UNIVERSITY OF PIRAEUS - DEPARTMENT OF INFORMATICS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSc «Informatics»

ΠΜΣ «Πληροφορική»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**Mobile application for prevention of accidents during scheduled maintenance of equipment.** Εφαρμογή κινητών συσκευών για την αποφυγή ατυχημάτων κατά τη συντήρηση εξοπλισμού.
Student's name-surname: Ονοματεπώνυμο φοιτητή:	**Sotirios-Andreas Kappos** Σωτήριος-Ανδρέας Κάππος
Father's name: Πατρώνυμο:	**Vasileios** Βασίλειος
Student's ID No: Αριθμός Μητρώου:	ΜΠΠΛ/18027
Supervisor: Επιβλέπων:	**Efthimios Alepis, Associate Professor** Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής

December 2023/ Δεκέμβριος 2023

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

Efthimios Alepis	**Maria Virvou**	**Evangelos Sakkopoulos**
Associate Professor	**Professor**	**Associate Professor**
Ευθύμιος Αλέπης	Μαρία Βίρβου	Ευάγγελος Σακκόπουλος
Αναπληρωτής Καθηγητής	Καθηγήτρια	Αναπληρωτής Καθηγητής

Contents

Περίληψη (Abstract)

Ελληνικά

Στην παρούσα μεταπτυχιακή διατριβή θα γίνει παρουσίαση της εφαρμογής ειδοποιήσεων για κινητές συσκευές NotificationApp Extreme .Η λειτουργικότητα (Activity 3)της οποίας θα γίνει αναλυτική περιγραφή παρακάτω θα κάνει χρήση, της σύνδεσης με το διαδίκτυο και της εφαρμογής ηλεκτρονικού ταχυδρομείου μέσα από την οποία θα γίνεται αποστολή ειδοποιήσεων σε προκαθορισμένο χρήστη. Σε συνεργασία με την ανάγνωση δεδομένων σε πραγματικό χρόνο από σύστημα τοποθετημένων αισθητήρων που στην παρούσα εφαρμογή για χάρη λειτουργικότητας έχουν αντικατασταθεί από μια βάση δεδομένων Firebase θα δημιουργούνται ειδοποιήσεις αν οι συνθήκες δεν ευνοούν για τη διεξαγωγή εργασιών . Σε συνέχεια της ανωτέρω λειτουργικότητας θα γίνεται χρήση των αισθητήρων επιτάχυνσης της κινητής συσκευής του χρήστη (Activity 2) όπου αν υπάρχει διάγνωση ακραίων τιμών θα ειδοποιείται και ο χρήστης με μήνυμα καθώς και ο οργανισμός εργασίας του χρήστη για πιθανό ατύχημα.

English

In this master's thesis the application NotificationApp Extreme for mobile devices will be presented. The functionality (Activity 3) which will be detailed below will make use of the internet connection and the e-mail application through which notifications will be sent to user preset. In cooperation with the reading of data in real time from a system of placed sensors that in this application for the sake of functionality have been replaced by a Firebase database, alerts will be generated if the conditions are not favorable for carrying out work. In continuation of the above functionality, the acceleration sensors of the user's mobile device will be used (Activity 2) where if there is a diagnosis of extreme values, the user will be notified with a message as well as the user's work organization for a possible accident.

Introduction

Outdoor machine maintenance is an essential facet of various industries, contributing to the smooth operation of heavy machinery that powers our modern world. However, this crucial task comes with inherent risks, as technicians often work in challenging environments where accidents can occur. Accidents not only pose a threat to the well-being of maintenance personnel but can also lead to substantial financial losses and downtime for businesses. In recent years, advancements in technology have emerged as a beacon of hope for mitigating these risks and significantly lowering the accident rates associated with outdoor machine maintenance.

This thesis delves into the critical importance of preventing accidents during outdoor machine maintenance of solar panels and wind turbines and explores how technological innovations have become indispensable tools in achieving this goal. By examining the challenges faced by maintenance workers, the thesis aims to shed light on the transformative impact of technology on promoting a safer work environment, reducing the occurrence of accidents, and ultimately enhancing overall operational efficiency.

The following sections will navigate through the diverse challenges encountered and the technological solutions that have emerged to combat them. From the deployment of advanced sensors and monitoring systems to the integration of artificial intelligence and machine learning algorithms, in the future.

As we embark on this exploration, it becomes evident that the fusion of technology with traditional maintenance practices not only minimizes the risk of accidents but also brings about a paradigm shift in how we approach industrial safety. By embracing innovation, organizations not only safeguard their most valuable asset—their workforce—but also foster an environment of continuous improvement and resilience against potential hazards. This thesis seeks to illuminate the symbiotic relationship between technology and safety in outdoor machine maintenance, demonstrating how these advancements are instrumental in shaping a future where accidents are the exception, not the norm.

Purpose of creating the specific application.

As the global landscape witnesses an increase in extreme weather events, the need for innovative solutions to mitigate their impact on human safety becomes paramount. Accidents resulting from adverse weather conditions, such as high wind speeds, extreme temperatures, and intense irradiance, pose significant threats to individuals, communities, and industries worldwide. This essay explores the creation of a mobile notification application designed to prevent accidents arising from extreme weather conditions, aligning with the growing importance of zero-accident policies adopted by companies and the humanitarian approach to safeguarding lives.

The Global Reality of Weather-Related Accidents:

Accidents related to extreme weather conditions are not isolated occurrences; they constitute a global challenge that transcends geographical boundaries. According to [insert reference to relevant statistics], the frequency and severity of weather-related accidents have shown an alarming upward trend in recent years. From workplace incidents to vehicular accidents and outdoor activities, individuals are vulnerable to the unpredictable nature of extreme weather events. This calls for a comprehensive and technologically advanced approach to enhance safety measures.

Zero Accident Policies: A Corporate Imperative:

In response to the escalating rates of weather-induced accidents, companies worldwide are embracing zero-accident policies as a fundamental component of their corporate responsibility. These policies, driven by a commitment to employee welfare and public safety, demand proactive measures to eliminate accidents entirely. The integration of technology into these policies becomes instrumental in achieving this ambitious goal. A mobile notification application serves as a real-time, accessible tool to keep individuals informed about changing weather conditions, enabling them to make informed decisions to avoid potential risks.

Humanitarian Approach to Technological Innovation:

Beyond corporate initiatives, a humanitarian approach underscores the importance of leveraging technology to protect lives and prevent weather-related accidents. As extreme weather events increasingly impact vulnerable populations, deploying accessible and user-friendly tools becomes a moral imperative. The development of a notification application reflects a commitment to the broader community, ensuring that vital weather information reaches those who need it the most. This aligns with the principles of humanitarianism, emphasizing the duty to alleviate suffering and protect the dignity of every individual.

Creating a Notification Application.

The envisioned notification application will utilize cutting-edge weather monitoring technologies, accessing real-time data from meteorological sources. Users will receive alerts and warnings tailored to their location, providing insights into impending weather conditions. Features may include wind speed notifications, temperature alerts, and UV index warnings, empowering individuals to make informed decisions about outdoor activities or work engagements.

In conclusion, the creation of a mobile notification application to prevent accidents caused by extreme weather conditions is a critical step towards fostering a safer and more resilient global community. The urgency of addressing weather-related accidents, coupled with the adoption of zero-accident policies and a humanitarian perspective, highlights the need for innovative solutions. By embracing mobile technology, we can empower individuals, enhance corporate safety measures, and contribute to a world where accidents due to extreme weather are not just reduced but become a rare exception in the pursuit of human well-being.

Popular accident preventing mobile applications.

Here are some types of apps that focus on safety and emergency assistance:

Driver Safety Apps: Applications like Life360 or SafeDrive offer features for monitoring driving behavior, detecting sudden movements, and providing feedback on safe driving practices.

Emergency Assistance Apps: Applications like Noonlight or OnWatch provide emergency assistance by connecting users with monitoring services or contacting emergency services when activated.

Roadside Assistance Apps: Services like AAA, Honk, or Urgently provide roadside assistance, including services for flat tires, fuel delivery, and towing in case of vehicle breakdowns.

