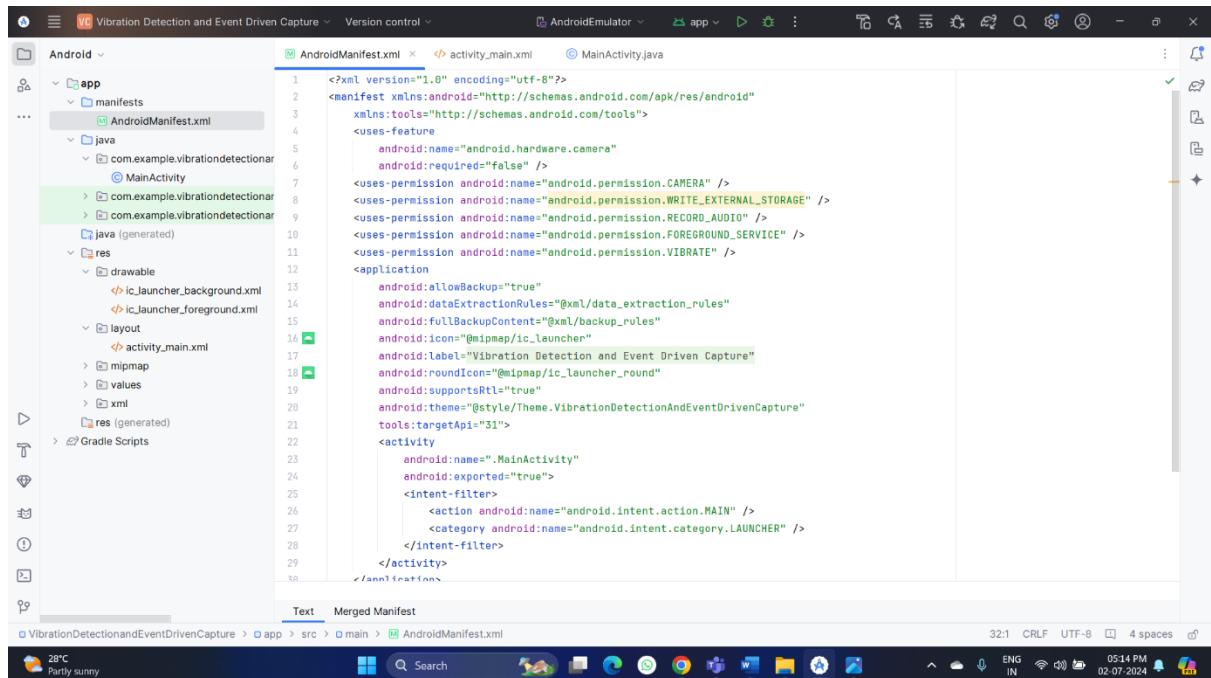
```
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEvent {  
    public void onSensorChanged(SensorEvent event) {  
        float x = event.values[0];  
        float y = event.values[1];  
        float z = event.values[2];  
  
        double acceleration = Math.sqrt(x * x + y * y + z * z);  
  
        // Adjust threshold as per your testing and requirements  
        if (acceleration > ACCELERATION_THRESHOLD) {  
            Log.d(TAG, msg: "Vibration detected. Starting capture.");  
            startCapturing();  
        }  
    }  
    private void startCapturing() {  
        if (!isCapturing) {  
            isCapturing = true;  
            handler.post(captureRunnable);  
        }  
    }  
    private void stopCapturing() {  
        isCapturing = false;  
        handler.removeCallbacks(captureRunnable);  
    }  
    @Override 3 usages  
    public void onAccuracyChanged(Sensor sensor, int accuracy) {  
        // Not used in this example  
    }  
}
```

Update the activity_main.xml

The screenshot shows the Android Studio interface with the project 'Vibration Detection and Event Driven Capture' open. The left sidebar displays the project structure under the 'Android' tab, showing files like AndroidManifest.xml, activity_main.xml, and MainActivity.java. The right pane shows the code editor with the activity_main.xml file open. The XML layout defines a relative layout containing a SurfaceView for camera preview, an ImageView for captured images, and a button labeled 'Capture'.

```
<?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 --&gt;<br/>  
    <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>
```

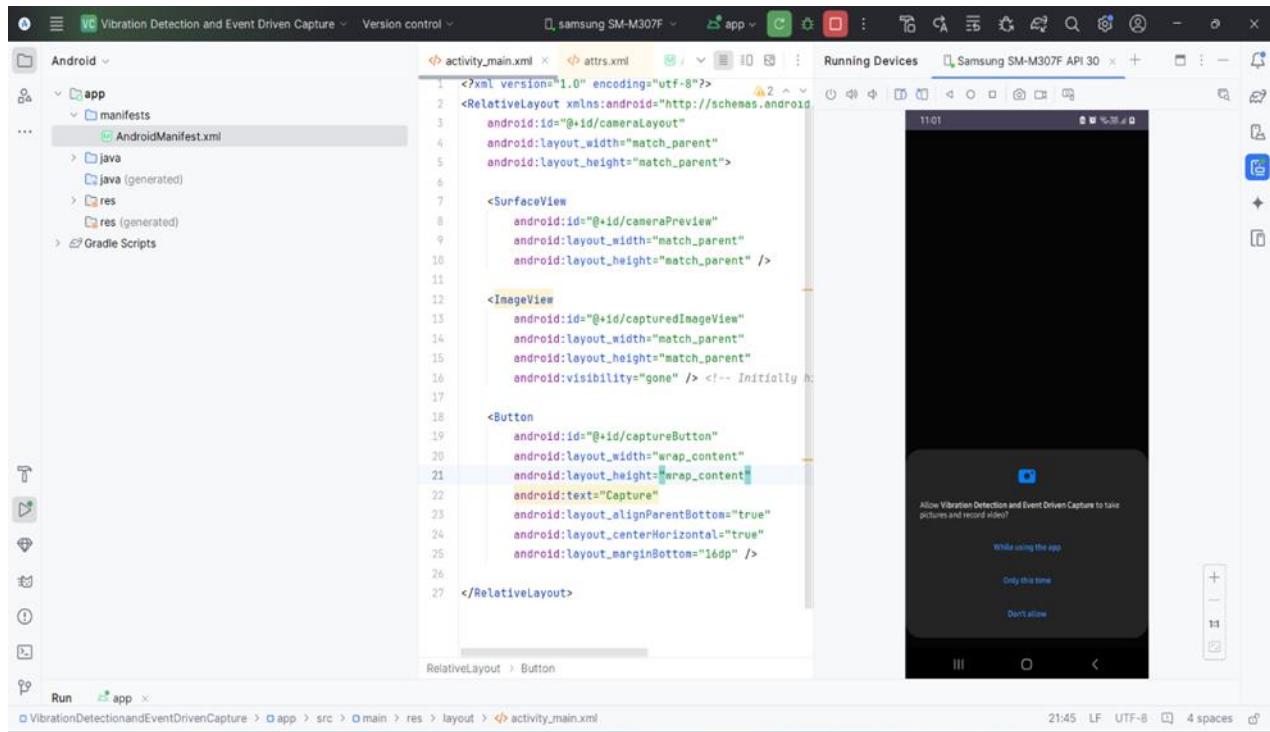
Update the AndroidManifest.xml



The screenshot shows the Android Studio interface with the project 'Vibration Detection and Event Driven Capture' open. The left sidebar shows the project structure with 'AndroidManifest.xml' selected. The main editor window displays the XML code for the manifest. The code includes various permission declarations and activity configurations. The status bar at the bottom shows the date and time as 02-07-2024.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Vibration Detection and Event Driven Capture"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.VibrationDetectionAndEventDrivenCapture"
        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>
```

DAY 2 TASK 2 OUTPUT SCREENSHOTS



The screenshot shows the Android Studio interface with the project 'Vibration Detection and Event Driven Capture' open. The left sidebar shows the project structure with 'activity_main.xml' selected. The main editor window displays the XML code for the layout. The right side shows a preview of the layout on a Samsung SM-M307F device. A permission dialog is visible on the screen, asking for camera access. The status bar at the bottom shows the date and time as 02-07-2024.

```
<?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 -->
    <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>
```

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
    <uses-permission android:name="android.permission.VIBRATE" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Vibration Detection and Event Driven Capture"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.VibrationDetectionAndEventDriver"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
            </intent-filter>
        </activity>
    </application>
</manifest>

```

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

Update the 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.media.MediaScannerConnection;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
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.content.ContextCompat;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

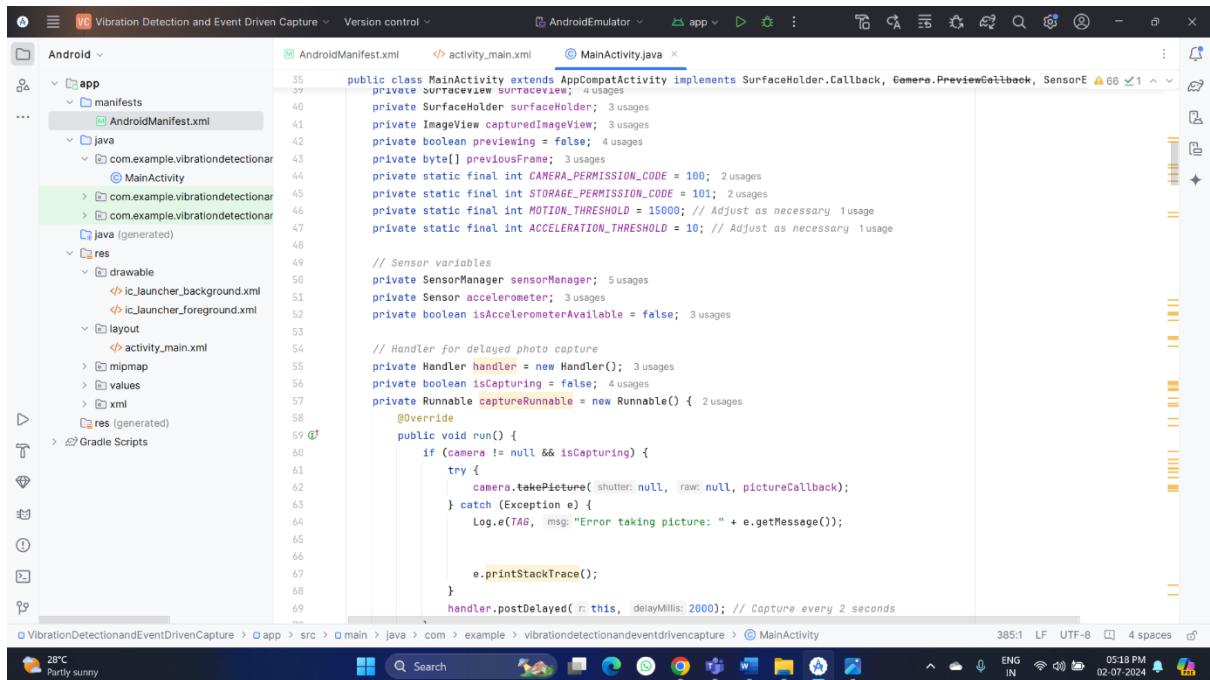
```

This screenshot shows the Android Studio interface with the project 'Vibration Detection and Event Driven Capture' open. The code editor displays the `MainActivity.java` file, which contains Java code for handling camera preview and surface changes. The code includes imports for `AppCompatActivity`, `SurfaceHolder`, `SurfaceHolder.Callback`, `SurfaceHolder.Callback`, `SurfaceHolder`, `SensorEvent`, and `SurfaceHolder.Callback`. It defines a class `MainActivity` that extends `AppCompatActivity` and implements `SurfaceHolder.Callback`. The code handles the creation, change, and destruction of a camera preview surface, setting up the camera's preview display and callback.

```
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEvent.Callback {  
    public void surfaceCreated(@NotNull SurfaceHolder holder) {  
        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 surfaceChanged(@NotNull 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(@NotNull SurfaceHolder holder) {  
            if (previewing) {  
                try {  
                    camera.stopPreview();  
                    previewing = false;  
                } catch (Exception e) {  
                    Log.e(TAG, msg: "Error stopping camera preview: " + e.getMessage());  
                    e.printStackTrace();  
                }  
            }  
        }  
    }  
}
```

This screenshot shows the same Android Studio session after additional code has been added to the `MainActivity.java` file. The new code, located between lines 354 and 581, demonstrates how to save a compressed image from a `MediaScannerConnection`. It uses a `FileOutputStream` to write the image data to a file and then registers a `OnScanCompletedListener` to handle the completion of the scan. The code also includes a `finally` block to ensure the `FileOutputStream` is closed properly.

```
private void saveCompressedImage(Bitmap bitmap) {  
    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  
}
```



```
public class MainActivity extends AppCompatActivity implements SurfaceHolder.Callback, Camera.PreviewCallback, SensorEvent
```

```
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 STORAGE_PERMISSION_CODE = 101; 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 boolean isCapturing = false; 4 usages
```

```
private Runnable captureRunnable = new Runnable() { 2 usages
```

```
    @Override
```

```
    public void run() {
```

```
        if (camera != null && isCapturing) {
```

```
            try {
```

```
                camera.takePicture(shutter: null, raw: null, pictureCallback);
```

```
            } catch (Exception e) {
```

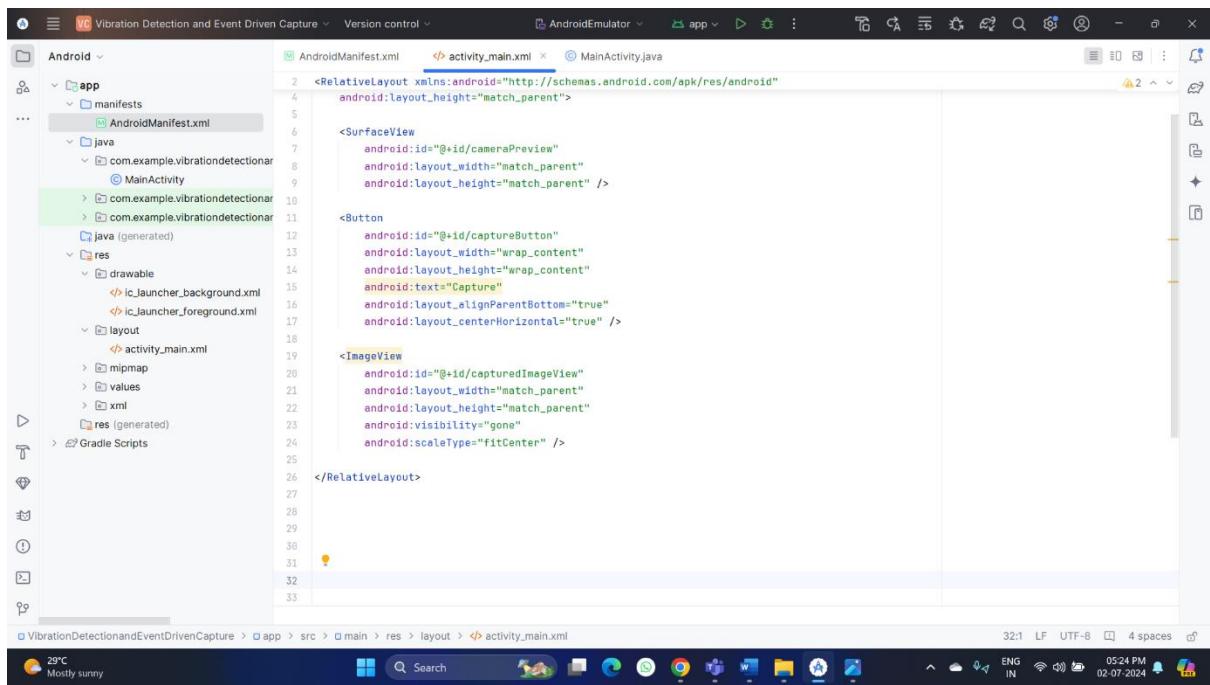
```
                Log.e(TAG, msg: "Error taking picture: " + e.getMessage());
```

```
                e.printStackTrace();
```

```
            }
```

```
            handler.postDelayed(r: this, delayMillis: 2000); // Capture every 2 seconds
```

Update the activity_main.xml



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