Navigation Apps with Safety Features: Navigation Applications like Waze or Google Maps often provide real-time traffic updates, accident alerts, and alternative route suggestions to help users avoid potential hazards.

Weather Apps: Weather applications such as AccuWeather or The Weather Channel can provide real-time weather updates, helping users plan their journeys more safely, especially in adverse weather conditions.

Community-Sourced Traffic Apps: Applications like INRIX Traffic or community-driven platforms like Citizen enable users to report and receive real-time updates on traffic incidents, accidents, and road closures.

Car Safety Apps: Some car manufacturers provide companion apps that include safety features, such as remote vehicle monitoring, maintenance alerts, and safety notifications.

Personal Safety Apps: Applications like bSafe or My Safetipin focus on personal safety by allowing users to share their location, set check-ins, or send distress signals to predefined contacts.

In-Car Safety Systems: Modern vehicles often come with built-in safety features and assistance systems, such as lane departure warnings, collision detection, and automatic emergency braking.

Health and Fitness Apps: Applications like Strava or Google Fit may include features that track outdoor activities, providing safety insights for runners, cyclists, or pedestrians.

Application Architecture and UI.

The User Interface (UI) of an Android application is the visual and interactive aspect of the app that users interact with. It includes all the elements and components that users see and use on the screen. The primary purpose of the UI is to provide a user-friendly and aesthetically pleasing experience, facilitating effective communication between the user and the application. Here are some key aspects of the UI in an Android application and their purposes:

Layouts and Views: Purpose: To organize and structure the visual elements of the app.

Description: Layouts, such as LinearLayout, RelativeLayout, and ConstraintLayout, define the structure of the user interface. Views, such as TextViews, EditTexts, Buttons, and ImageViews, represent the various interactive and non-interactive elements displayed on the screen.

Navigation Components: Purpose: To facilitate navigation within the app.

Description: Navigation components, such as the Toolbar, TabLayout, and NavigationView, help users move between different sections or features of the app. They provide a clear and intuitive way for users to navigate the app's content.

Input Controls: Purpose: To capture user input.

Description: Input controls include EditTexts, Spinners, Checkboxes, RadioButtons, and other elements that allow users to input data or make selections. They are crucial for collecting information or preferences from users.

Feedback Mechanisms: Purpose: To provide feedback on user actions.

Description: Toast messages, Snackbars, and Dialogs are used to inform users about the success or failure of their actions. Providing feedback enhances the user experience by confirming that their interactions have been recognized.

Lists and Adapters: Purpose: To display lists of data. Description: RecyclerViews and ListViews are commonly used to present lists of items, such as contacts, messages, or products. Adapters are used to bind data to these views, ensuring dynamic and efficient content display.

Images and Multimedia: Purpose: To enhance visual appeal and convey information. Description: ImageViews are used to display images, while VideoViews are used for playing videos. Multimedia elements contribute to the overall aesthetics of the app and are often used to convey information or engage users.

Action Bars and Menus: Purpose: To provide access to app features and actions. Description: The ActionBar or Toolbar typically contains app-related actions and navigation options. Overflow menus and contextual menus provide additional options based on the user's context within the app.

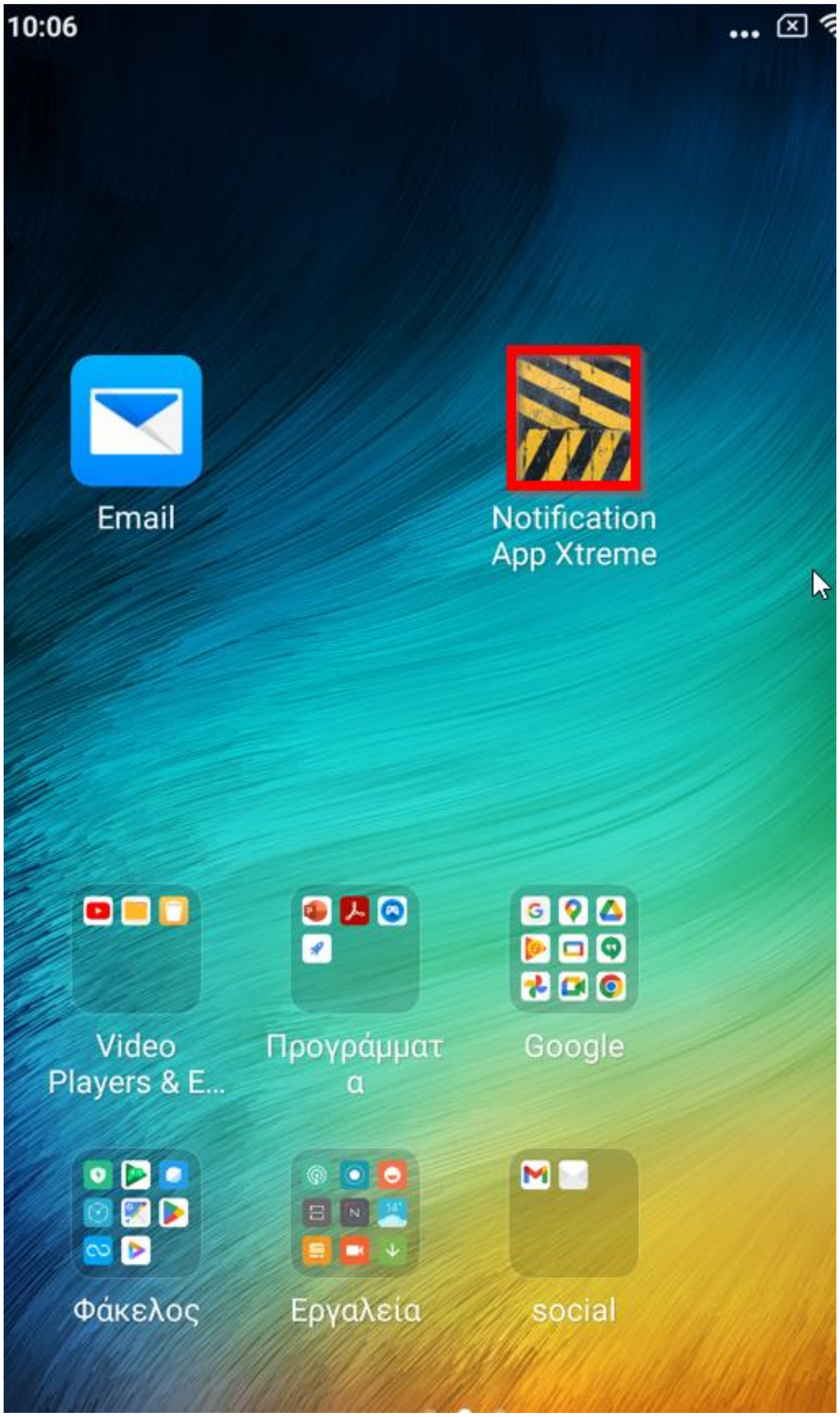
Animations and Transitions: Purpose: To create engaging and smooth user experiences. Description: Animations and transitions, such as fading, sliding, or scaling, can be used to provide visual feedback, guide users through changes in the UI, or simply enhance the overall aesthetic appeal of the app.

Accessibility Features: Purpose: To ensure inclusivity and usability for all users. Description: UI elements can be designed with accessibility features in mind, such as proper content descriptions, focus indicators, and compatibility with screen readers. This ensures that the app is usable by individuals with disabilities.