```
    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"
```

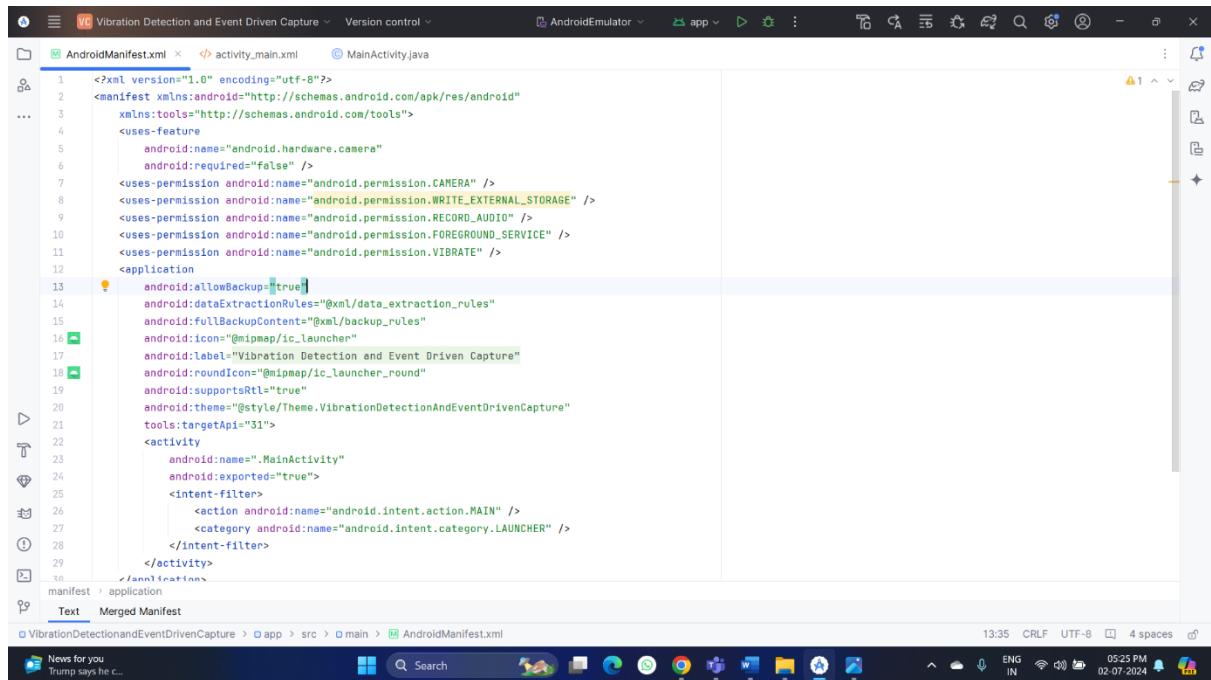
```
        android:layout_height="match_parent"
```

```
        android:visibility="gone"
```

```
        android:scaleType="fitCenter" />
```

```
</RelativeLayout>
```

Update the AndroidManifest.xml



The screenshot shows the Android Studio interface with the project 'Vibration Detection and Event Driven Capture' open. The code editor displays the `AndroidManifest.xml` file, which has been updated to include the following permissions:

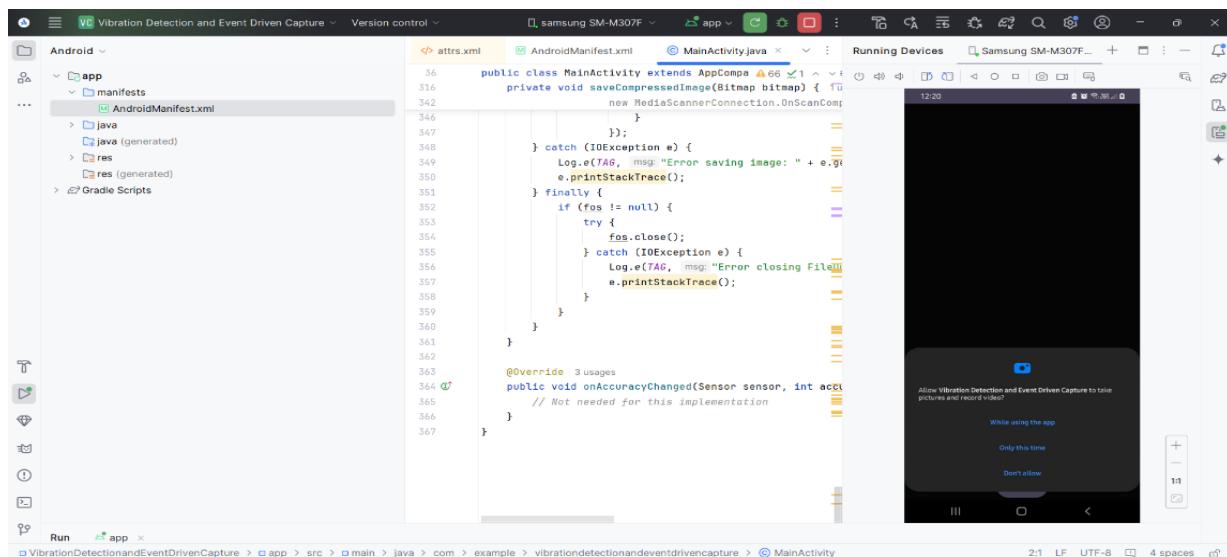
```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.VIBRATE" />
```

The manifest also includes the following application settings:

```
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Vibration Detection and Event Driven Capture"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.VibrationDetectionAndEventDrivenCapture"
        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" />
            
        
    
```

The status bar at the bottom shows the time as 13:35, battery level at 05:25 PM, and date as 02-07-2024.

DAY 2 TASK 3 OUTPUT SCREENSHOTS



The screenshot shows the Android Studio interface with the project 'Vibration Detection and Event Driven Capture' open. The code editor displays the `MainActivity.java` file, which contains the following code snippet:

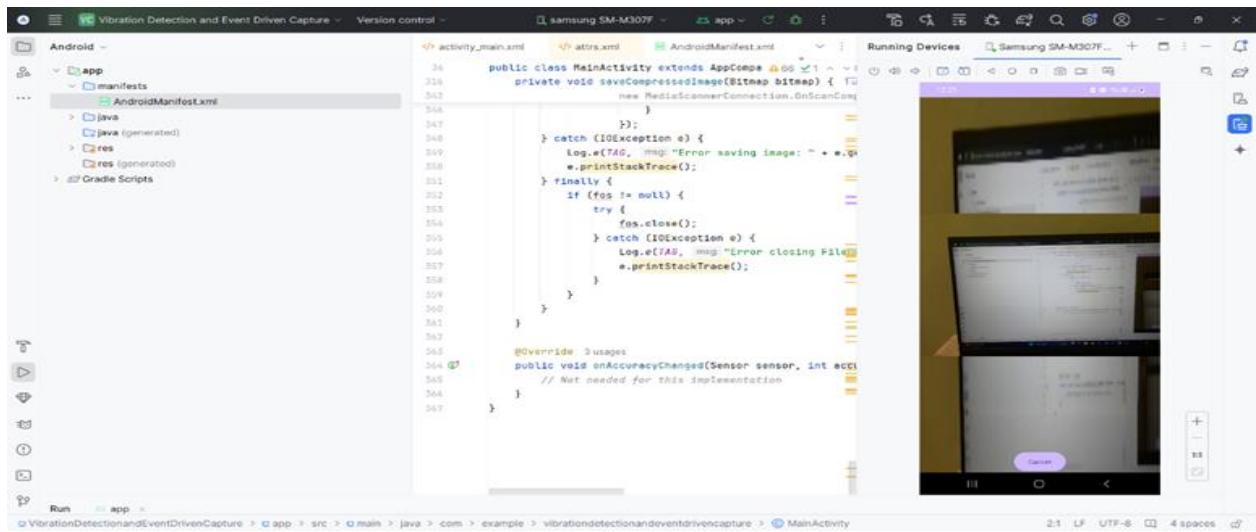
```
private void saveCompressedImage(Bitmap bitmap) {
    try {
        fos = new FileOutputStream(new MediaScannerConnection.OnScanCompleted
            .Listener() {
                public void onMediaScanned(String path, Uri uri) {
                    Log.d(TAG, "Image saved to " + path);
                }
            }.getOutputFile());
        fos.write(bitmap.compress(Bitmap.CompressFormat.JPEG, 100, null));
    } catch (IOException e) {
        Log.e(TAG, "Error saving image: " + e.getMessage());
        e.printStackTrace();
    } finally {
        if (fos != null) {
            try {
                fos.close();
            } catch (IOException e) {
                Log.e(TAG, "Error closing File");
                e.printStackTrace();
            }
        }
    }
}
```

To the right of the code editor, a screenshot of the Samsung SM-M307F smartphone emulator is displayed, showing a permission dialog:

Allow Vibration Detection and Event Driven Capture to take pictures and record video?

While using the app
Only this time
Don't allow

The status bar at the bottom shows the time as 12:20, battery level at 2:1, and date as 02-07-2024.



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

Update the MainActivity.java

```

1 package com.example.vibrationdetectionandeventdrivencapture;
2 import android.Manifest;
3 import android.content.Context;
4 import android.content.pm.PackageManager;
5 import android.graphics.Bitmap;
6 import android.graphics.BitmapFactory;
7 import android.hardware.Camera;
8 import android.hardware.Sensor;
9 import android.hardware.SensorEvent;
10 import android.hardware.SensorEventListener;
11 import android.hardware.SensorManager;
12 import android.media.MediaScannerConnection;
13 import android.net.Uri;
14 import android.os.Bundle;
15 import android.os.Environment;
16 import android.os.Handler;
17 import android.util.Log;
18 import android.view.SurfaceHolder;
19 import android.view.SurfaceView;
20 import android.view.View;
21 import android.widget.Button;
22 import android.widget.ImageView;
23 import android.widget.Toast;
24 import androidx.annotation.NonNull;
25 import androidx.appcompat.app.AppCompatActivity;
26 import androidx.core.app.ActivityCompat;
27 import androidx.core.content.ContextCompat;
28 import java.io.ByteArrayOutputStream;
29 import java.io.File;
30 import java.io.FileOutputStream;
31 import java.io.IOException;
32 import java.util.List;

```

AndroidManifest.xml activity_main.xml 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, Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(activity, new String[]{Manifest.permission.CAMERA}, CAMERA_PERMISSION_CODE);
    } else {
        openCamera();
    }
    if (ContextCompat.checkSelfPermission(context, Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(activity, 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, SensorManager.SENSOR_DELAY_NORMAL);
    }
}
@Override
protected void onPause() {
    super.onPause();
    // Unregister accelerometer sensor listener and release camera
    if (isAccelerometerAvailable) {
        sensorManager.unregisterListener(listener);
        releaseCamera();
    }
}

```

VibrationDetectionandEventDrivenCapture > app > src > main > java > com > example > vibrationdetectionandeventdrivencapture > MainActivity

AndroidManifest.xml activity_main.xml MainActivity.java

```
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, text: "Camera permission required", Toast.LENGTH_SHORT).show();
            }
        } else if (requestCode == STORAGE_PERMISSION_CODE) {
    }
}
```

VibrationDetectionandEventDrivenCapture > app > src > main > java > com > example > vibrationdetectionandeventdrivencapture > MainActivity

The screenshot shows the Android Studio interface with the project 'Vibration Detection and Event Driven Capture' open. The code editor displays `MainActivity.java`. The code implements a `SurfaceHolder.Callback`, `Camera.PreviewCallback`, and `SensorEventListener`. It includes logic for compressing a `Bitmap` to JPEG format and saving it to a file using a `FileOutputStream`. It also uses a `MediaScannerConnection` to scan the saved image. Error handling is provided for `IOException` and `FileNotFoundException`.

```
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, 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

The screenshot shows the Android Studio interface with the project 'Vibration Detection and Event Driven Capture' open. The code editor displays `activity_main.xml`. The XML layout defines a `RelativeLayout` containing a `SurfaceView` for camera preview, a `Button` for capturing images, and an `ImageView` for displaying captured images.

```
<?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

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Vibration Detection and Event Driven Capture"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.VibrationDetectionAndEventDrivenCapture"
        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>
```

DAY 2 TASK 4 OUTPUT SCREENSHOTS

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



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

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <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:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.PowerManagementAndBackgroundServices"
        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 android:name=".Powermanagement_Activity"
            android:exported="false" />
        <activity android:name=".BackgroundService_Activity"
            android:exported="false" />
        <service android:name=".MyService"
            android:exported="false" />
        <service android:name=".PowerManagementService"
            android:exported="false" />
    </application>

```

Powermanage Activity.java

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar displays the project structure under "Android". It includes the "app" module with "manifests" (AndroidManifest.xml), "java" (com.example.powermanagementandbackgroundservices containing BackgroundService_Activity, MainActivity, MyService, Powermanage_Activity, and PowerManagementService), "res" (drawable, layout, mipmap, values, and xml), and "Gradle Scripts".
- Code Editor:** The main window shows the code for "Powermanage_Activity.java". The code defines a class that extends AppCompatActivity and implements ServiceConnection. It includes methods for onServiceConnected and onServiceDisconnected.
- Toolbars and Status Bar:** The top bar shows tabs for "Power Management and Background Services" and "Version control". The bottom status bar shows the date (02-07-2024), time (20:37 PM), battery level (26°C), signal strength, and connectivity icons.

```
fest.xml
13     @BackgroundService_Activity.java
14
15     public class Powermanagement_Activity extends AppCompatActivity {
16         protected void onCreate(Bundle savedInstanceState) {
17             super.onCreate(savedInstanceState);
18
19             startServiceButton.setOnClickListener(new View.OnClickListener() {
20                 @Override
21                 public void onClick(View v) { startPowerManagementService(); }
22             });
23
24             stopServiceButton.setOnClickListener(new View.OnClickListener() {
25                 @Override
26                 public void onClick(View v) { stopPowerManagementService(); }
27             });
28
29             navigateToBackgroundServiceButton.setOnClickListener(new View.OnClickListener() {
30                 @Override
31                 public void onClick(View v) {
32                     Intent intent = new Intent(getApplicationContext(), Powermanagement_Activity.this, BackgroundService_Activity.class);
33                     startActivity(intent);
34                 }
35             });
36
37             private void startPowerManagementService() {
38                 Intent serviceIntent = new Intent(getApplicationContext(), PowerManagementService.class);
39                 startService(serviceIntent);
40                 bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
41                 statusTextView.setText("Power Management Active\nBattery usage minimized by turning off non-essential services");
42                 Toast.makeText(getApplicationContext(), "Power Management Service started", Toast.LENGTH_SHORT).show();
43             }
44
45             private void stopPowerManagementService() {
46                 if (serviceIsBound) {
47                     if (serviceConnection != null) {
48                         unbindService(serviceConnection);
49                         serviceConnection = null;
50                     }
51                 }
52             }
53
54         }
55
56         @Override
57         public void onBackPressed() {
58             finish();
59         }
60     }
61
62     </activity>
63
64     <activity android:name=".MainActivity" />
65
66     <activity android:name=".MyService" />
67
68     <activity android:name=".Powermanagement_Activity" />
69
70     <activity android:name=".PowerManagementService" />
71
72     <activity android:name=".BackgroundService_Activity" />
73
74     <activity android:name=".Powermanagement_Activity" />
75
76     <activity android:name=".Powermanagement_Activity" />
77
78     <activity android:name=".Powermanagement_Activity" />
```

PowerManagementActivity.java

```

13 public class Powermanagement_Activity extends AppCompatActivity {
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17     }
18
19     @Override
20     public void onClick(View v) {
21         Intent intent = new Intent(getApplicationContext(), PowerManagementService.class);
22         startService(intent);
23     }
24
25     private void startPowerManagementService() {
26         Intent serviceIntent = new Intent(getApplicationContext(), PowerManagementService.class);
27         startService(serviceIntent);
28         bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
29         statusTextView.setText("Power Management Active\nBattery usage minimized by turning off non-essential services");
30         Toast.makeText(getApplicationContext(), "Power Management Service started", Toast.LENGTH_SHORT).show();
31     }
32
33     private void stopPowerManagementService() {
34         if (isServiceBound) {
35             unbindService(serviceConnection);
36             isServiceBound = false;
37         }
38         Intent serviceIntent = new Intent(getApplicationContext(), PowerManagementService.class);
39         stopService(serviceIntent);
40         statusTextView.setText("Power Management Service stopped");
41         Toast.makeText(getApplicationContext(), "Power Management Service stopped", Toast.LENGTH_SHORT).show();
42     }
43
44 }

```

PowerManagementService.java

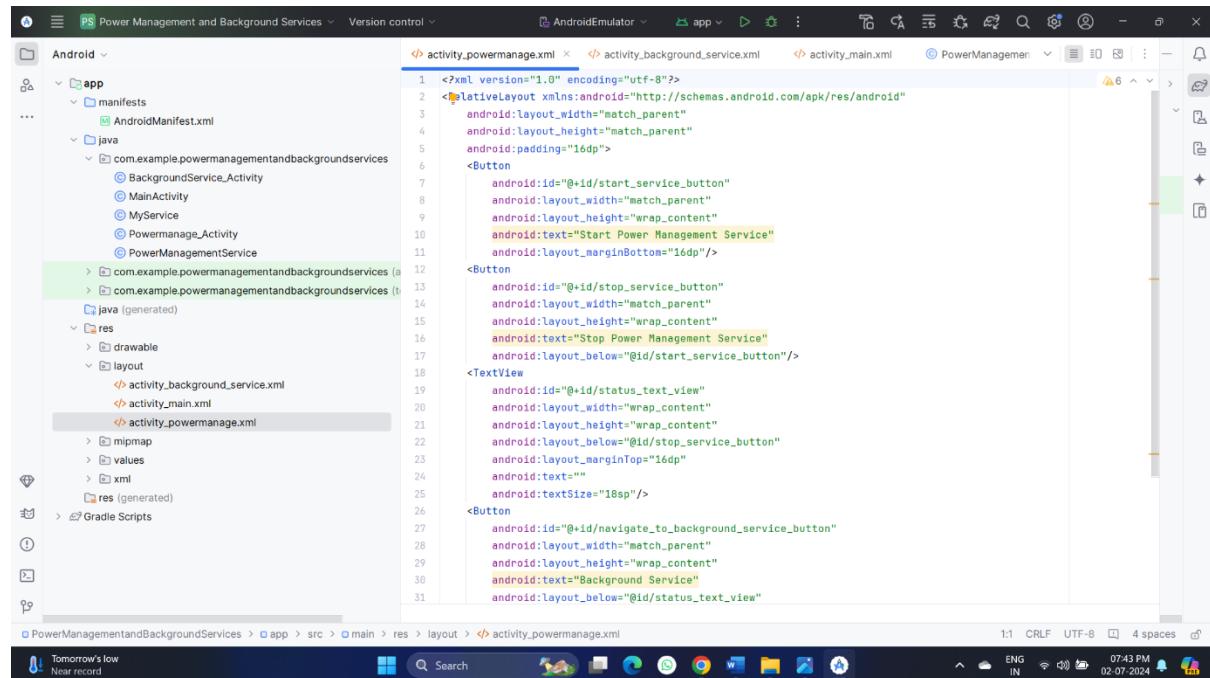
PowerManagementService.java

```

1 package com.example.powermanagementandbackgroundservices;
2 import android.app.Service;
3 import android.content.Intent;
4 import android.os.Binder;
5 import android.os.IBinder;
6
7 public class PowerManagementService extends Service {
8     private final IBinder binder = new LocalBinder();
9     private boolean isRunning = false;
10    public class LocalBinder extends Binder {
11        PowerManagementService getService() { return PowerManagementService.this; }
12    }
13    @Override
14    public IBinder onBind(Intent intent) { return binder; }
15    @Override
16    public int onStartCommand(Intent intent, int flags, int startId) {
17        isRunning = true;
18        // Add your power management logic here
19        return START_STICKY;
20    }
21    @Override
22    public void onDestroy() {
23        super.onDestroy();
24        isRunning = false;
25        // Cleanup logic if needed
26    }
27    public boolean isRunning() { return isRunning; }
28}

```

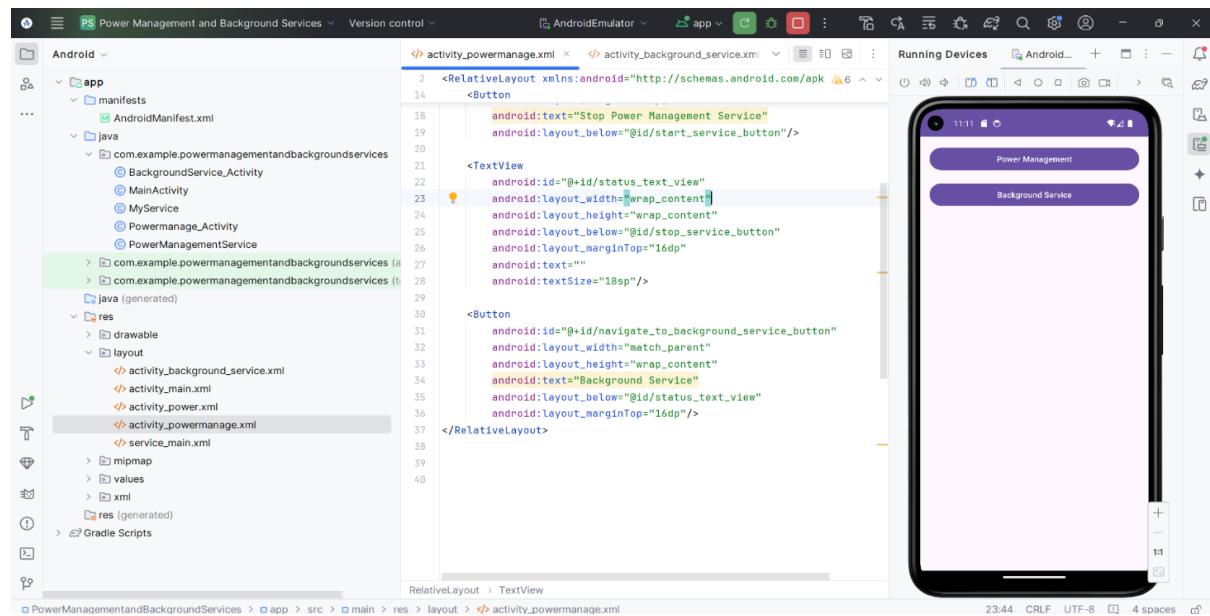
activity_powermanage.xml

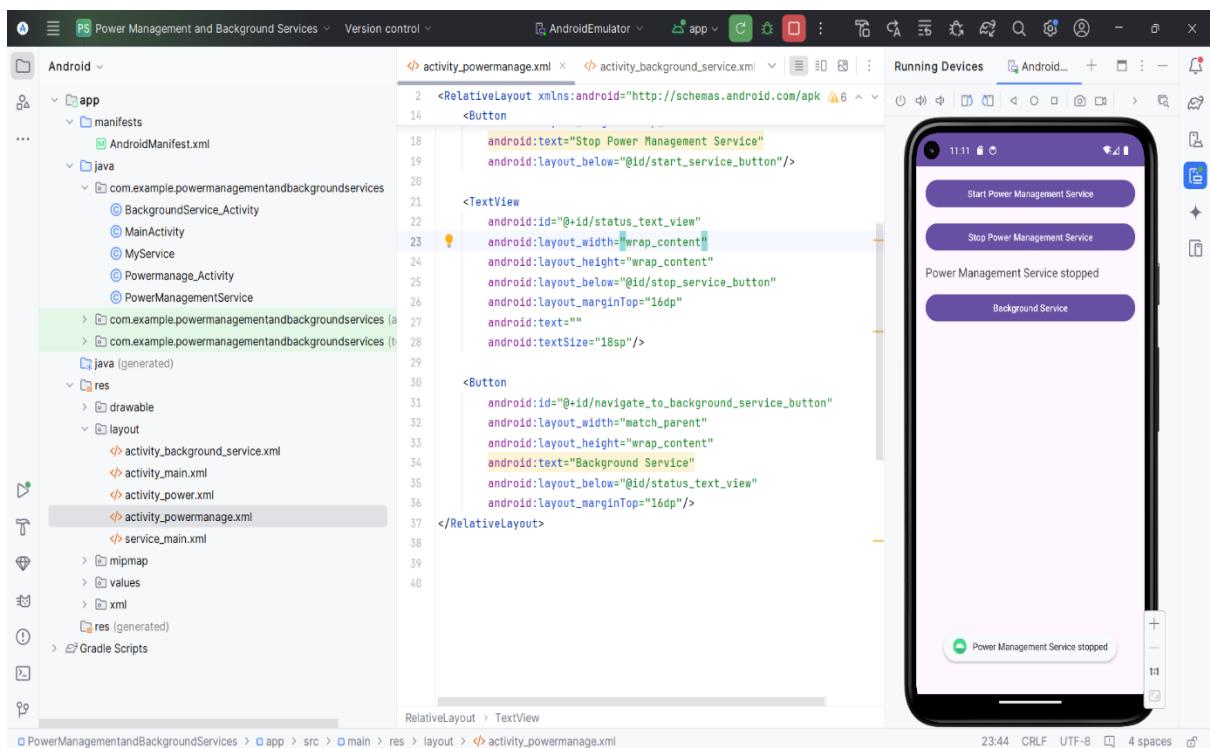
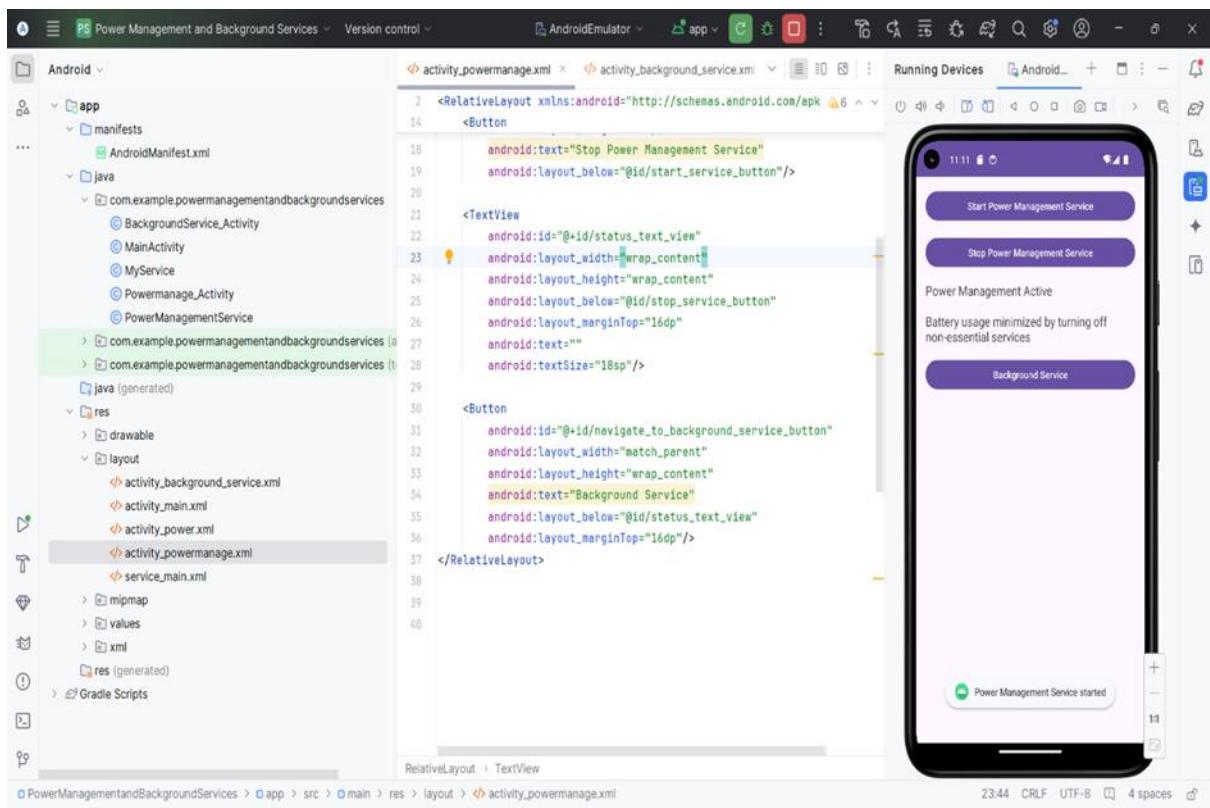


The screenshot shows the Android Studio interface with the project navigation bar at the top. The left sidebar displays the project structure under 'Android'. The main area shows the XML code for 'activity_powermanage.xml'. The code defines a relative layout with two buttons for starting and stopping a service, and a text view for displaying status information.

```
<?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/start_service_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start Power Management Service"
        android:layout_marginBottom="16dp"/>
    <Button
        android:id="@+id/stop_service_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        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>
```

DAY 3 TASK 1 OUTPUT SCREENSHOTS

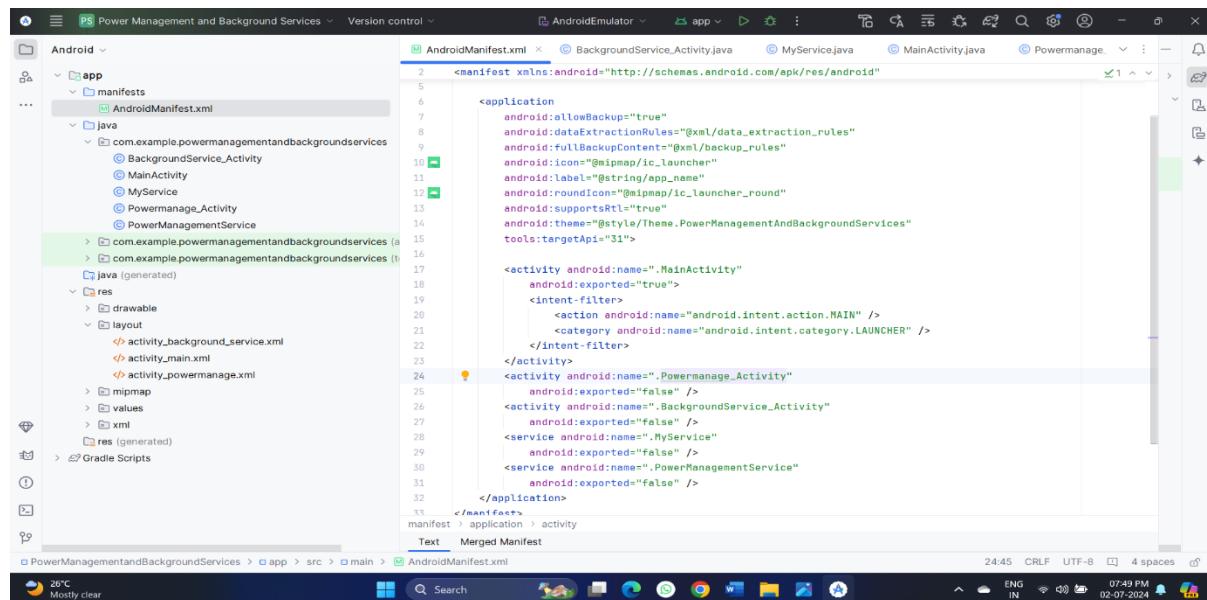




Task 2: Implement a background service that runs the surveillance system only when the vehicle is locked and the engine is off.

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.

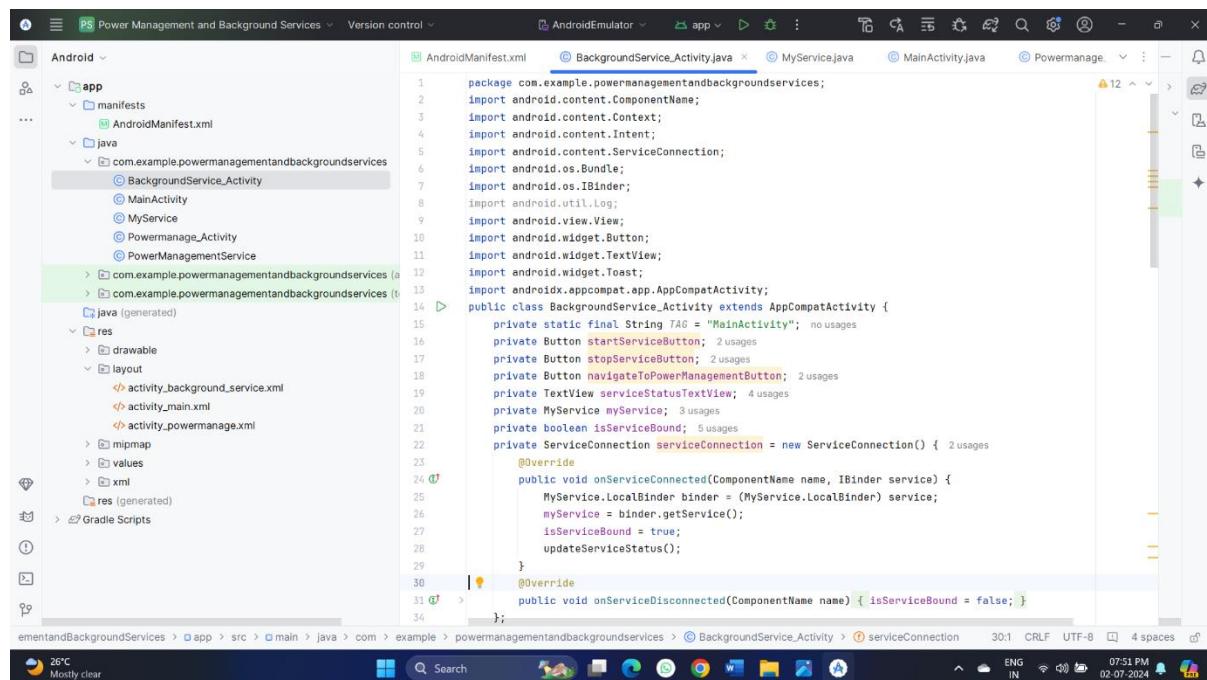
AndroidManifest.xml



The screenshot shows the AndroidManifest.xml file in the Android Studio code editor. The manifest defines an application with three activities: MainActivity, BackgroundService_Activity, and Powermanage_Activity. It also defines two services: MyService and PowerManagementService. The manifest includes various attributes like theme, label, and icons.