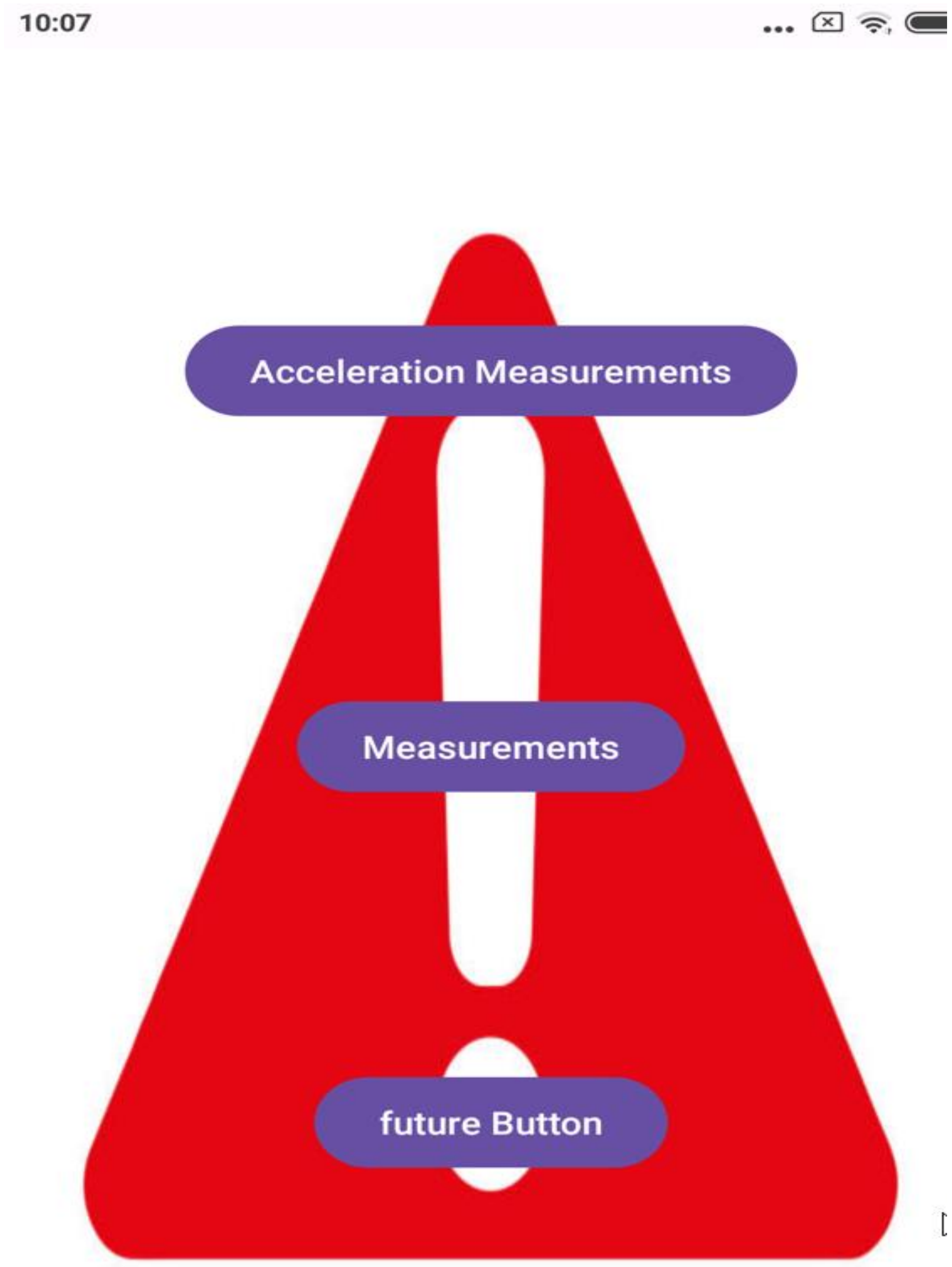
Themes and Styles: Purpose: To maintain a consistent and branded visual identity. Description: Themes and styles define the overall look and feel of the app, including colors, fonts, and other visual attributes. Consistent styling contributes to a cohesive and professional appearance.

In summary, the UI of an Android application serves as the user's gateway to interacting with the app. Its purpose is to provide a visually appealing, intuitive, and responsive environment that facilitates effective communication between the user and the application's functionalities. A well-designed UI enhances the overall user experience and contributes to the success and popularity of the app.

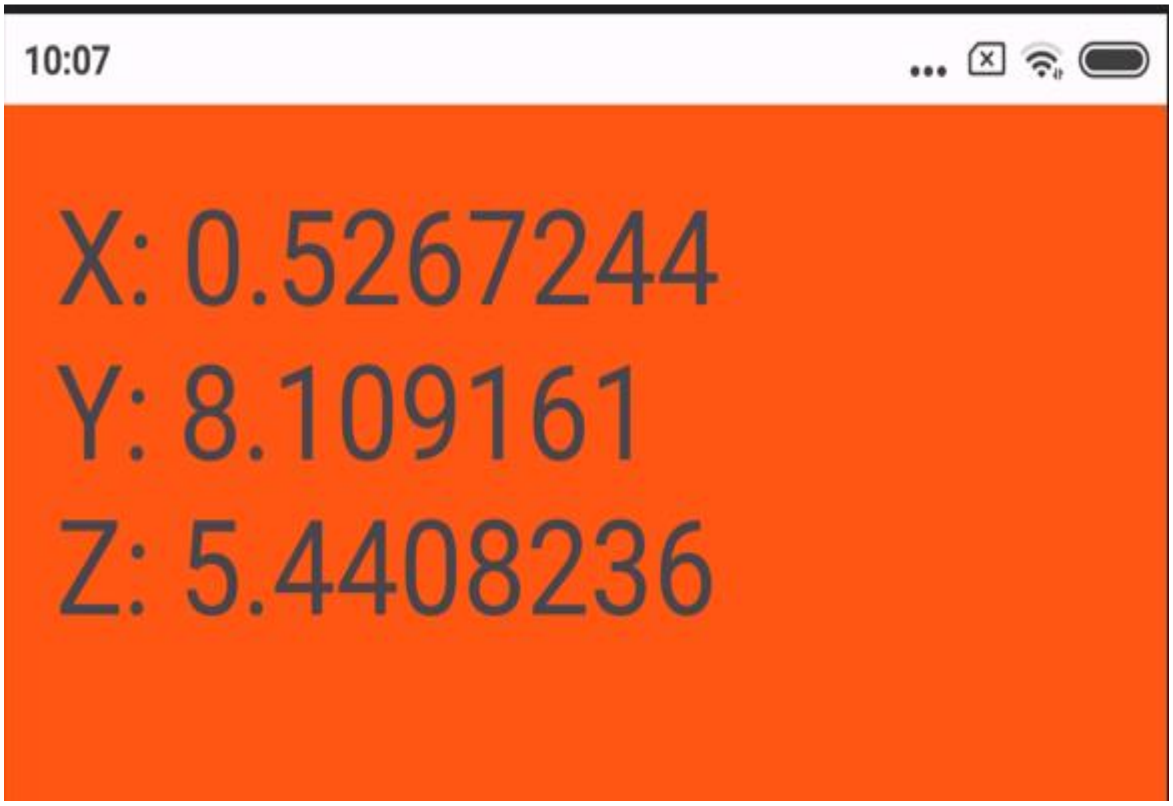
The NotificationApp Xtreme mobile application uses a unique icon with the international warning colors in order to be easy to select from other applications installed in the device .


10:06
Email
Notification App Xtreme
Video Players & E...
Προγράμματ α
Google
Φάκελος
Εργαλεία
social

A simple layout has been selected with limited buttons, only three, and no other selections for the main screen of the application. The first button “Acceletion Measurements” leeds to Activity2 .The second button “Measurements” leeds to Activity3.The trird and last button”future button” leeds to Activity4, which is a blank activity for future use in orde not to be changed the present UI with additional functionality.



The enviroment in Activity2 that the user faces is an easy to read and high contrast three line display of the triaxel values of the accelerometer measurements that the sensors of the device send in real time.Here we have to clarify that the constant acceleration of earth's gravity is equal to 9,81 m/sec*sec and it splits amoung the axis x,y,z as the device rotates.

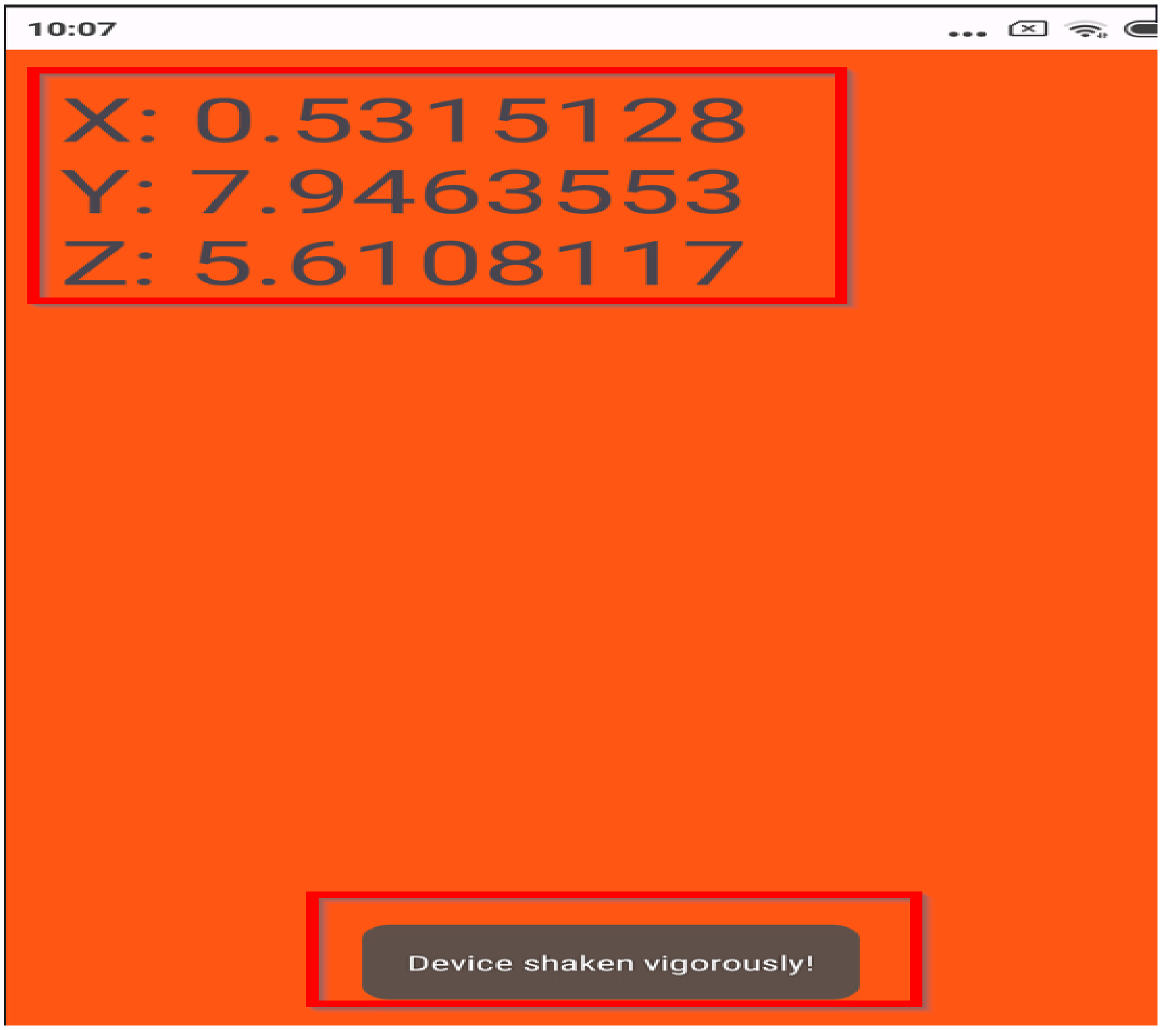

10:07

X: 0.5267244

Y: 8.109161

Z: 5.4408236

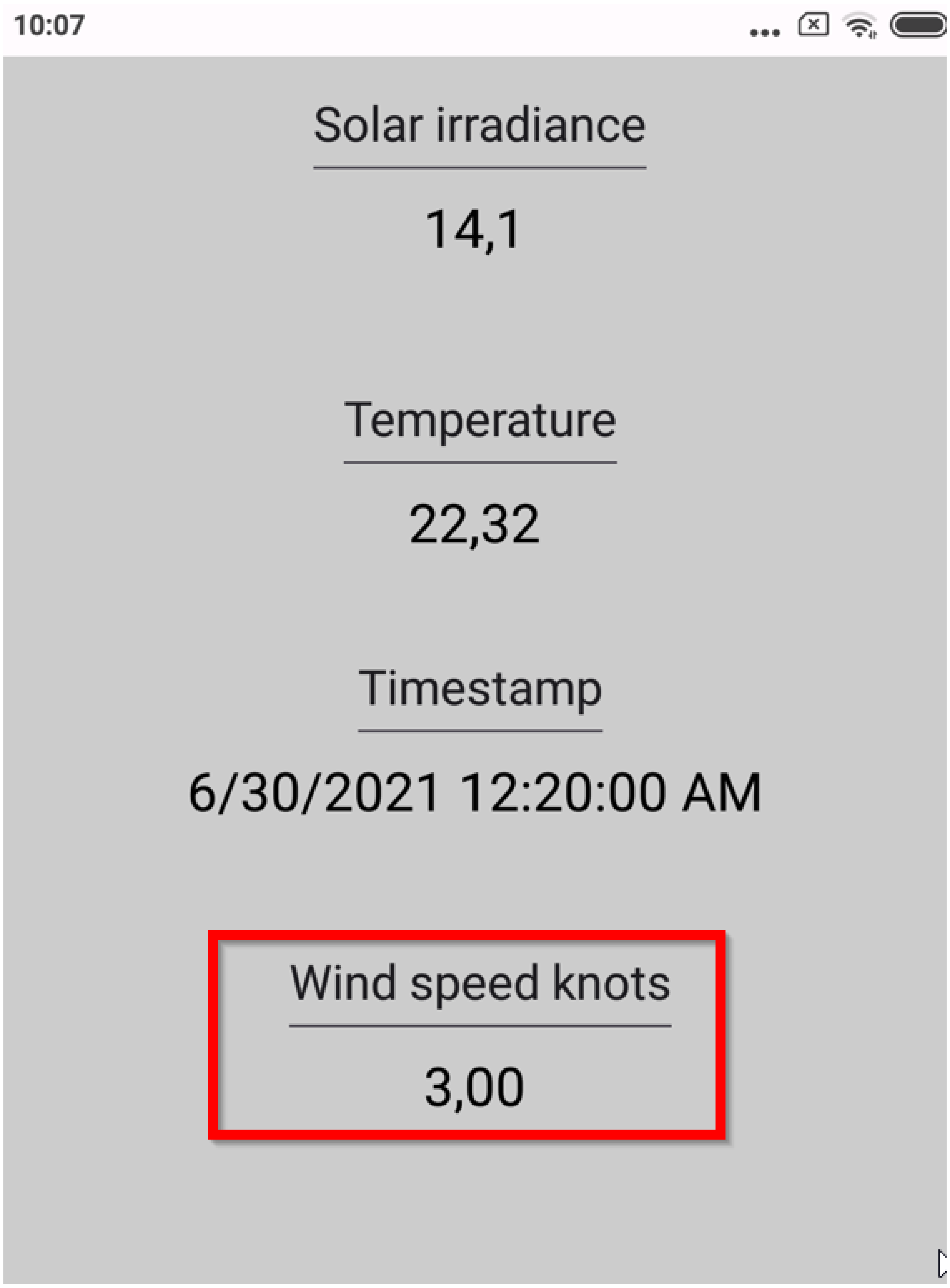
The live feed of measurements not only displaed in the screen but when any of the value of the three axis is greater than 15 the application displays a warning message "Device shaken vigorously" that can also can be sent via email to a predifined receiver as shown below.



X: 0.5315128
Y: 7.9463553
Z: 5.6108117

Device shaken vigorously!

The Ui in Activity3 demonstrates the “live feed” of a remote placed set of sensors. There we have set custom rules that when a value , for examle the Wind Speed is greater than “2” the devise activates the vibration for a short period of time and displays the message “Too windy weather, caution”. This Message also is sent to a predifined receiver via mail.