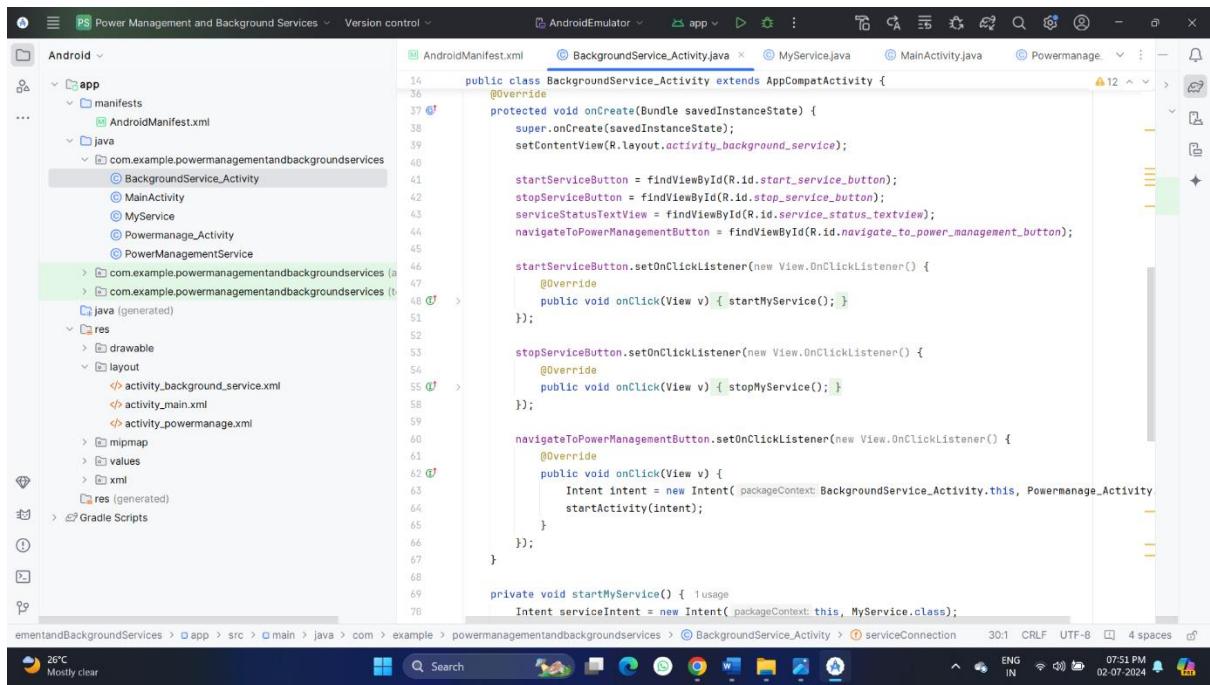
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.PowerManagementAndBackgroundServices"
        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>
        <activity android:name=".Powermanage_Activity"
            android:exported="false" />
        <activity android:name=".BackgroundService_Activity"
            android:exported="false" />
        <service android:name=".MyService"
            android:exported="false" />
        <service android:name=".PowerManagementService"
            android:exported="false" />
    </application>
</manifest>
```

BackgroundService_Activity.java



The screenshot shows the BackgroundService_Activity.java file in the Android Studio code editor. The class extends AppCompatActivity and implements ServiceConnection. It contains methods for starting and stopping the service, as well as handling service connection changes.

```
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.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
public class BackgroundService_Activity extends AppCompatActivity {
    private static final String TAG = "MainActivity"; no usages
    private Button startServiceButton; 2 usages
    private Button stopServiceButton; 2 usages
    private Button navigateToPowerManagementButton; 2 usages
    private TextView serviceStatusTextView; 4 usages
    private MyService myService; 3 usages
    private boolean isServiceBound; 5 usages
    private ServiceConnection serviceConnection = new ServiceConnection() { 2 usages
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            MyService.LocalBinder binder = (MyService.LocalBinder) service;
            myService = binder.getService();
            isServiceBound = true;
            updateServiceStatus();
        }
        @Override
        public void onServiceDisconnected(ComponentName name) { isServiceBound = false; }
    };
}
```



```
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(); }

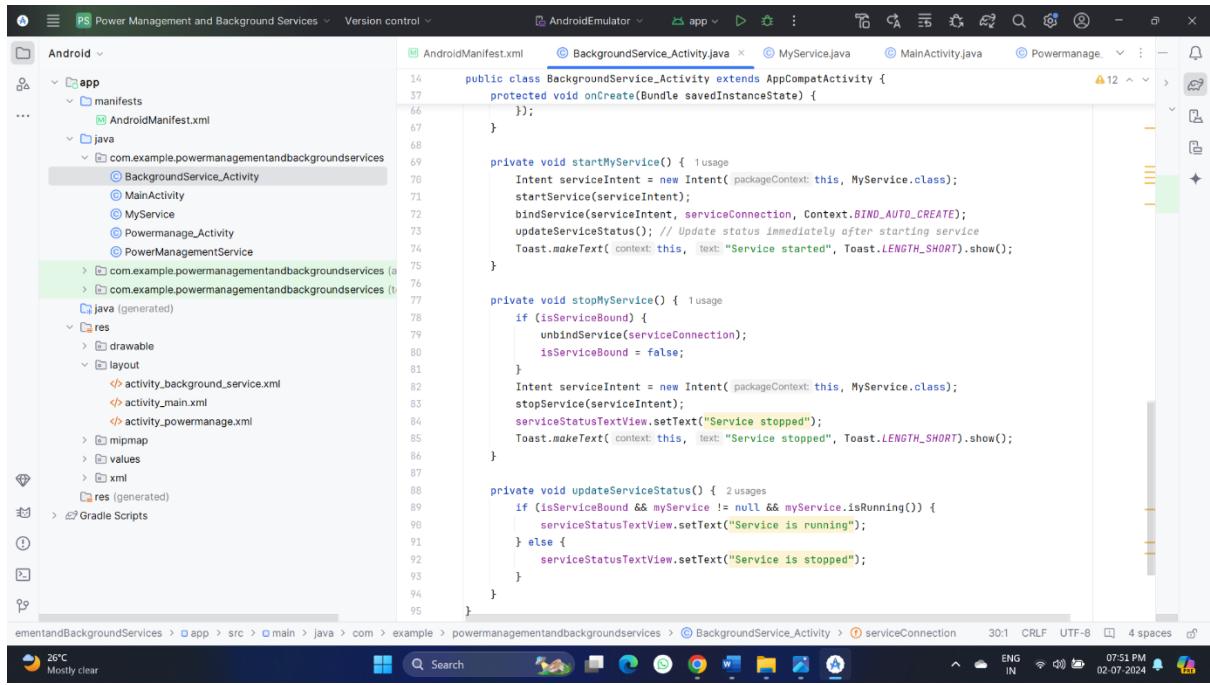
        });

        navigateToPowerManagementButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(), Powermanagement_Activity.class);
                startActivity(intent);
            }
        });
    }

    private void startMyService() { 1 usage
        Intent serviceIntent = new Intent(getApplicationContext(), MyService.class);
        startService(serviceIntent);
        bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
        updateServiceStatus(); // Update status immediately after starting service
        Toast.makeText(getApplicationContext(), "Service started", Toast.LENGTH_SHORT).show();
    }

    private void stopMyService() { 1 usage
        if (isServiceBound) {
            unbindService(serviceConnection);
            isServiceBound = false;
        }
        Intent serviceIntent = new Intent(getApplicationContext(), MyService.class);
        stopService(serviceIntent);
        serviceStatusTextView.setText("Service stopped");
        Toast.makeText(getApplicationContext(), "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");
        }
    }
}
```



```
public class BackgroundService_Activity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_background_service);

        private void startMyService() { 1 usage
            Intent serviceIntent = new Intent(getApplicationContext(), MyService.class);
            startService(serviceIntent);
            bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
            updateServiceStatus(); // Update status immediately after starting service
            Toast.makeText(getApplicationContext(), "Service started", Toast.LENGTH_SHORT).show();
        }

        private void stopMyService() { 1 usage
            if (isServiceBound) {
                unbindService(serviceConnection);
                isServiceBound = false;
            }
            Intent serviceIntent = new Intent(getApplicationContext(), MyService.class);
            stopService(serviceIntent);
            serviceStatusTextView.setText("Service stopped");
            Toast.makeText(getApplicationContext(), "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");
            }
        }
}
```

MyService.java

The screenshot shows the Android Studio interface with the code editor open to the `MyService.java` file. The code defines a service that starts when the application starts and can be stopped via a command. It includes logging for service creation and start.

```
1 package com.example.powermanagementandbackgroundservices;
2 import android.app.Service;
3 import android.content.Intent;
4 import android.os.Binder;
5 import android.os.IBinder;
6 import android.util.Log;
7 public class MyService extends Service {
8     private static final String TAG = "MyService";
9     private final IBinder binder = new LocalBinder();
10    private boolean isRunning = false;
11    public class LocalBinder extends Binder {
12        MyService getService() { return MyService.this; }
13    }
14    @Override
15    public void onCreate() {
16        super.onCreate();
17        Log.d(TAG, msg("Service created"));
18        isRunning = true;
19    }
20    @Override
21    public int onStartCommand(Intent intent, int flags, int startId) {
22        Log.d(TAG, msg("Service started"));
23        // Your service logic here
24        return START_STICKY;
25    }
26    @Override
27    public void onDestroy() {
28        super.onDestroy();
29        Log.d(TAG, msg("Service destroyed"));
30        isRunning = false;
31    }
32    @Override
33    public void onHandleIntent(Intent intent) {
34        Log.d(TAG, msg("Handling intent"));
35    }
36}
```

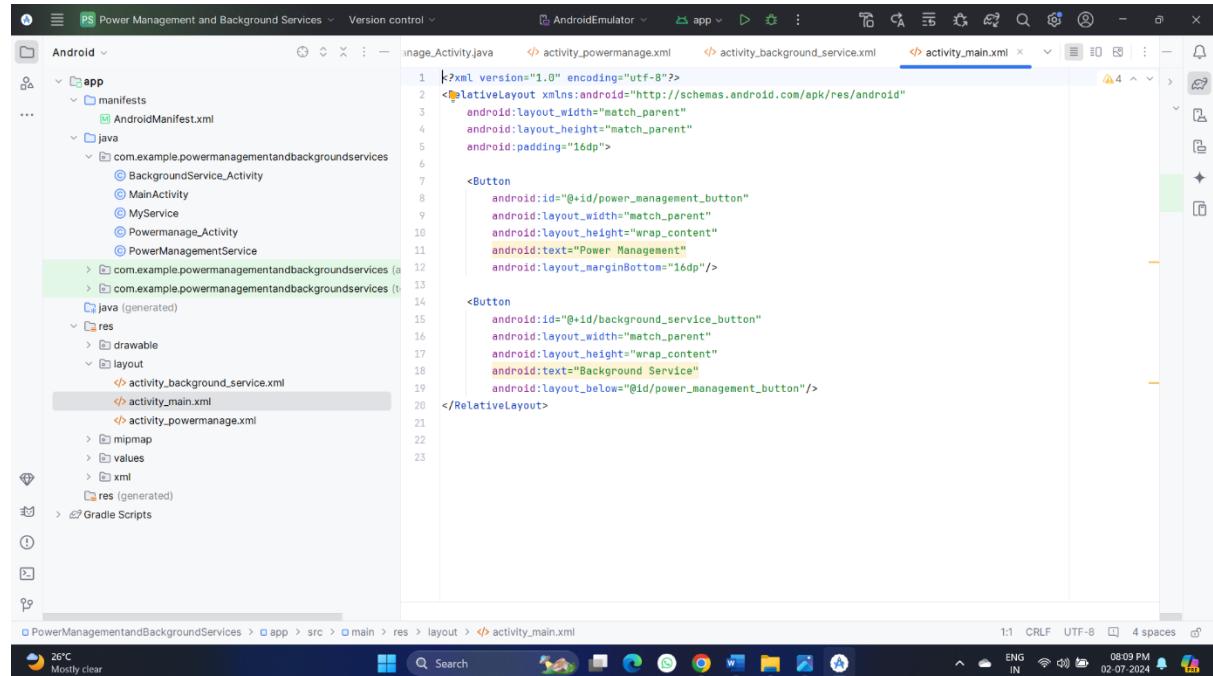
activity_background_service.xml

The screenshot shows the Android Studio interface with the code editor open to the `activity_background_service.xml` file. It defines a layout with two buttons for starting and stopping the service, and a text view for displaying service status.

```
<?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/start_service_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start Service"
        android:layout_marginBottom="16dp"/>
    <Button
        android:id="@+id/stop_service_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Stop Service"
        android:layout_below="@+id/start_service_button"
        android:layout_marginTop="16dp"/>
    <TextView
        android:id="@+id/service_status_textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Service status"
        android:layout_below="@+id/stop_service_button"
        android:layout_marginTop="16dp"/>
    <Button
        android:id="@+id/navigate_to_power_management_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Go to Power Management"
        android:layout_below="@+id/service_status_textview"
        android:layout_marginTop="16dp"/>

```

activity_main.xml



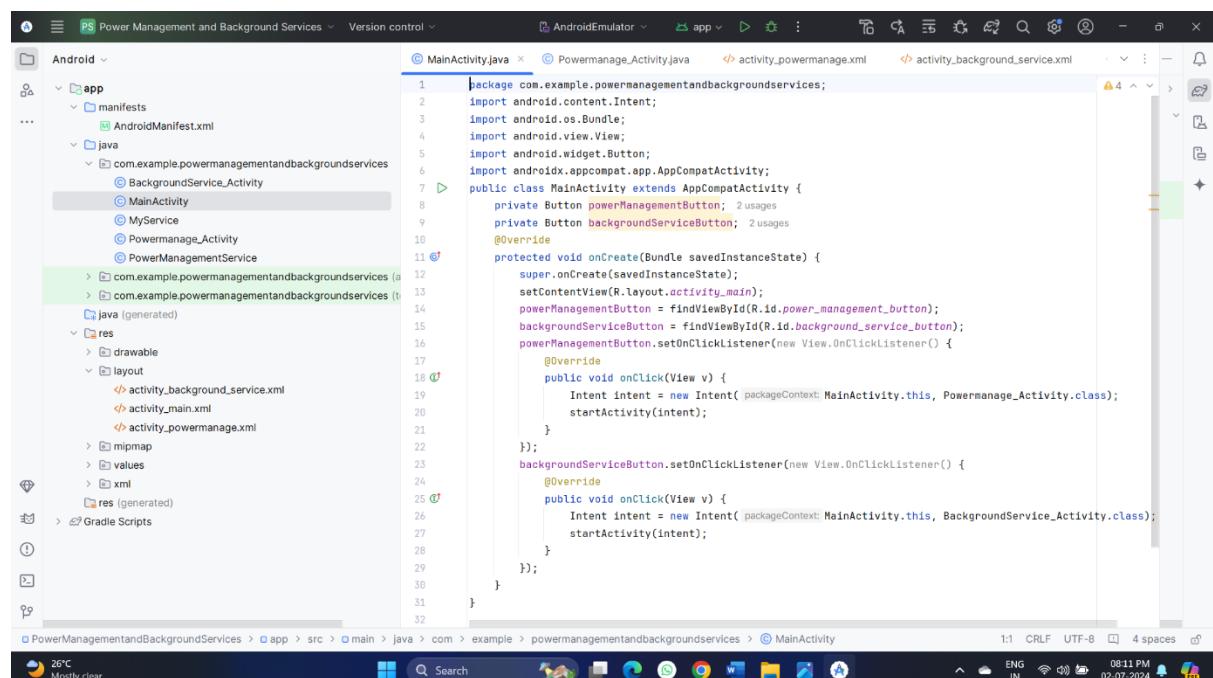
The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is the file tree for the 'app' module, showing files like AndroidManifest.xml, MainActivity.java, and various XML layout files. The main code editor window displays the XML code for 'activity_main.xml'. The code defines a relative layout containing two buttons. The first button is for power management and the second is for background service control.