10:07

Solar irradiance

14,1

Temperature

22,32

Timestamp

6/30/2021 12:20:00 AM

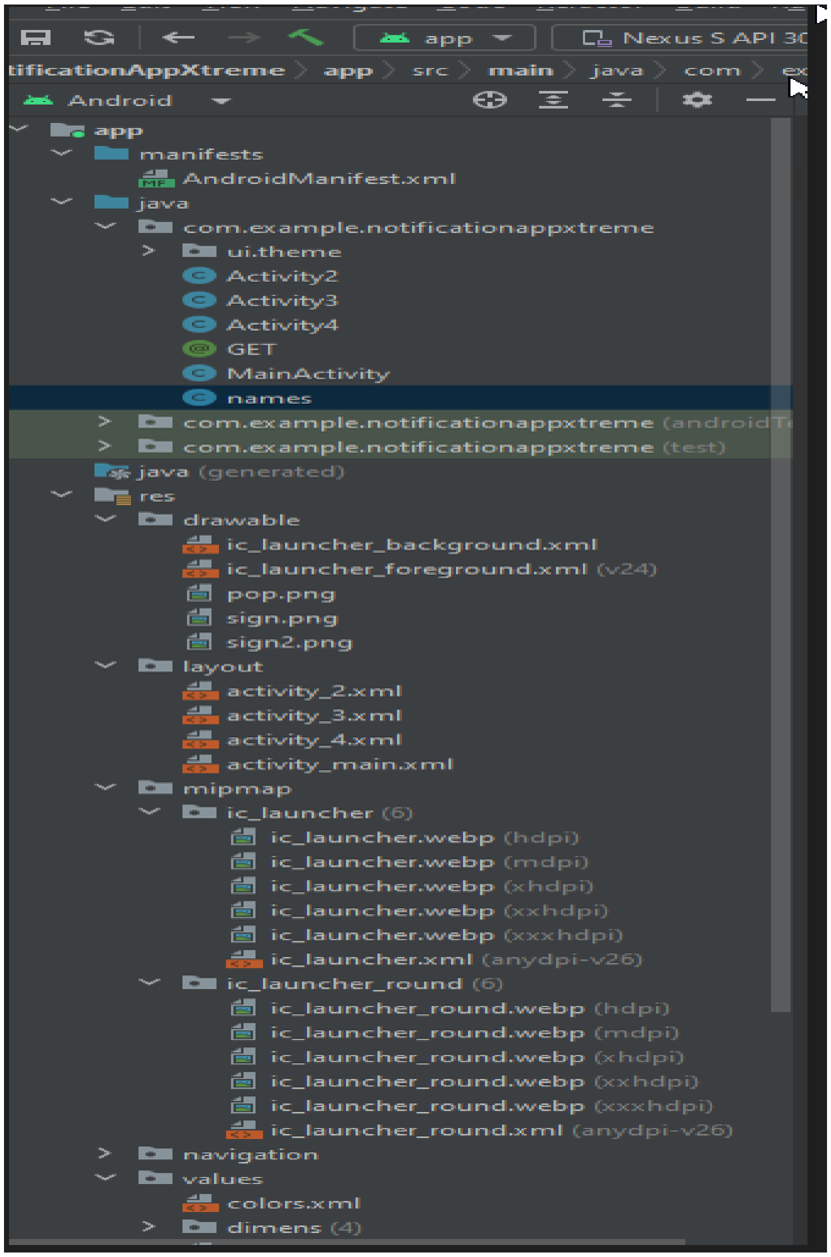
Wind speed knots

3,00

Android Studio set up.

Project tree menu Drill Down.

The below project tree represents the backbone of the mobile application with all the "activities" and the separate Java classes. An unused class has been created for future use (Activity 4) to keep the elements of the UI unchanged when the time comes.



```
app
  manifests
    AndroidManifest.xml
  java
    com.example.notificationappxtreme
      > ui.theme
        Activity2
        Activity3
        Activity4
        GET
        MainActivity
        names
    > com.example.notificationappxtreme (androidTe
    > com.example.notificationappxtreme (test)
  java (generated)
  res
    drawable
      ic_launcher_background.xml
      ic_launcher_foreground.xml (v24)
      pop.png
      sign.png
      sign2.png
    layout
      activity_2.xml
      activity_3.xml
      activity_4.xml
      activity_main.xml
    mipmap
      ic_launcher (6)
        ic_launcher.webp (hdpi)
        ic_launcher.webp (mdpi)
        ic_launcher.webp (xhdpi)
        ic_launcher.webp (xxhdpi)
        ic_launcher.webp (xxxhdpi)
        ic_launcher.xml (anydpi-v26)
      ic_launcher_round (6)
        ic_launcher_round.webp (hdpi)
        ic_launcher_round.webp (mdpi)
        ic_launcher_round.webp (xhdpi)
        ic_launcher_round.webp (xxhdpi)
        ic_launcher_round.webp (xxxhdpi)
        ic_launcher_round.xml (anydpi-v26)
    > navigation
    values
      colors.xml
      > dimens (4)
```

Manifest xml Drill Down.

The AndroidManifest.xml file is a crucial component of an Android application. It serves as a declaration file that provides essential information about the application to the Android operating system. The AndroidManifest.xml file is in the root directory of the Android project and is a necessary part of every Android app. Here are some key aspects of the AndroidManifest.xml file and its uses:

App Identity: The manifest file contains information about the app, such as its package name, version code, and version name. These details uniquely identify the application on the device and in the Google Play Store.

Permissions: Android requires explicit permission from users to access certain features or data on the device. The manifest file includes declarations for the permissions the app needs. For example, if an app needs access to the device's camera or internet, the corresponding permissions must be specified in the manifest.

Components Declaration: Activities, services, broadcast receivers, and content providers, which are essential building blocks of an Android app, are declared in the manifest. Each component is defined with an intent filter that describes the types of intents it can respond to.

Activity Configuration: Each activity in the app is declared in the manifest, along with information such as its label (displayed in the launcher), icon, theme, and other properties. Additionally, the manifest specifies the activities that serve as entry points into the app.

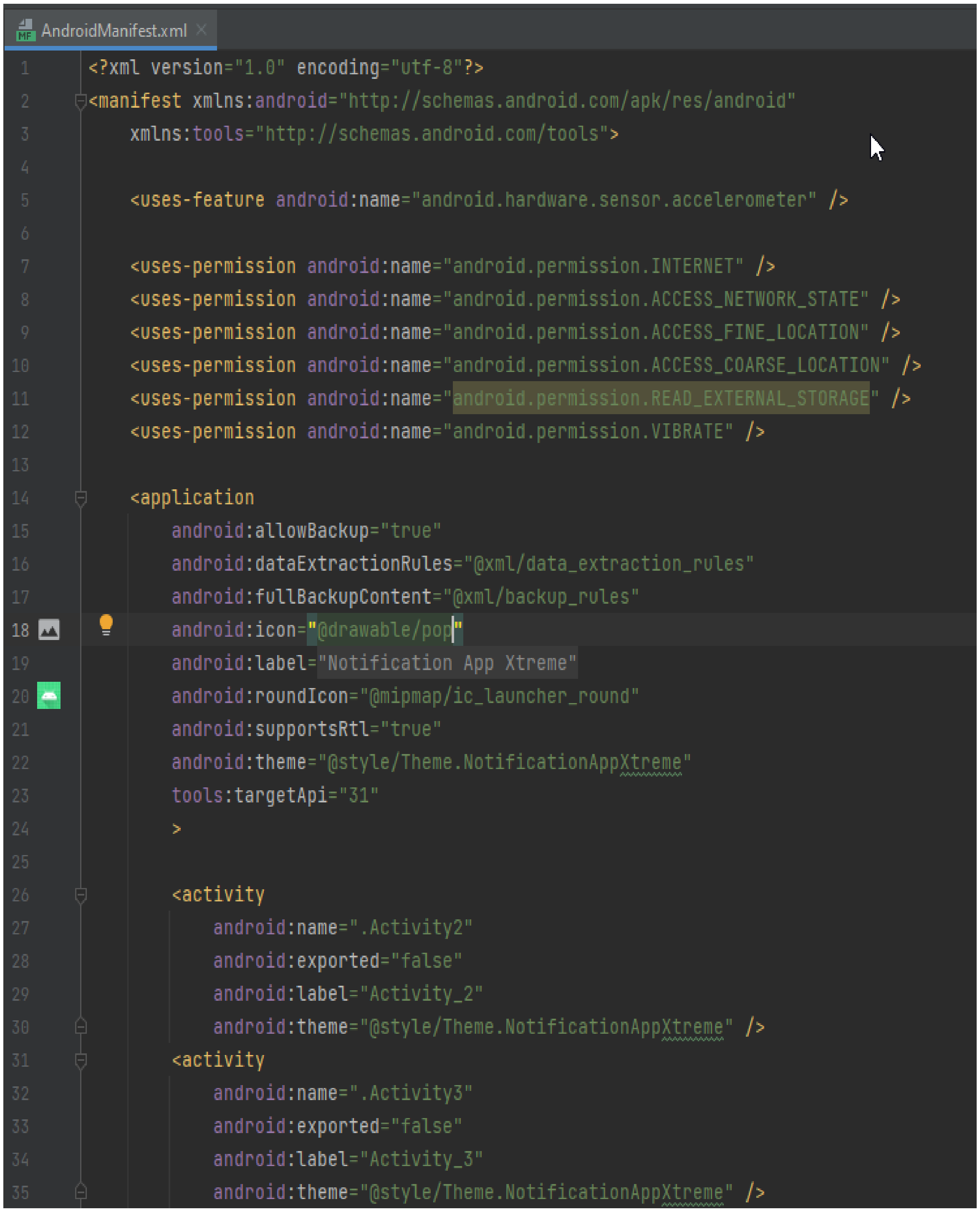
Intent Filters: Intent filters specify the types of intents an activity, service, or broadcast receiver can respond to. This is crucial for components to interact with each other and for handling implicit intents.

Application Theme and Icon: The manifest file specifies the theme for the entire application. Additionally, the launcher icon and label are declared here, determining how the app appears on the device's home screen.

Default Intent Filter: The manifest can declare a default intent filter for the entire application. This is used when an implicit intent doesn't match any specific activity's intent filter.

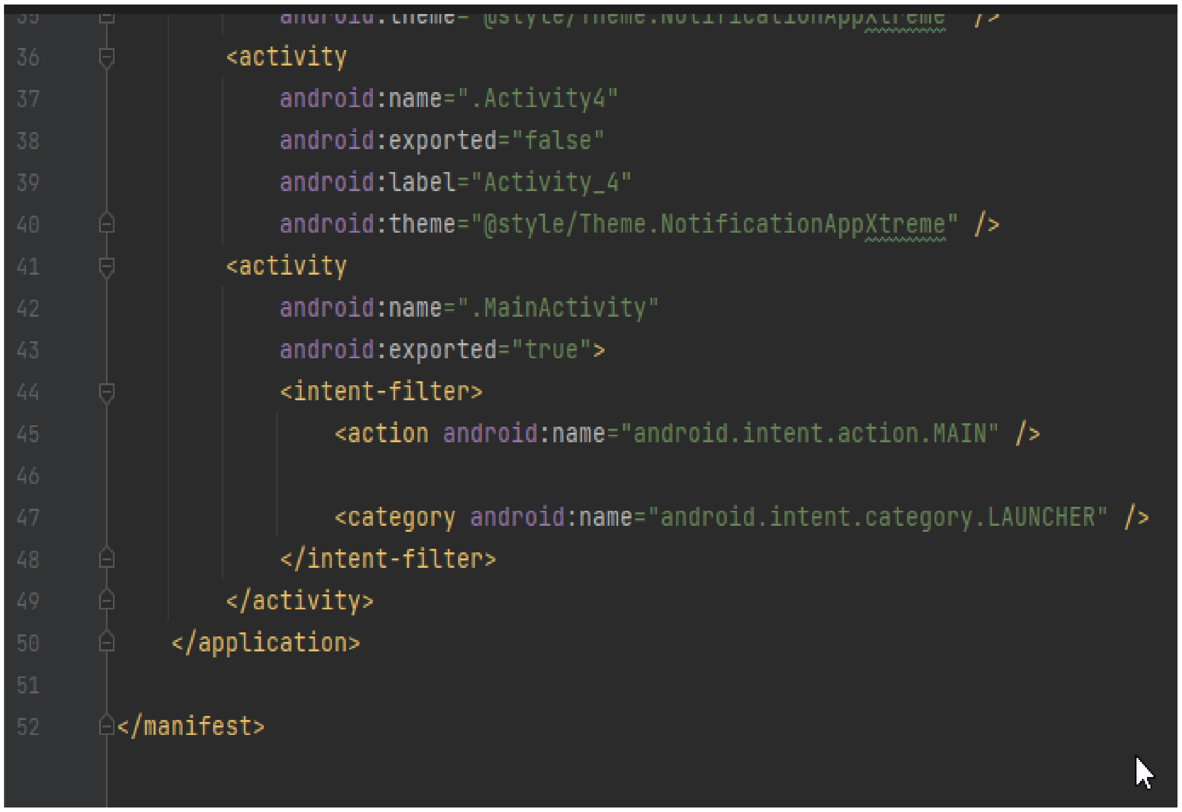
App Permissions and Security: The manifest file plays a crucial role in the security model of Android by declaring the necessary permissions. It ensures that apps have the required permissions to perform specific actions and protects user privacy.

The above attributes can be shown in the screenshots below.



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.sensor.accelerometer" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.VIBRATE" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@drawable/pop"
        android:label="Notification App Xtreme"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.NotificationAppXtreme"
        tools:targetApi="31"
        >

        <activity
            android:name=".Activity2"
            android:exported="false"
            android:label="Activity_2"
            android:theme="@style/Theme.NotificationAppXtreme" />
        <activity
            android:name=".Activity3"
            android:exported="false"
            android:label="Activity_3"
            android:theme="@style/Theme.NotificationAppXtreme" />
```