```
<?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>
```

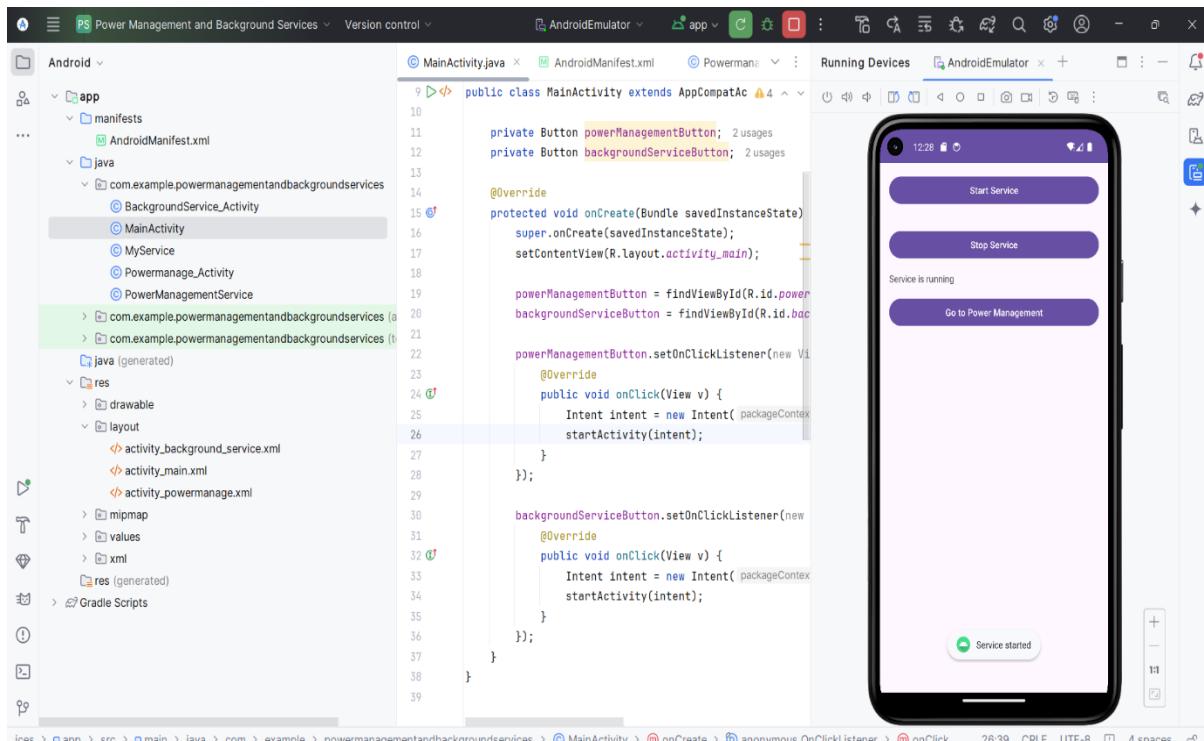
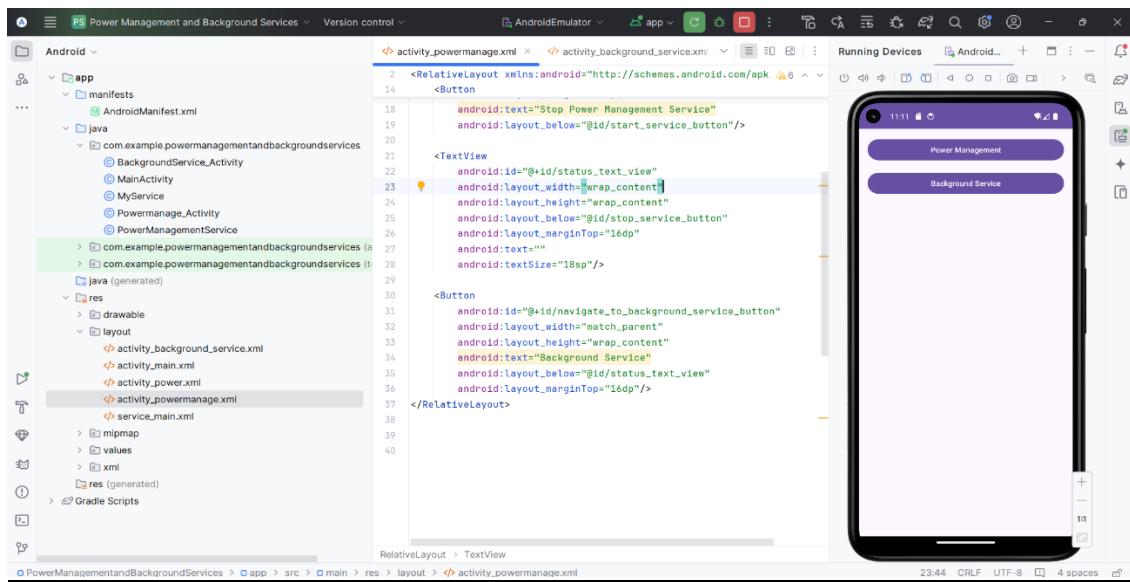
MainActivity.java

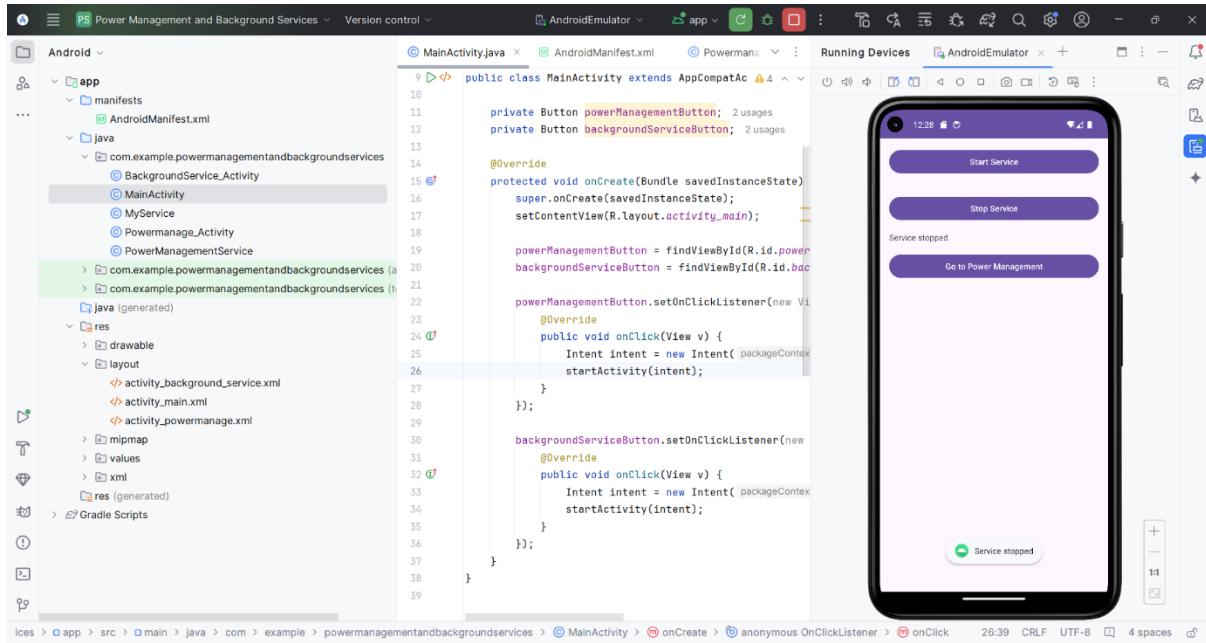


The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is the file tree for the 'app' module, showing files like AndroidManifest.xml, MainActivity.java, and various XML layout files. The main code editor window displays the Java code for 'MainActivity'. It extends AppCompatActivity and contains logic for handling button clicks to start different services.

```
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(getApplicationContext(), Powermanage_Activity.class);
                startActivity(intent);
            }
        });
        backgroundServiceButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(), 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

The screenshot shows the Android Studio interface with the file `MainActivity.java` open in the editor. The code implements an `AppCompatActivity` with methods for `onCreate` and button click listeners. The `onCreate` method initializes views and sets up button click listeners. The `onClick` methods handle media display and capture.

```
public class MainActivity extends AppCompatActivity {
    private TextView textViewMediaDisplay;
    private Uri capturedImageUri;
    private ImageView imageViewMedia;
    private Button buttonViewMedia;
    private Button buttonCaptureImage;
    private TextView textViewAlerts;
    private ListView listViewAlerts;

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

        // Initialize views
        textViewMediaDisplay = findViewById(R.id.text_media_display);
        imageViewMedia = findViewById(R.id.image_media);
        buttonViewMedia = findViewById(R.id.button_view_media);
        buttonCaptureImage = findViewById(R.id.button_capture_image);
        textViewAlerts = findViewById(R.id.text_alerts);
        listViewAlerts = 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) {
                ...
            }
        });
    }

    private void showCapturedMediaAndAlert(Uri uri) {
        ...
    }

    private void showToast(String message) {
        ...
    }
}
```

The project structure on the left includes `AndroidManifest.xml`, `build.gradle.kts`, `activity_main.xml`, `item_alert.xml`, `AlertsAdapter.java`, and `AndroidManifest.xml`. The bottom status bar shows weather information (28°C, Mostly cloudy), system icons, and the date/time (03-07-2024).

This screenshot shows the same `MainActivity.java` file with additional imports at the top. The imports include various Android components like `Intent`, `PackageManager`, `Bitmap`, `Uri`, `Bundle`, `MediaStore`, `View`, `Button`, `ImageView`, `ListView`, `TextView`, `Toast`, `NonNull`, `AppCompatActivity`, `ActivityCompat`, `ContextCompat`, `Glide`, `ArrayList`, and `List`.

```
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;
```

The project structure and bottom status bar are identical to the first screenshot.

```

public class MainActivity extends AppCompatActivity {
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == REQUEST_CODE_IMAGE_CAPTURE) {
            if (resultCode == RESULT_OK) {
                Uri imageUri = data.getData();
                // Implement logic to save bitmap to a file and return its URI
                // Placeholder implementation
                String path = MediaStore.Images.Media.insertImage(getApplicationContext(), bitmap, "CapturedImage", null);
                return Uri.parse(path);
            }
        }
    }

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

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

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

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

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

```

The screenshot shows the Android Studio interface with the code editor open to `MainActivity.java`. The code implements a camera functionality, saving a captured image to the device's storage and displaying it in a `Glide`-based image view. It also updates a `ListView` with alert data and shows a toast message.

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> alertDialog;

    public AlertsAdapter(Context context, List<String> alertDialog) {
        super(context, 0, alertDialog);
        this.context = context;
        this.alertData = alertDialog;
    }

    @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;
    }
}

```

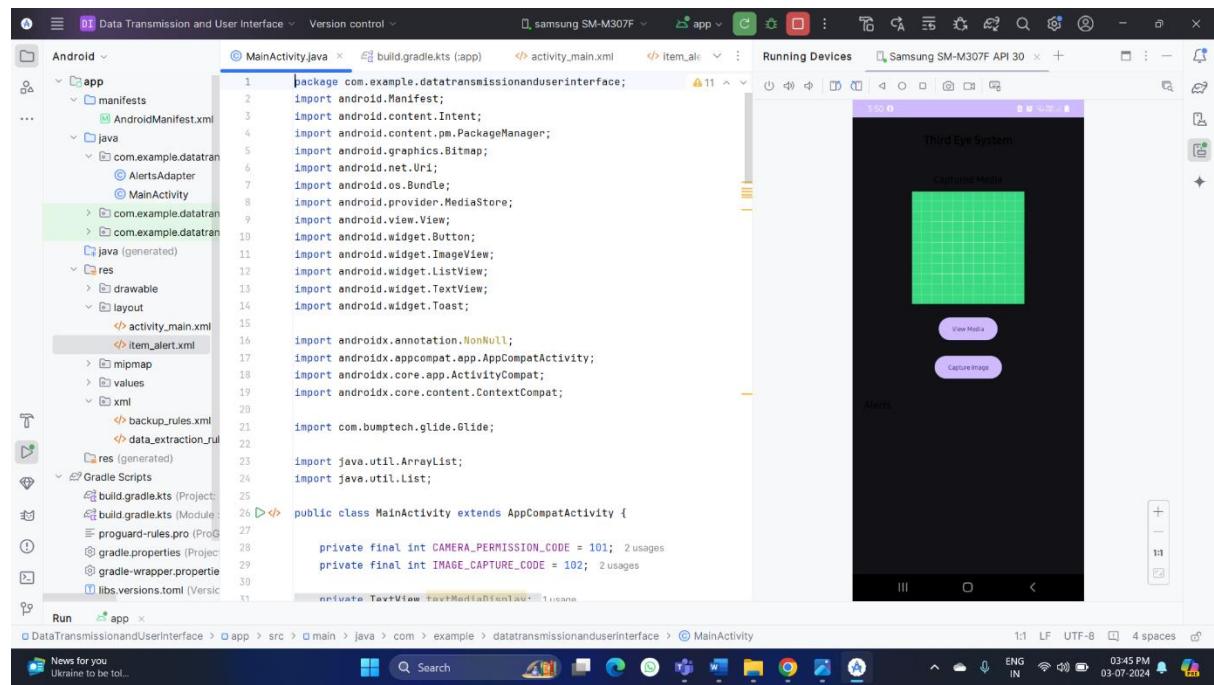
The screenshot shows the Android Studio interface with the code editor open to `AlertsAdapter.java`. This adapter class extends `ArrayAdapter` and overrides the `getView` method to inflate a custom layout (`R.layout.item_alert`) for each item in the list, setting the alert text from the adapter's data source.

AndroidManifest.xml

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `AndroidManifest.xml` file, which defines the application's permissions and activities. The manifest includes declarations for camera access, internet permission, and external storage write permission. It also specifies the main activity and launcher category.

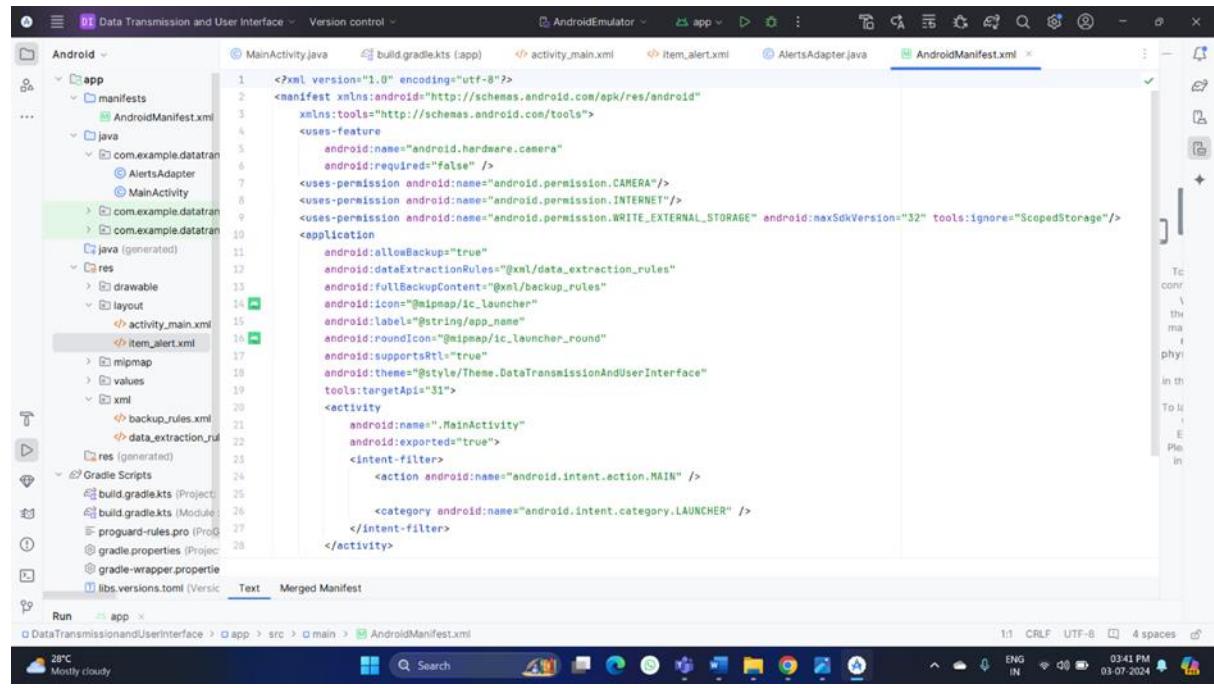
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" android:maxSdkVersion="32" tools:ignore="ScopedStorage"/>
    <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:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.DataTransmissionAndUserInterface"
        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>
```

DAY 4 TASK 1 OUTPUT SCREENSHOTS



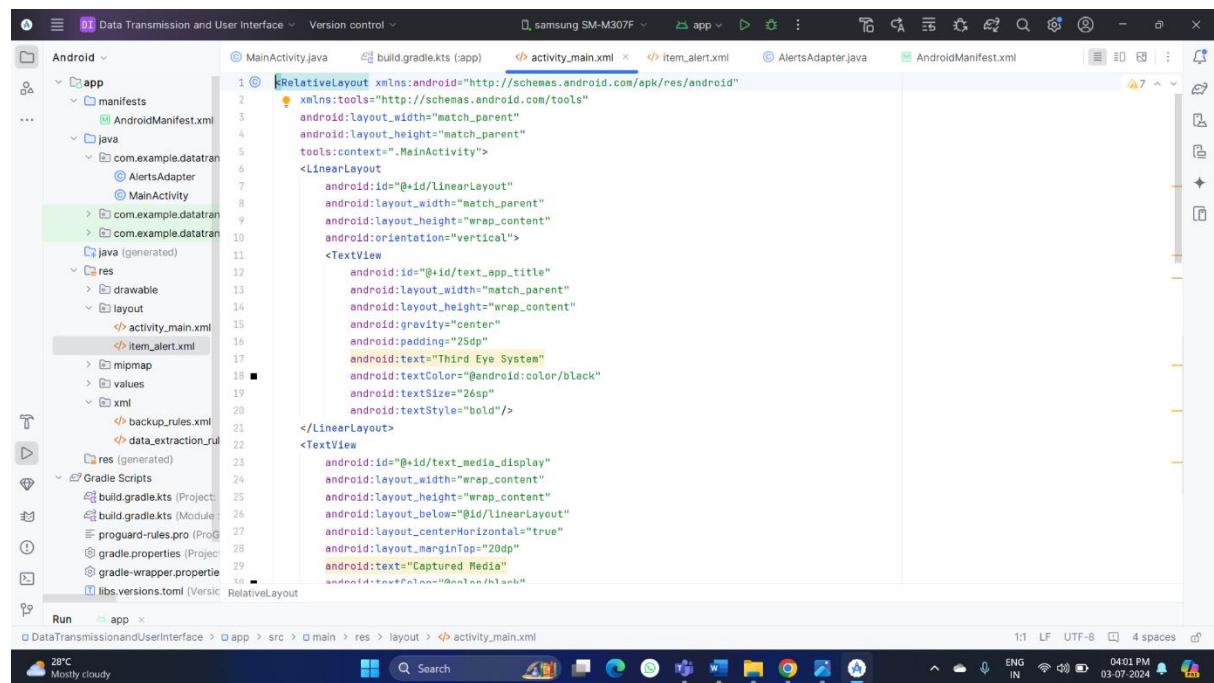
Task 2: Develop a user interface for the owner to interact with the Third Eye system, including viewing the captured media and receiving alerts.