```
            android:theme="@style/Theme.NotificationAppXtreme" />
        <activity
            android:name=".Activity4"
            android:exported="false"
            android:label="Activity_4"
            android:theme="@style/Theme.NotificationAppXtreme" />
        <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>
```

Project Gradle Drill Down.

The project-level build.gradle file in an Android application is located in the root directory of the Android project. This file is distinct from the module-level build.gradle files, such as the one found in the app module. The project-level build.gradle file primarily configures settings that are applicable to the entire Android project. Here are some key aspects of the project-level build.gradle file and its uses:

Gradle Version: The build.gradle file for the project specifies the version of the Gradle build system to be used. This version is separate from the Android Gradle Plugin version used by the app modules.

Repositories: The repositories block in the project-level build.gradle file specifies the repositories from which Gradle should resolve dependencies. Common repositories include Google's Maven repository (google()) and JCenter (jcenter()).

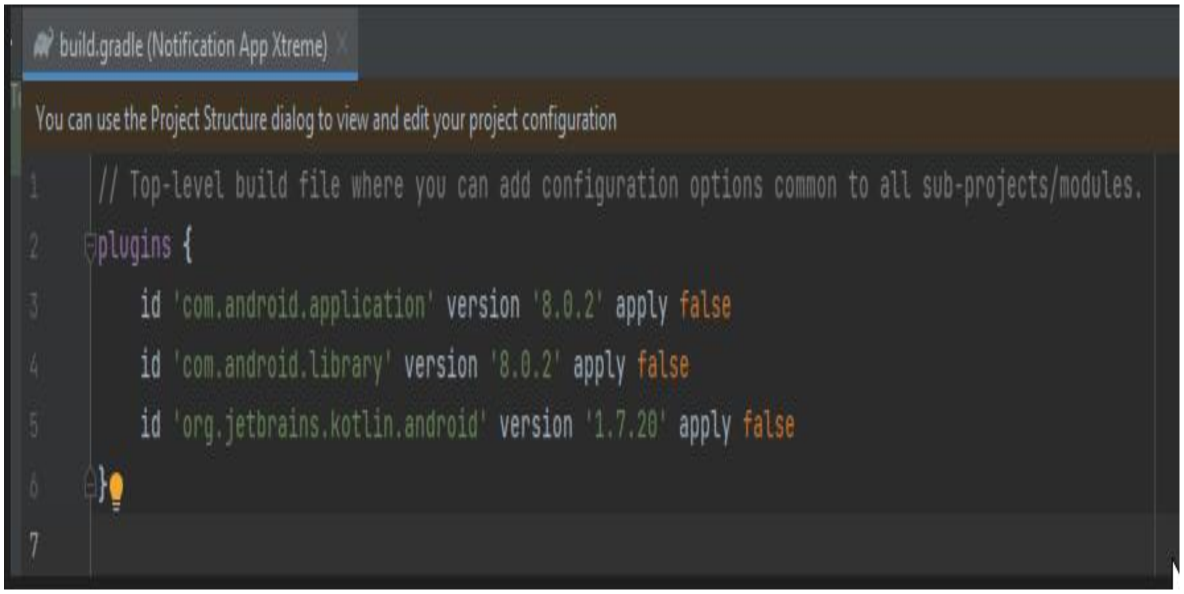
Gradle Plugin for Android: The build.gradle file configures the classpath for the Android Gradle Plugin. This plugin is responsible for various tasks related to building and packaging Android applications.

Top-level Dependencies: Dependencies that are common across multiple modules can be declared in the project-level build.gradle file. This is useful when certain libraries or dependencies are shared among different parts of the project.

Gradle Wrapper: The gradle-wrapper.properties file, often included in the project-level directory, defines the Gradle distribution used for building the project. This includes the Gradle version and distribution URL. The Gradle Wrapper allows the project to use a specific version of Gradle without requiring it to be installed separately.

Plugin Application: The apply plugin statement is used to apply plugins at the project level. For Android projects, the Android Gradle Plugin is applied, as well as any other plugins necessary for the overall project configuration.

The simplified version that uses the presented application is the below:


build.gradle (Notification App Xtreme)
You can use the Project Structure dialog to view and edit your project configuration

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id 'com.android.application' version '8.0.2' apply false
    id 'com.android.library' version '8.0.2' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.20' apply false
}
```

App Gradle Drill Down.

In Android development, the build.gradle file for the app module is a critical configuration file used with the Gradle build system. This file, often referred to as the "app-level build.gradle" or simply the "app build file," is located in the app module directory of an Android project. It specifies various settings and dependencies for building the Android application. Here are some key aspects of the build.gradle file for an Android app and its uses:

Android Plugin and SDK Versions: The build.gradle file specifies the Android Gradle Plugin version and the minimum and target SDK versions for the application. These settings ensure compatibility with different Android platform versions.

Dependencies: Dependencies on external libraries are declared in the dependencies block. This includes both the Android-specific dependencies (such as support libraries) and any third-party libraries required for the app.

Build Types and Product Flavors: The buildTypes and productFlavors blocks allow customization of the build process for different scenarios. For example, you can define different build types like "debug" and "release" with specific configurations.

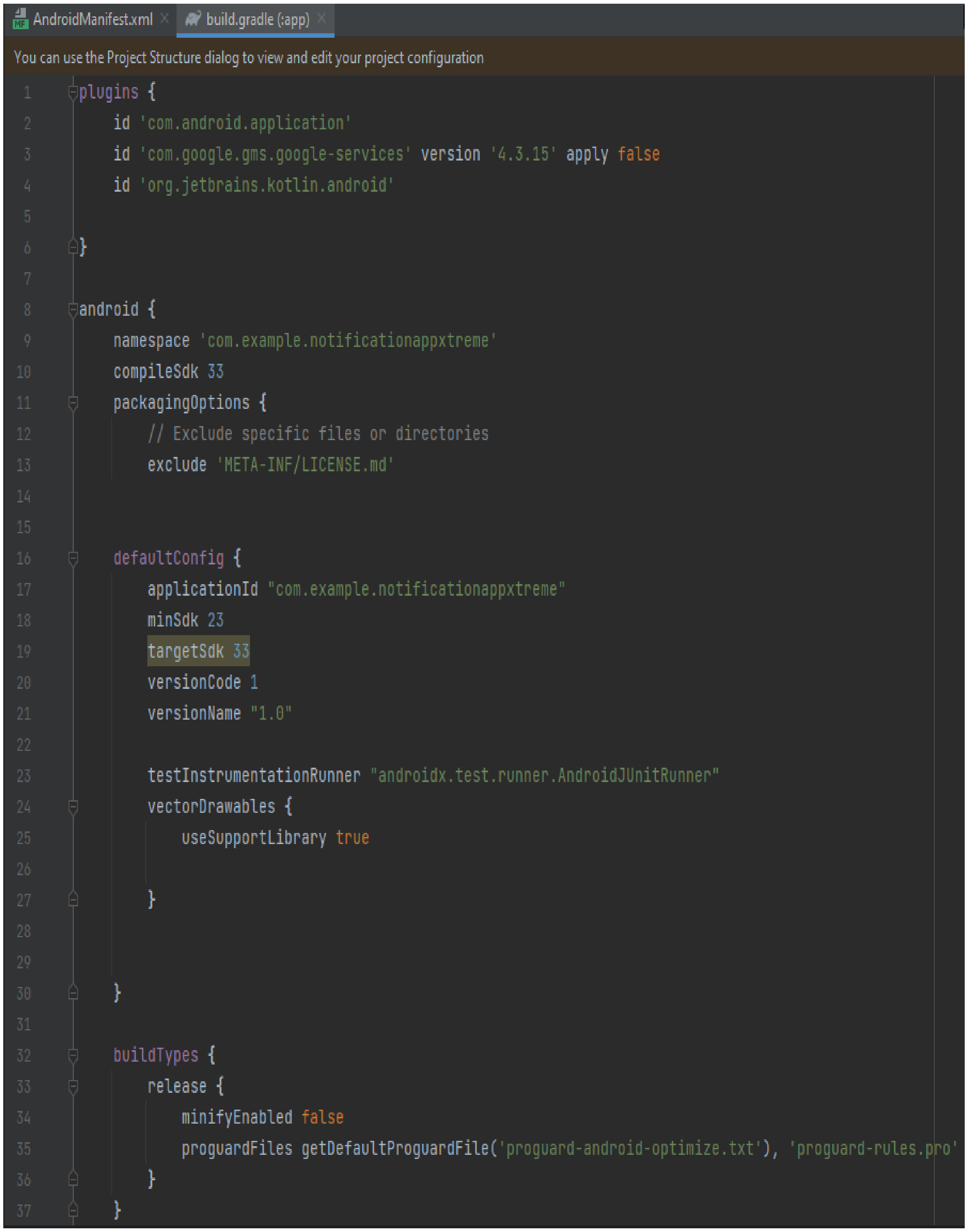
Other features that can be found in the buildgradle(app) file are the below:

Proguard Configuration: Proguard is a tool for code shrinking, obfuscation, and optimization. The build.gradle file may include Proguard configurations to specify rules for code obfuscation and optimization.

Signing Configurations: Signing configurations are specified for release builds to sign the APKs with a keystore. This is essential for publishing apps to the Google Play Store.