AndroidManifest.xml



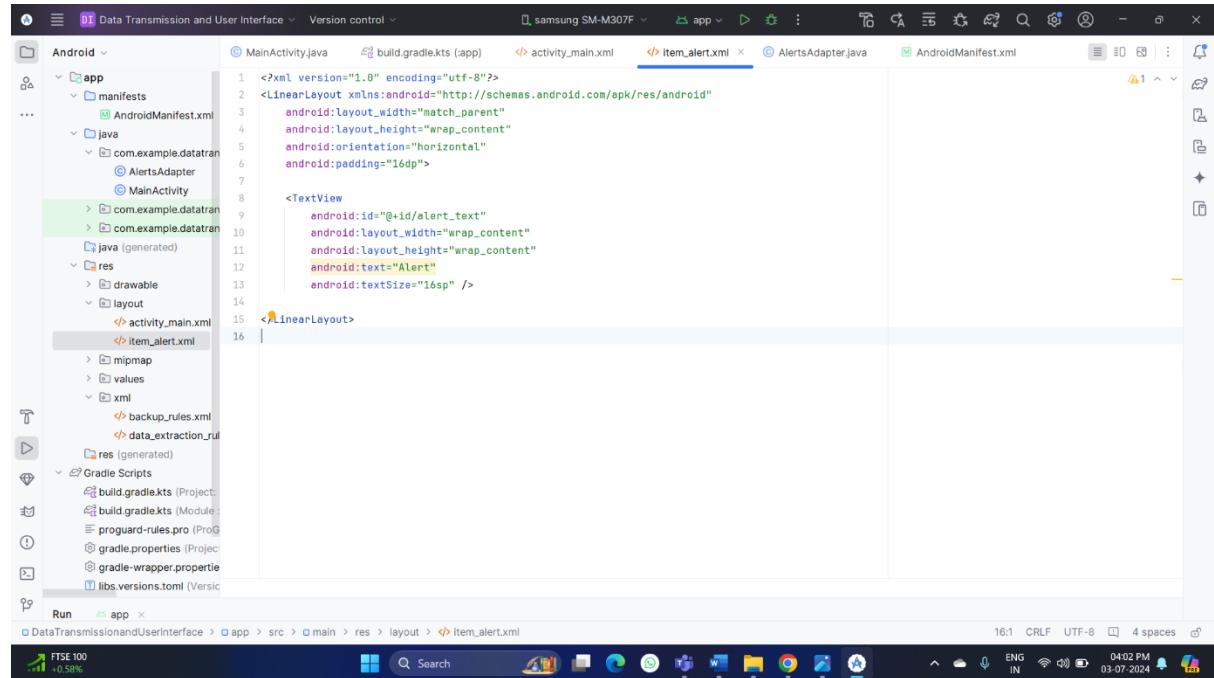
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" android:maxSdkVersion="32" tools:ignore="ScopedStorage"/>
    <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:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.DataTransmissionAndUserInterface"
        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>
```

activity_main.xml



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <TextView
            android:id="@+id/text_app_title"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:padding="25dp"
            android:text="Third Eye System"
            android:textColor="@android:color/black"
            android:textSize="26sp"
            android:textStyle="bold"/>
        </LinearLayout>
        <Textview
            android:id="@+id/text_media_display"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/linearLayout"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="20dp"
            android:text="Captured Media"
            android:textColor="#000000"/>
```

item_alert.xml

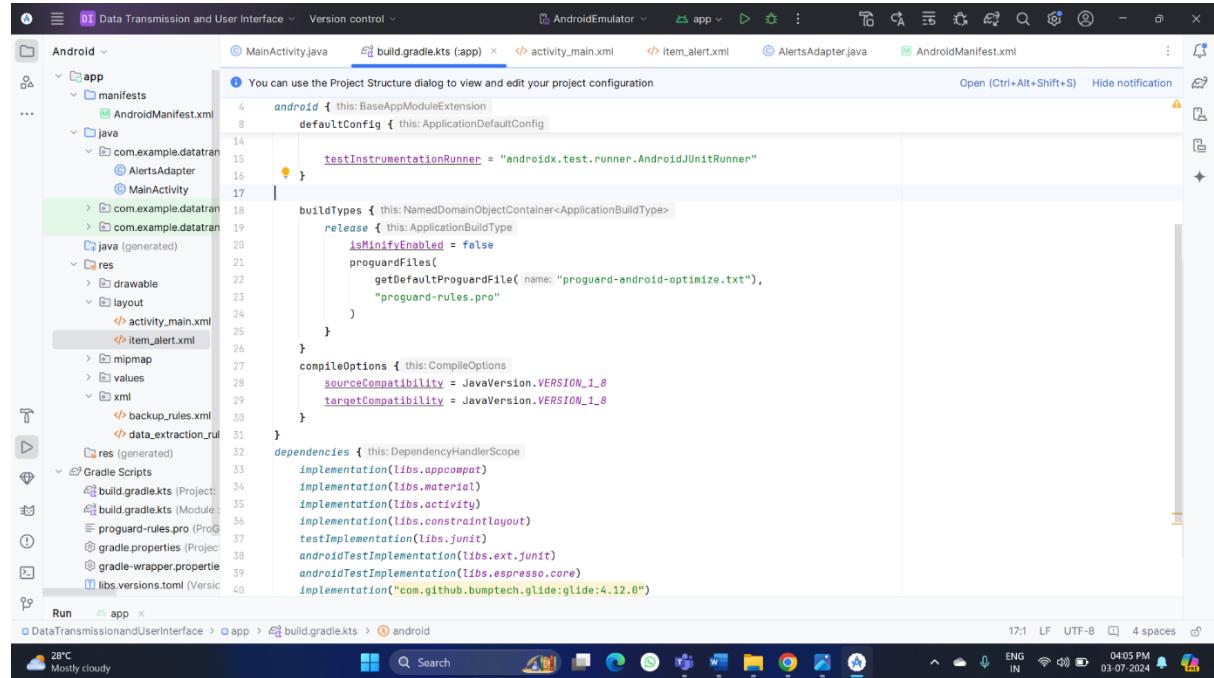


The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the XML content of the `item_alert.xml` file.

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

    <TextView
        android:id="@+id/alert_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Alert"
        android:textSize="16sp" />
</LinearLayout>
```

build.gradle.kts(:app)



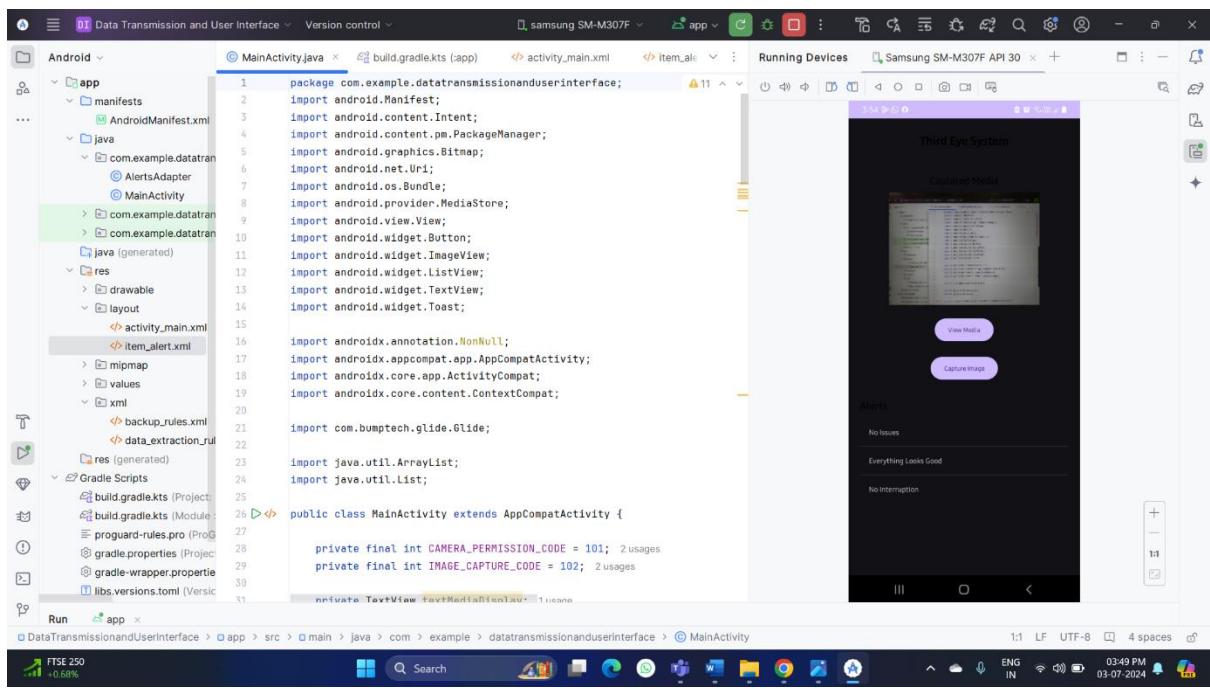
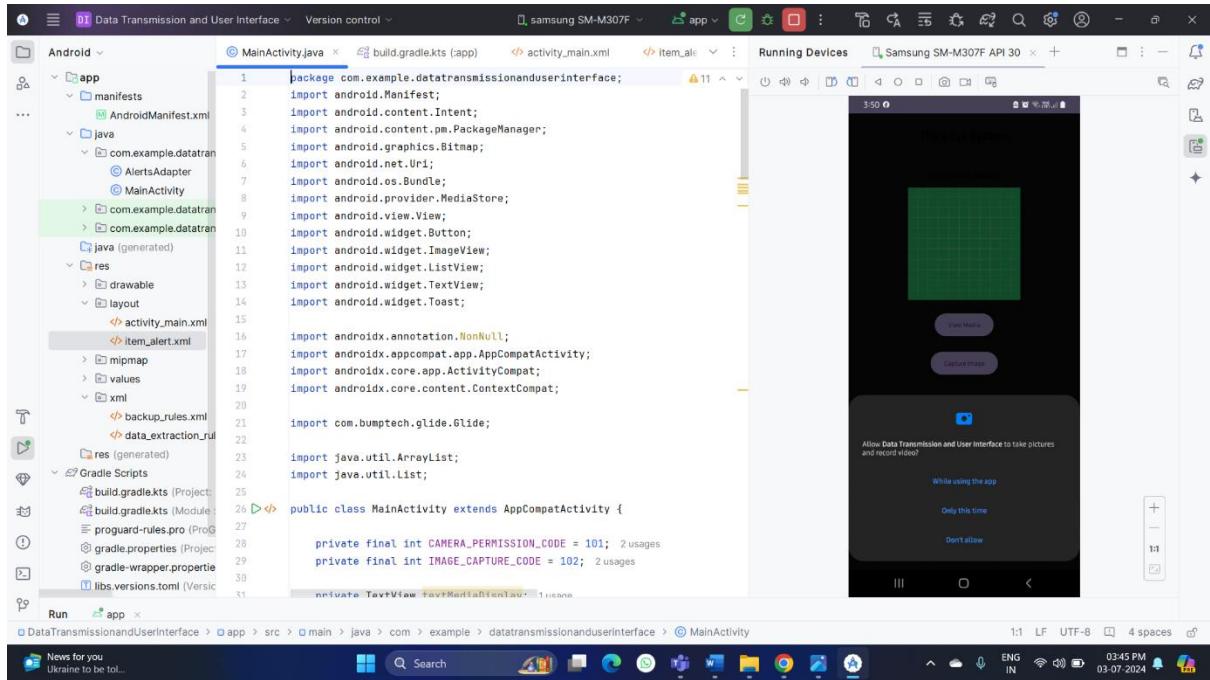
The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the Kotlin-based build script for the `:app` module.

```
android {
    defaultConfig {
        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
        compileOptions {
            sourceCompatibility = JavaVersion.VERSION_1_8
            targetCompatibility = JavaVersion.VERSION_1_8
        }
    }

    dependencies {
        implementation(libs.appcompat)
        implementation(libs.material)
        implementation(libs.activity)
        implementation(libs.constraintlayout)
        testImplementation(libs.junit)
        androidTestImplementation(libs.ext.junit)
        androidTestImplementation(libs.espresso.core)
        implementation("com.github.bumptech.glide:glide:4.12.0")
    }
}
```

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

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar shows the project structure under the package `com.example.testingsecurityandcompliance`. It includes sub-folders `app`, `java`, and `res`. The `java` folder contains several Java files: `AlertAdapter`, `ImageAdapter`, `ImageCompressor`, `ImageTransmissionService`, `MainActivity` (selected), `VibrationDetectionJobService`, and `VibrationDetectionService`. The `res` folder contains `drawable` and `layout` sub-folders with various XML files like `ic_launcher_background.xml` and `activity_main.xml`.
- Code Editor:** The main window displays the `MainActivity.java` file. The code implements a `MainActivity` class that extends `AppCompatActivity`. It handles camera permissions, initializes a `SurfaceView` and `SurfaceHolder`, and sets up a `Button` to take pictures. The `onClick` event for the button calls `takePicture()`. The code also checks for camera permission and requests it if necessary.
- Toolbars and Status Bar:** The top bar has tabs for `TestingSecurityAndCompliance`, `AndroidEmulator`, `app`, and `SystemTest`. The status bar at the bottom shows the time as 30:14, the date as LF, the battery level as 4 spaces, and the date as 05/04/2024.

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar displays the project structure under "Android". It includes the "app" module with its manifest files (AndroidManifest.xml), Java source files (MainActivity.java, ImageAdapter.java, SystemTest.java), and resource files (res/drawable/ic_launcher_background.xml, ic_launcher_foreground.xml, layout/activity_main.xml, values/itemlist.xml, itemimage.xml). It also lists sub-modules: com.example.testingsecurityandcompliance (androidTest) and com.example.testingsecurityandcompliance (test).
- MainActivity.java:** The main code editor shows the implementation of the MainActivity class. It includes methods for taking pictures and saving images to storage.
- Code Navigation:** A vertical bar on the right side of the code editor highlights specific lines of code, likely indicating the current scope or context of the selected line.

AlertAdapter.java

The screenshot shows the Android Studio interface with the code editor open to AlertAdapter.java. The code implements a RecyclerView.Adapter for displaying alerts. It includes imports for Context, View, ViewGroup, TextView, NonNull, Recycler View, and Util.List. The class AlertAdapter extends RecyclerView.Adapter<AlertAdapter.ViewHolder>. It has a private final Context context and a private final List<String> alerts. The onCreateViewHolder method inflates R.layout.itemalert. The onBindViewHolder method sets the text of a TextView in the layout to the alert at the given position. getItemCount returns the size of the alerts list. A ViewHolder class extends RecyclerView.ViewHolder and contains a TextView for displaying the alert text.

```
1 package com.example.testingsecurityandcompliance;
2 import android.content.Context;
3 import android.view.LayoutInflater;
4 import android.view.View;
5 import android.view.ViewGroup;
6 import android.widget.TextView;
7 import androidx.annotation.NonNull;
8 import androidx.recyclerview.widget.RecyclerView;
9 import java.util.List;
10 public class AlertAdapter extends RecyclerView.Adapter<AlertAdapter.ViewHolder> {
11     private final Context context;
12     private final List<String> alerts;
13     public AlertAdapter(Context context, List<String> alerts) {
14         this.context = context;
15         this.alerts = alerts;
16     }
17     @NonNull
18     @Override
19     public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
20         View view = LayoutInflater.from(context).inflate(R.layout.itemalert, parent, false);
21         return new ViewHolder(view);
22     }
23     @Override
24     public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
25         String alert = alerts.get(position);
26         holder.alertTextView.setText(alert);
27     }
28     @Override
29     public int getItemCount() { return alerts.size(); }
30     static class ViewHolder extends RecyclerView.ViewHolder {
31         TextView alertTextView;
32         ViewHolder(@NonNull View itemView) {
33             alertTextView = itemView.findViewById(R.id.alert_text);
34         }
35     }
36 }
```

ImageAdapter.java

The screenshot shows the Android Studio interface with the code editor open to ImageAdapter.java. The code implements a RecyclerView.Adapter for displaying image paths. It includes imports for Context, View, ViewGroup, ImageView, NonNull, Recycler View, Glide, and Util.List. The class ImageAdapter extends RecyclerView.Adapter<ImageAdapter.ViewHolder>. It has a private final Context context and a private final List<String> imagePaths. The onCreateViewHolder method inflates R.layout.itemimage. The onBindViewHolder method uses Glide.with(context).load(imagePath).into(holder.imageView) to set the image view. getItemCount returns the size of the imagePaths list. A ViewHolder class extends RecyclerView.ViewHolder and contains an ImageView for displaying the image path.

```
1 package com.example.testingsecurityandcompliance;
2 import android.content.Context;
3 import android.view.LayoutInflater;
4 import android.view.View;
5 import android.view.ViewGroup;
6 import android.widget.ImageView;
7 import androidx.annotation.NonNull;
8 import androidx.recyclerview.widget.RecyclerView;
9 import com.bumptech.glide.Glide;
10 import java.util.List;
11 public class ImageAdapter extends RecyclerView.Adapter<ImageAdapter.ViewHolder> {
12     private final Context context;
13     private final List<String> imagePaths;
14     public ImageAdapter(Context context, List<String> imagePaths) {
15         this.context = context;
16         this.imagePaths = imagePaths;
17     }
18     @NonNull
19     @Override
20     public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
21         View view = LayoutInflater.from(context).inflate(R.layout.itemimage, parent, false);
22         return new ViewHolder(view);
23     }
24     @Override
25     public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
26         String imagePath = imagePaths.get(position);
27         Glide.with(context).load(imagePath).into(holder.imageView);
28     }
29     @Override
30     public int getItemCount() { return imagePaths.size(); }
31     static class ViewHolder extends RecyclerView.ViewHolder {
32         ImageView imageView;
33         ViewHolder(@NonNull View itemView) {
34             imageView = itemView.findViewById(R.id.image_view);
35         }
36     }
37 }
```

ImageCompressor.java

The screenshot shows the Android Studio interface with the code editor open to the `ImageCompressor.java` file. The code implements a static method to compress a bitmap. The code editor has syntax highlighting and code completion suggestions.

```
1 package com.example.testingsecurityandcompliance;
2 import android.graphics.Bitmap;
3 import android.graphics.BitmapFactory;
4 import java.io.ByteArrayOutputStream;
5
6 public class ImageCompressor {
7     @Override
8     public static byte[] compressImage(byte[] imageData) {
9         BitmapFactory.Options options = new BitmapFactory.Options();
10        options.inSampleSize = 2;
11        Bitmap bmp = BitmapFactory.decodeByteArray(imageData, 0, imageData.length, options);
12
13        ByteArrayOutputStream baos = new ByteArrayOutputStream();
14        bmp.compress(Bitmap.CompressFormat.JPEG, 80, baos);
15        return baos.toByteArray();
16    }
}
```

ImageTransmissionService.java

The screenshot shows the Android Studio interface with the code editor open to the `ImageTransmissionService.java` file. The code defines a service that overrides the `onStartCommand` and `onBind` methods. The code editor has syntax highlighting and code completion suggestions.

```
1 package com.example.testingsecurityandcompliance;
2 import android.app.Service;
3 import android.content.Intent;
4 import android.os.IBinder;
5
6 public class ImageTransmissionService extends Service {
7     @Override
8     public int onStartCommand(Intent intent, int flags, int startId) {
9         // Code to transmit images and alerts
10        return START_STICKY;
11    }
12
13    @Override
14    public IBinder onBind(Intent intent) { return null; }
15}

```

VibrationDetectionJobService.java

The screenshot shows the Android Studio interface with the code editor open to the `VibrationDetectionJobService.java` file. The code implements a `JobService` that starts itself when triggered by a job. It includes methods for handling start and stop jobs, and a scheduled task to trigger the next job.

```
1 package com.example.testingsecurityandcompliance;
2 import android.app.job.JobParameters;
3 import android.app.job.JobService;
4 import android.content.Intent;
5 import android.os.Handler;
6 import android.util.Log;
7
8 public class VibrationDetectionJobService extends JobService {
9     private static final String TAG = "VibrationDetectionJob";
10    @Override
11    public boolean onStartJob(JobParameters params) {
12        Log.d(TAG, msg: "Job started");
13        Intent serviceIntent = new Intent(packageContext: this, VibrationDetectionService.class);
14        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
15            startForegroundService(serviceIntent);
16        } else {
17            startService(serviceIntent);
18        }
19
20        scheduleNextJob();
21        return false;
22    }
23    @Override
24    public boolean onStopJob(JobParameters params) {
25        Log.d(TAG, msg: "Job stopped");
26        return true;
27    }
28    private void scheduleNextJob() { 1 usage
29        Handler handler = new Handler();
30        handler.postDelayed(() -> {
31            // Code to schedule the next job
32        }, delayMillis: 15 * 60 * 1000); // delay for 15 minutes
33    }
34}
```

Below the code editor, the status bar shows the date as 03-07-2024 and the time as 05:52 PM.

VibrationDetectionService.java

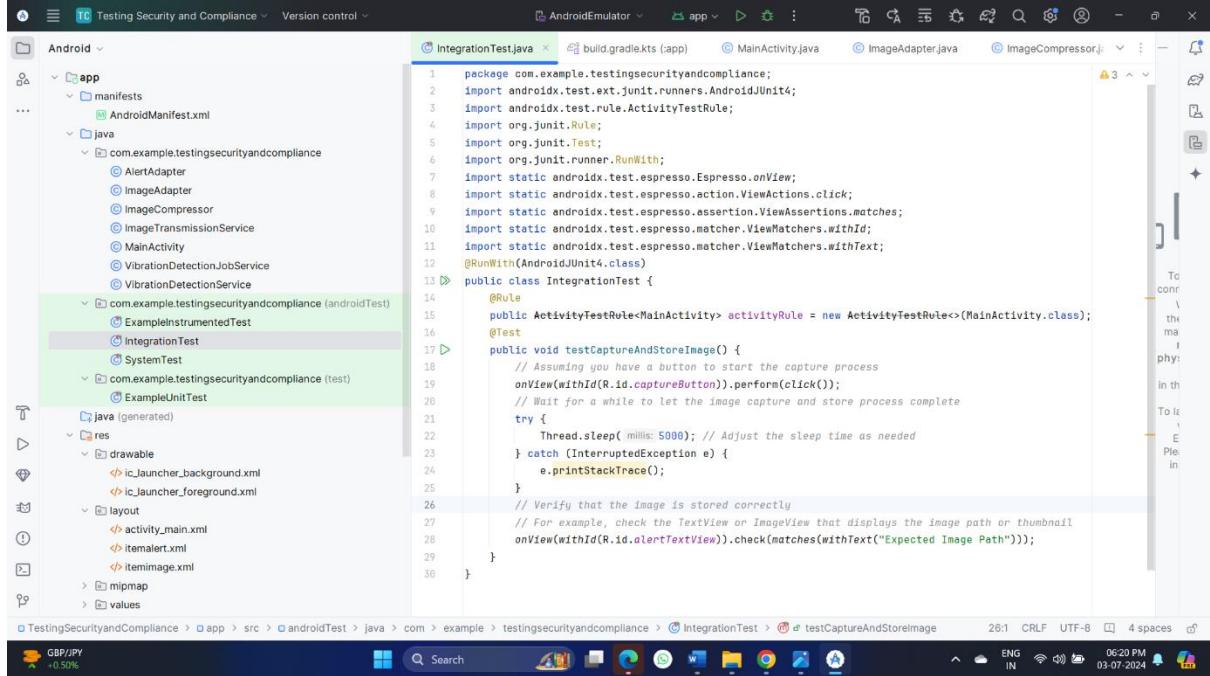
The screenshot shows the Android Studio interface with the code editor open to the `VibrationDetectionService.java` file. The code defines a `Service` that handles start commands and binds to it. It includes an `onStartCommand` method to start vibration detection logic.

```
1 package com.example.testingsecurityandcompliance;
2 import android.app.Service;
3 import android.content.Intent;
4 import android.os.IBinder;
5
6 public class VibrationDetectionService extends Service {
7     @Override
8     public int onStartCommand(Intent intent, int flags, int startId) {
9         // Start vibration detection logic here
10        return START_STICKY;
11    }
12
13    @Override
14    public IBinder onBind(Intent intent) { return null; }
15}
16
```

Below the code editor, the status bar shows the date as 03-07-2024 and the time as 05:53 PM.

androidTest

IntegrationTest.java

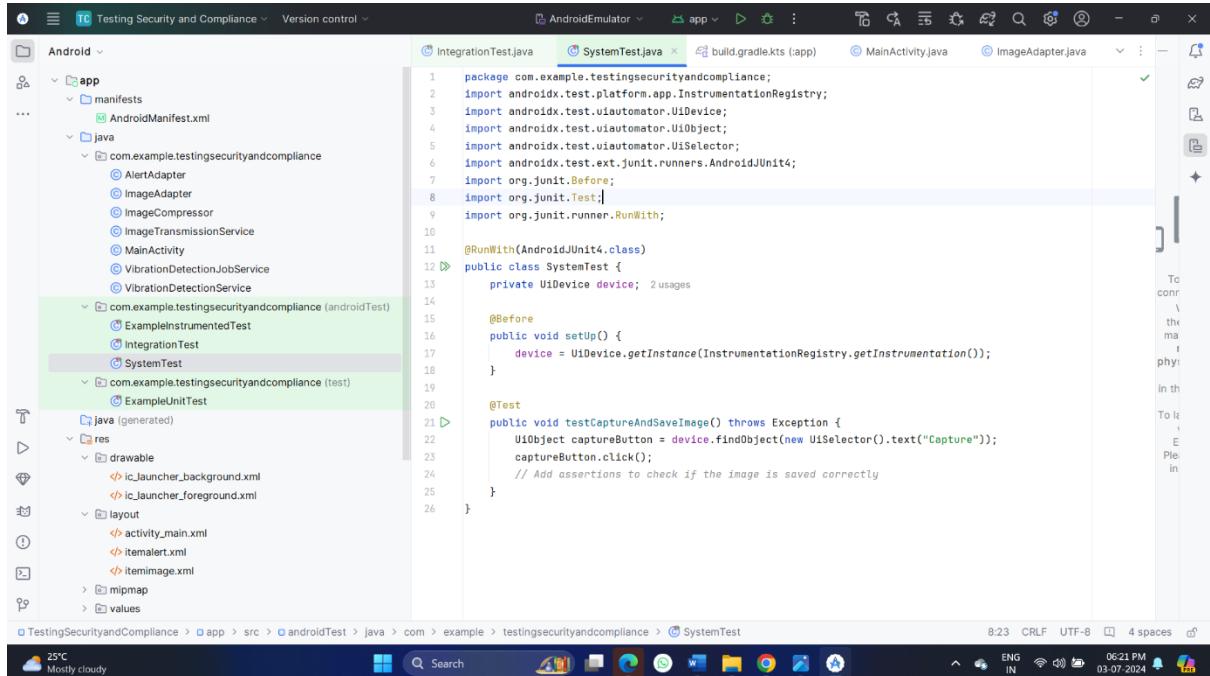


The screenshot shows the Android Studio interface with the 'Testing Security and Compliance' project selected. The left sidebar shows the project structure under the 'Android' tab, including the 'app' module with its manifest, Java files (MainActivity, ImageAdapter, ImageCompressor, ImageTransmissionService, MainActivity, VibrationDetectionJobService, VibrationDetectionService), and resource files (drawable, layout, mipmap, values). The right panel displays the 'IntegrationTest.java' code:

```
1 package com.example.testingsecurityandcompliance;
2 import androidx.test.ext.junit.runners.AndroidJUnit4;
3 import androidx.test.rule.ActivityTestRule;
4 import org.junit.Rule;
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7 import static androidx.test.espresso.Espresso.onView;
8 import static androidx.test.espresso.action.ViewActions.click;
9 import static androidx.test.espresso.assertion.ViewAssertions.matches;
10 import static androidx.test.espresso.matcher.ViewMatchers.withId;
11 import static androidx.test.espresso.matcher.ViewMatchers.withText;
12 @RunWith(AndroidJUnit4.class)
13 public class IntegrationTest {
14     @Rule
15     public ActivityTestRule<MainActivity> activityRule = new ActivityTestRule<>(MainActivity.class);
16     @Test
17     public void testCaptureAndStoreImage() {
18         // Assuming you have a button to start the capture process
19         onView(withId(R.id.captureButton)).perform(click());
20         // Wait for a while to let the image capture and store process complete
21         try {
22             Thread.sleep(5000); // Adjust the sleep time as needed
23         } catch (InterruptedException e) {
24             e.printStackTrace();
25         }
26         // Verify that the image is stored correctly
27         // For example, check the TextView or ImageView that displays the image path or thumbnail
28         onView(withId(R.id.alertTextView)).check(matches(withText("Expected Image Path")));
29     }
30 }
```

The status bar at the bottom shows 'TestingSecurityandCompliance > app > src > androidTest > java > com > example > testingsecurityandcompliance > IntegrationTest > testCaptureAndStoreImage'. The bottom right corner shows '26:1 CRLF UTF-8 4 spaces' and the date '03-07-2024'.

SystemTest.java

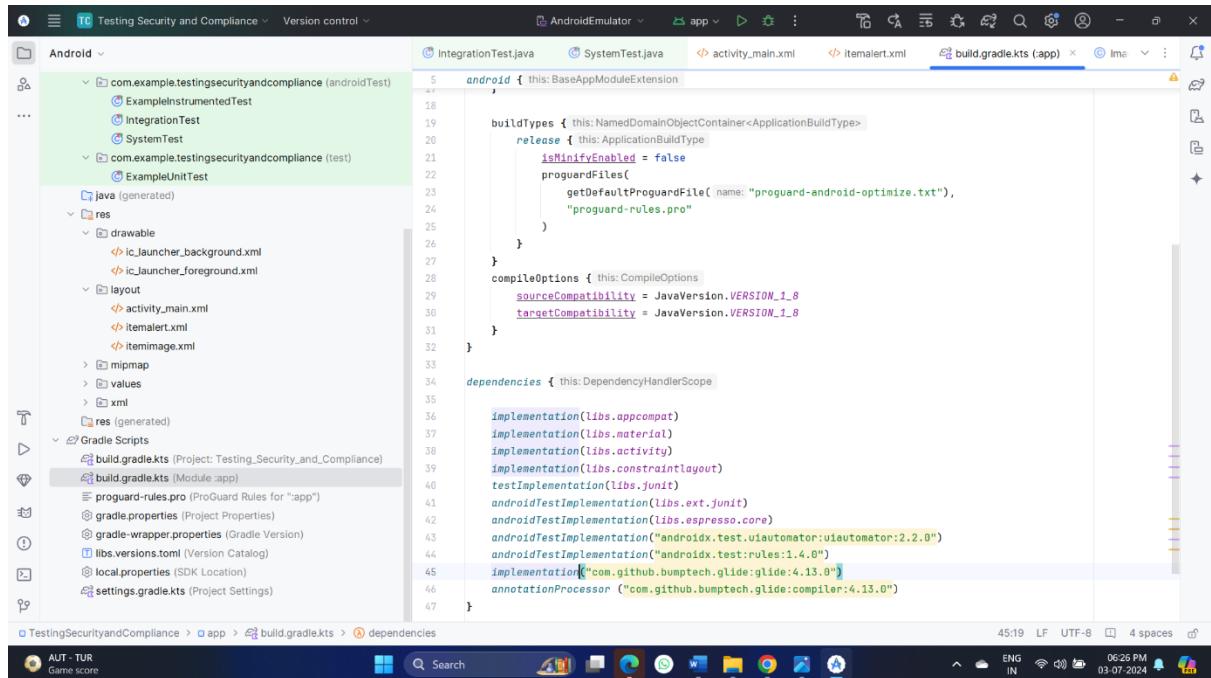


The screenshot shows the Android Studio interface with the 'Testing Security and Compliance' project selected. The left sidebar shows the project structure under the 'Android' tab, including the 'app' module with its manifest, Java files (MainActivity, ImageAdapter, ImageCompressor, ImageTransmissionService, MainActivity, VibrationDetectionJobService, VibrationDetectionService), and resource files (drawable, layout, mipmap, values). The right panel displays the 'SystemTest.java' code:

```
1 package com.example.testingsecurityandcompliance;
2 import androidx.test.platform.app.InstrumentationRegistry;
3 import androidx.test.uiautomator.UiDevice;
4 import androidx.test.uiautomator.UiObject;
5 import androidx.test.uiautomator.UiSelector;
6 import androidx.test.ext.junit.runners.AndroidJUnit4;
7 import org.junit.Before;
8 import org.junit.Test;
9 import org.junit.runner.RunWith;
10
11 @RunWith(AndroidJUnit4.class)
12 public class SystemTest {
13     private UiDevice device; 2 usages
14
15     @Before
16     public void setUp() {
17         device = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation());
18     }
19
20     @Test
21     public void testCaptureAndSaveImage() throws Exception {
22         UiObject captureButton = device.findObject(new UiSelector().text("Capture"));
23         captureButton.click();
24         // Add assertions to check if the image is saved correctly
25     }
26 }
```

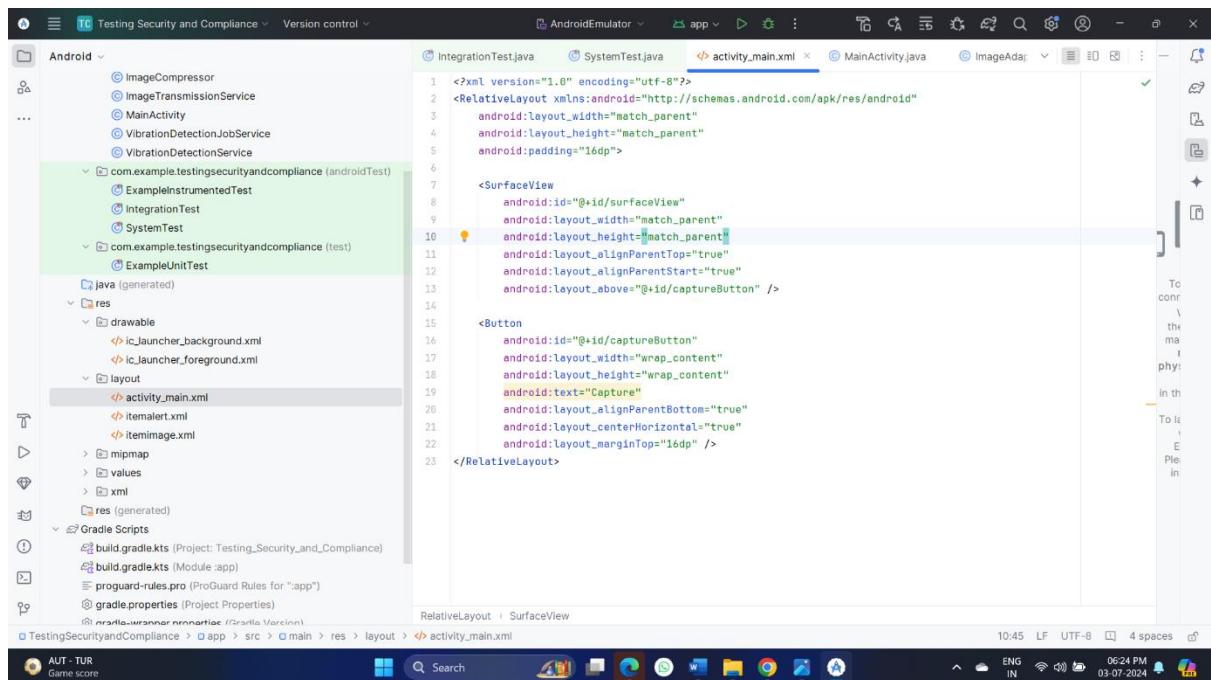
The status bar at the bottom shows 'TestingSecurityandCompliance > app > src > androidTest > java > com > example > testingsecurityandcompliance > SystemTest > testCaptureAndSaveImage'. The bottom right corner shows '8:23 CRLF UTF-8 4 spaces' and the date '03-07-2024'.

build.gradle.kts(Module:app)



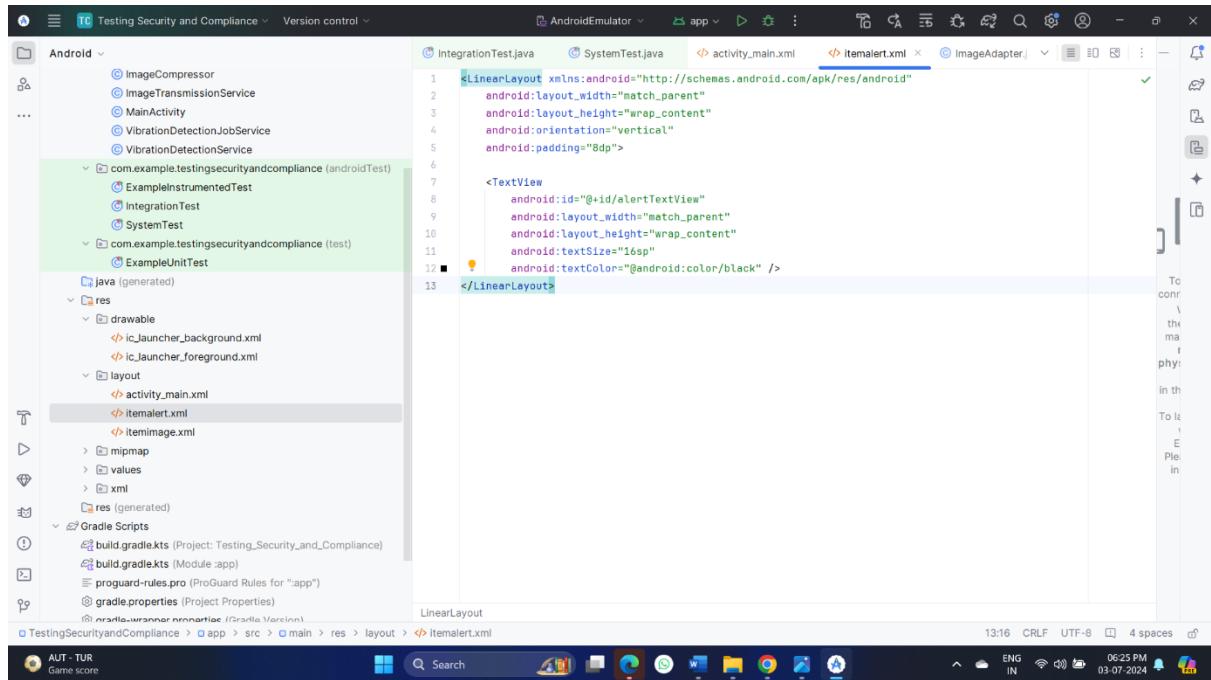
```
5 android { this:BaseAppModuleExtension
18
19 release { this:ApplicationBuildType
20     isMinifyEnabled = false
21     proguardFiles(
22         getDefaultProguardFile( name: "proguard-android-optimize.txt"),
23         "proguard-rules.pro"
24     )
25 }
26
27 compileOptions { this:CompileOptions
28     sourceCompatibility = JavaVersion.VERSION_1_8
29     targetCompatibility = JavaVersion.VERSION_1_8
30 }
31
32 }
33
34 dependencies { this:DependencyHandlerScope
35
36     implementation(libs.appcompat)
37     implementation(libs.material)
38     implementation(libs.activity)
39     implementation(libs.constraintlayout)
40     testImplementation(libs.junit)
41     androidTestImplementation(libs.ext.junit)
42     androidTestImplementation(libs.espresso.core)
43     androidTestImplementation("androidx.test.uiautomator:uiautomator:2.2.0")
44     androidTestImplementation("androidx.test:rules:1.4.0")
45     implementation("com.github.bumptech.glide:glide:4.13.0")
46     annotationProcessor ("com.github.bumptech.glide:compiler:4.13.0")
47 }
```

activity_main.xml



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:padding="16dp">
6
7     <SurfaceView
8         android:id="@+id/surfaceView"
9         android:layout_width="match_parent"
10        android:layout_height="match_parent"
11        android:layout_alignParentTop="true"
12        android:layout_alignParentStart="true"
13        android:layout_above="@+id/captureButton" />
14
15     <Button
16         android:id="@+id/captureButton"
17         android:layout_width="wrap_content"
18         android:layout_height="wrap_content"
19         android:text="Capture"
20         android:layout_alignParentBottom="true"
21         android:layout_centerHorizontal="true"
22         android:layout_marginTop="16dp" />
23     </RelativeLayout>
```

itemalert.xml



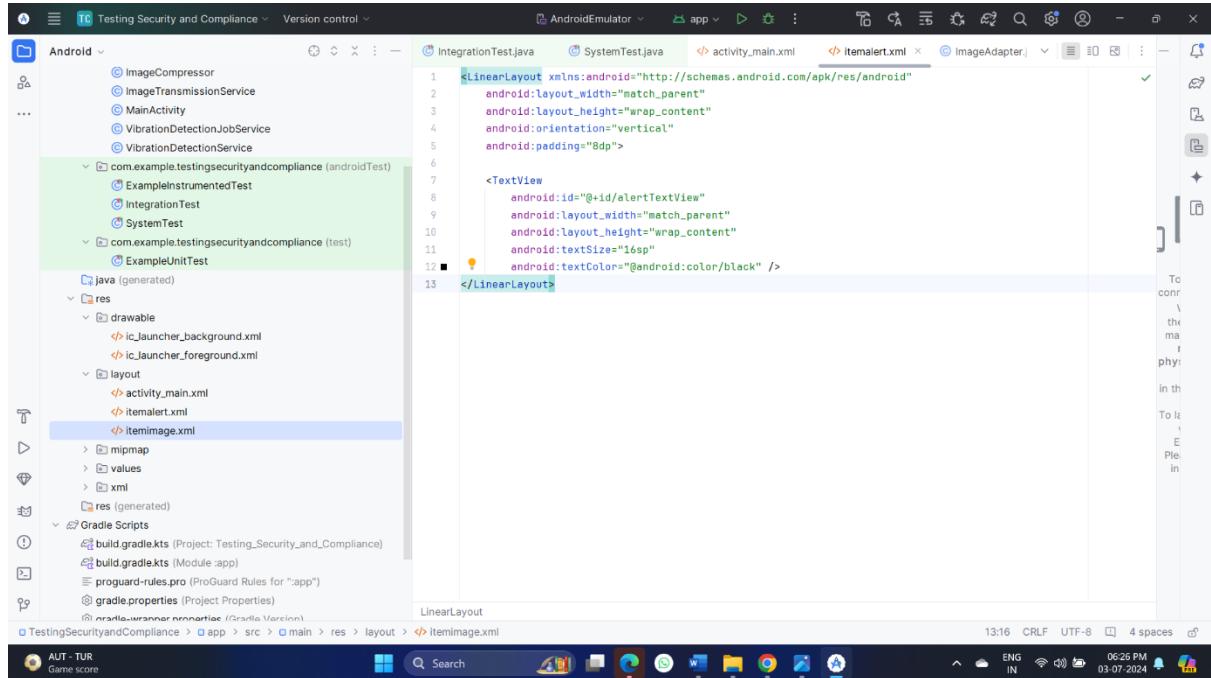
The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the XML content for itemalert.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="8dp"

    <TextView
        android:id="@+id/alertTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:textColor="@android:color/black" />
</LinearLayout>
```

The file is located at `TestingSecurityandCompliance/app/src/main/res/layout/itemalert.xml`. The code editor has syntax highlighting and a status bar at the bottom showing the date and time.

itemimage.xml



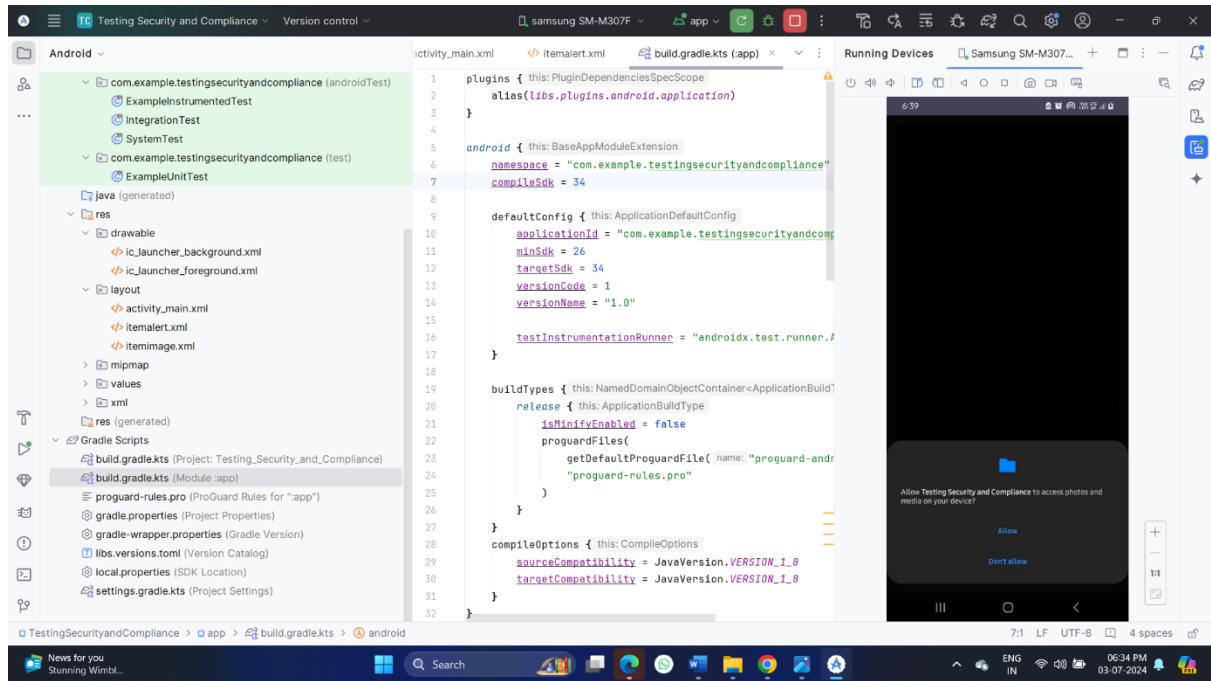
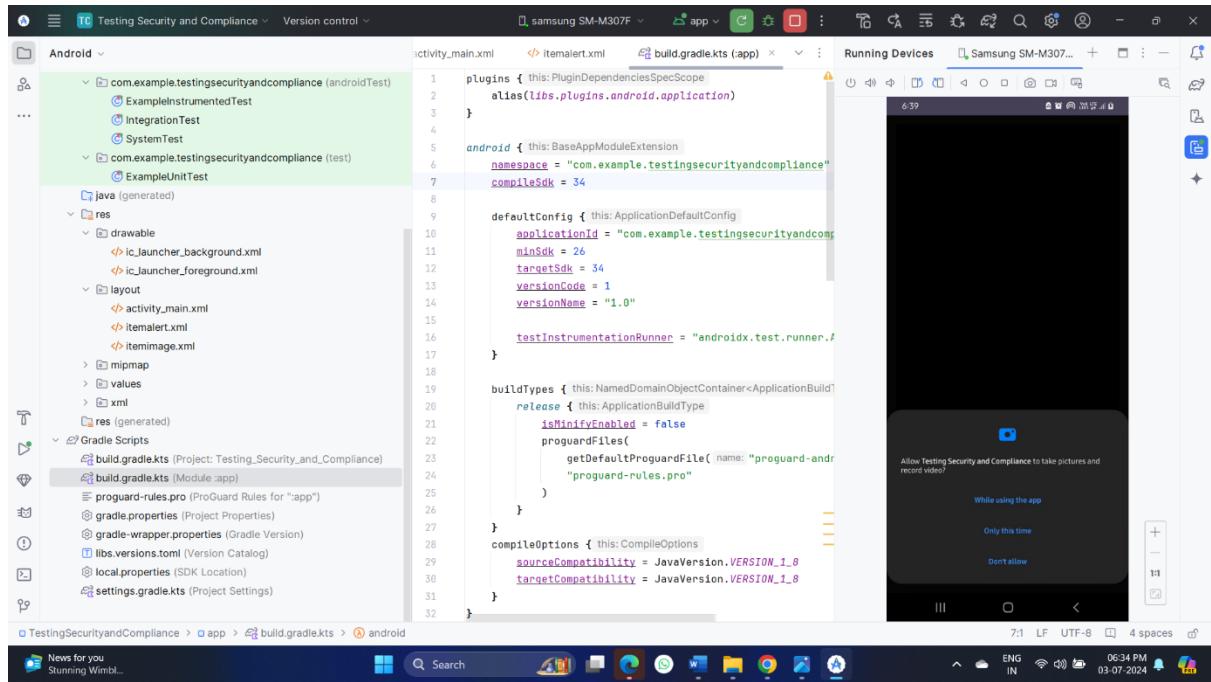
The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the XML content for itemimage.xml:

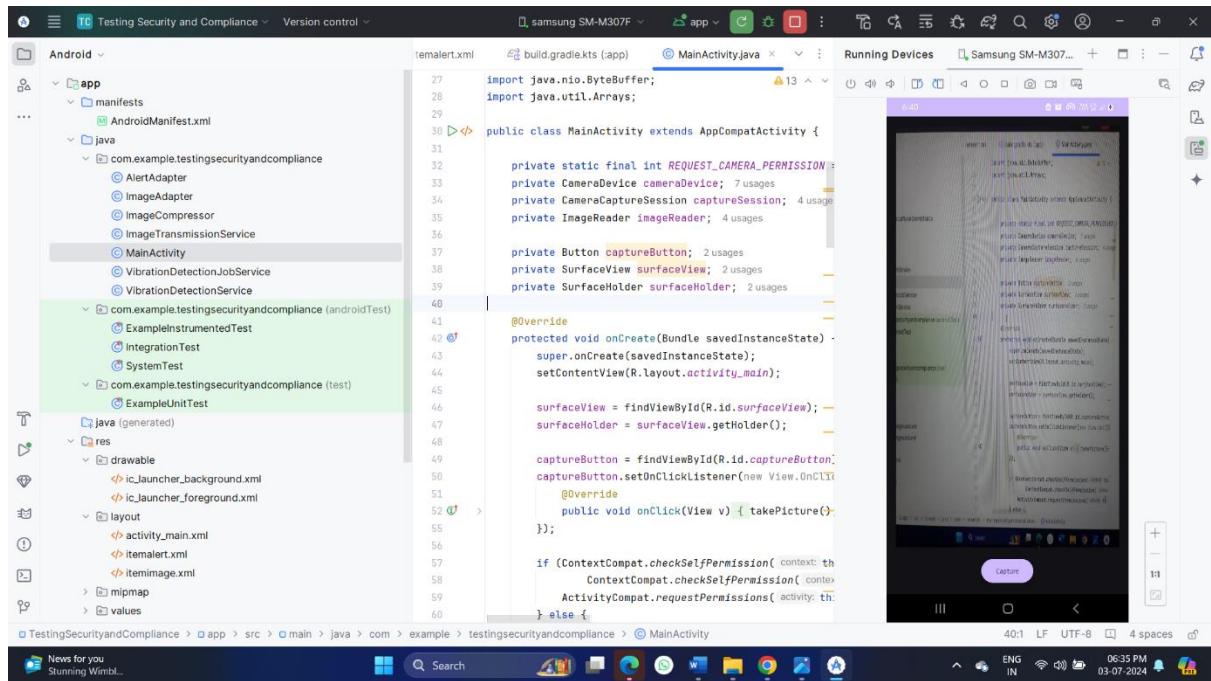
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="8dp"

    <TextView
        android:id="@+id/alertTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:textColor="@android:color/black" />
</LinearLayout>
```

The file is located at `TestingSecurityandCompliance/app/src/main/res/layout/itemimage.xml`. The code editor has syntax highlighting and a status bar at the bottom showing the date and time.

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.