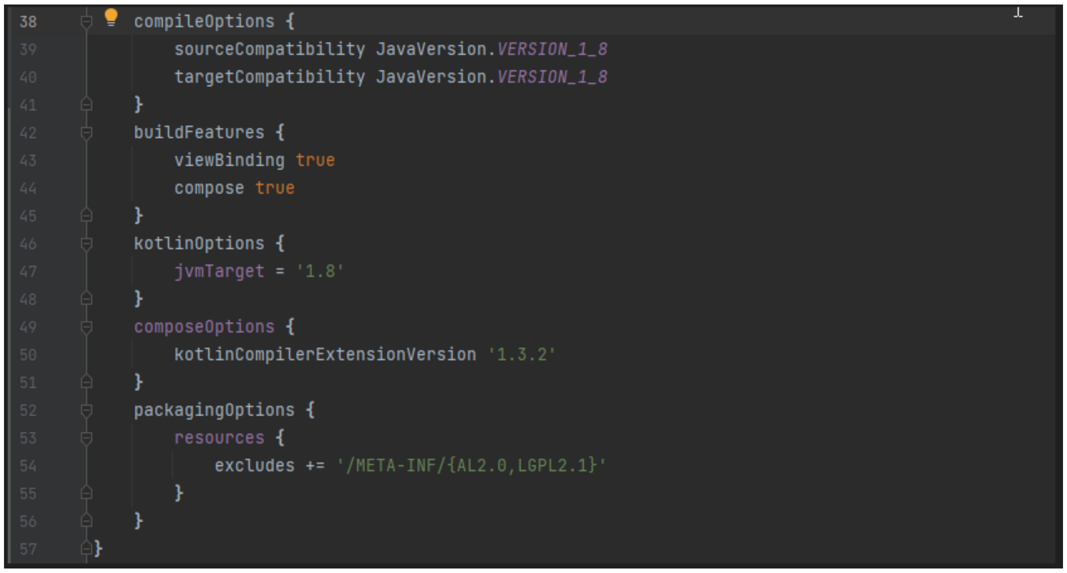
Lint Checks and Configuration: Lint checks are static code analysis tools provided by the Android Gradle Plugin. The lintOptions block allows configuration of lint checks, including enabling or disabling specific checks.

Custom Tasks and Scripts: The build.gradle file can include custom tasks and scripts to perform specific actions during the build process.

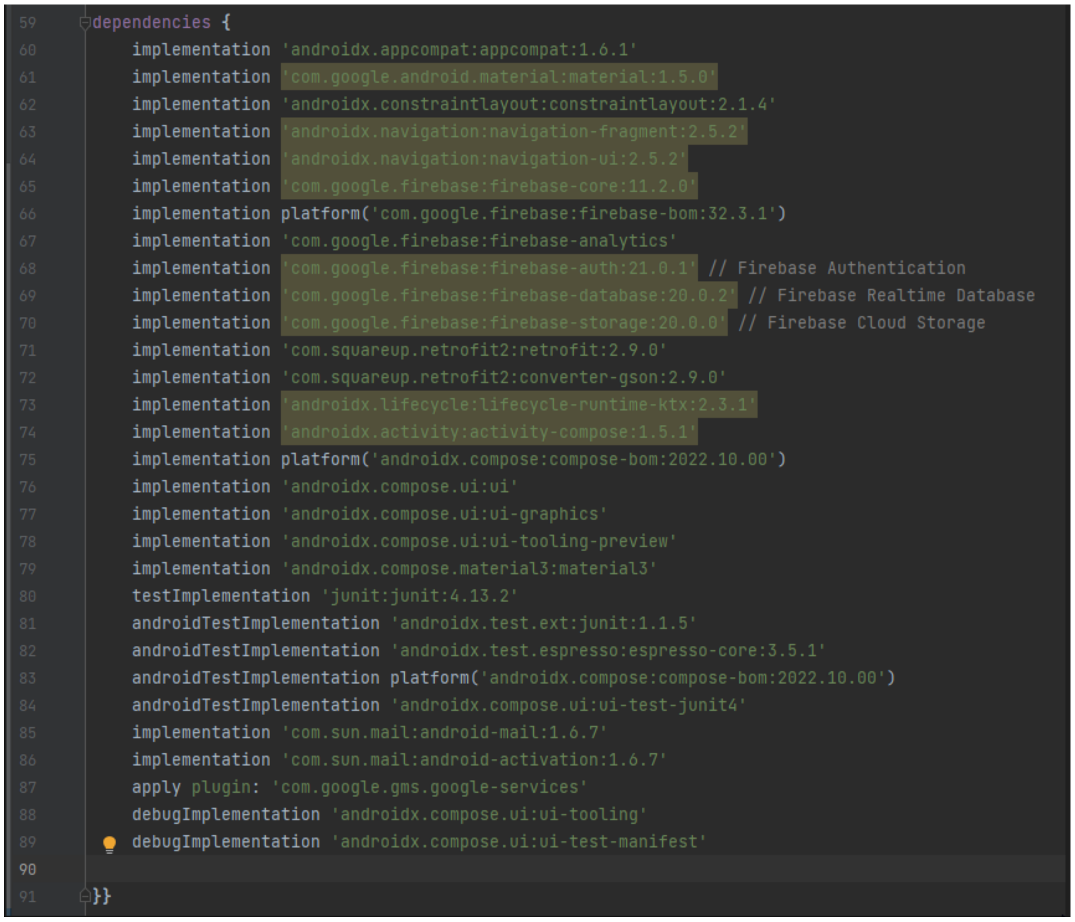


You can use the Project Structure dialog to view and edit your project configuration

```
plugins {
    id 'com.android.application'
    id 'com.google.gms.google-services' version '4.3.15' apply false
    id 'org.jetbrains.kotlin.android'

}

android {
    namespace 'com.example.notificationappxtreme'
    compileSdk 33
    packagingOptions {
        // Exclude specific files or directories
        exclude 'META-INF/LICENSE.md'


    defaultConfig {
        applicationId "com.example.notificationappxtreme"
        minSdk 23
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary true

        }


    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
```



```
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    buildFeatures {
        viewBinding true
        compose true
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    composeOptions {
        kotlinCompilerExtensionVersion '1.3.2'
    }
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}
```



```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'androidx.navigation:navigation-fragment:2.5.2'
    implementation 'androidx.navigation:navigation-ui:2.5.2'
    implementation 'com.google.firebase:firebase-core:11.2.0'
    implementation platform('com.google.firebase:firebase-bom:32.3.1')
    implementation 'com.google.firebase:firebase-analytics'
    implementation 'com.google.firebase:firebase-auth:21.0.1' // Firebase Authentication
    implementation 'com.google.firebase:firebase-database:20.0.2' // Firebase Realtime Database
    implementation 'com.google.firebase:firebase-storage:20.0.0' // Firebase Cloud Storage
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.5.1'
    implementation platform('androidx.compose:compose-bom:2022.10.00')
    implementation 'androidx.compose.ui:ui'
    implementation 'androidx.compose.ui:ui-graphics'
    implementation 'androidx.compose.ui:ui-tooling-preview'
    implementation 'androidx.compose.material3:material3'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation platform('androidx.compose:compose-bom:2022.10.00')
    androidTestImplementation 'androidx.compose.ui:ui-test-junit4'
    implementation 'com.sun.mail:android-mail:1.6.7'
    implementation 'com.sun.mail:android-activation:1.6.7'
    apply plugin: 'com.google.gms.google-services'
    debugImplementation 'androidx.compose.ui:ui-tooling'
    debugImplementation 'androidx.compose.ui:ui-test-manifest'

}}
```

Firebase dataset and set up.

In the rapidly evolving landscape of mobile app development, the synergy between Firebase and Android Studio stands as a testament to the power of seamless integration and robust tooling. Firebase, the comprehensive platform offered by Google, and Android Studio, the official integrated development environment (IDE) for Android app development, come together to create a dynamic duo that empowers developers to build feature-rich, scalable, and efficient applications.

The foundation of this collaboration lies in the ease of integration. Developers can initiate the process by creating a Firebase project through the Firebase Console, where they define their app's identity and configure the necessary services. Once the project is set up, integrating Firebase with an Android Studio app involves adding the required dependencies and configurations. This streamlined setup, often facilitated by the inclusion of a configuration file (google-services.json), ensures that developers can focus on building functionality rather than grappling with complex integration processes.

Authentication, a cornerstone of many modern applications, is significantly enhanced through Firebase Authentication. Android developers can seamlessly integrate methods such as email/password, Google Sign-In, and social media logins, providing a secure and user-friendly experience. Firebase Authentication simplifies the implementation of identity management, allowing developers to focus on building unique features rather than reinventing the wheel when it comes to user authentication.

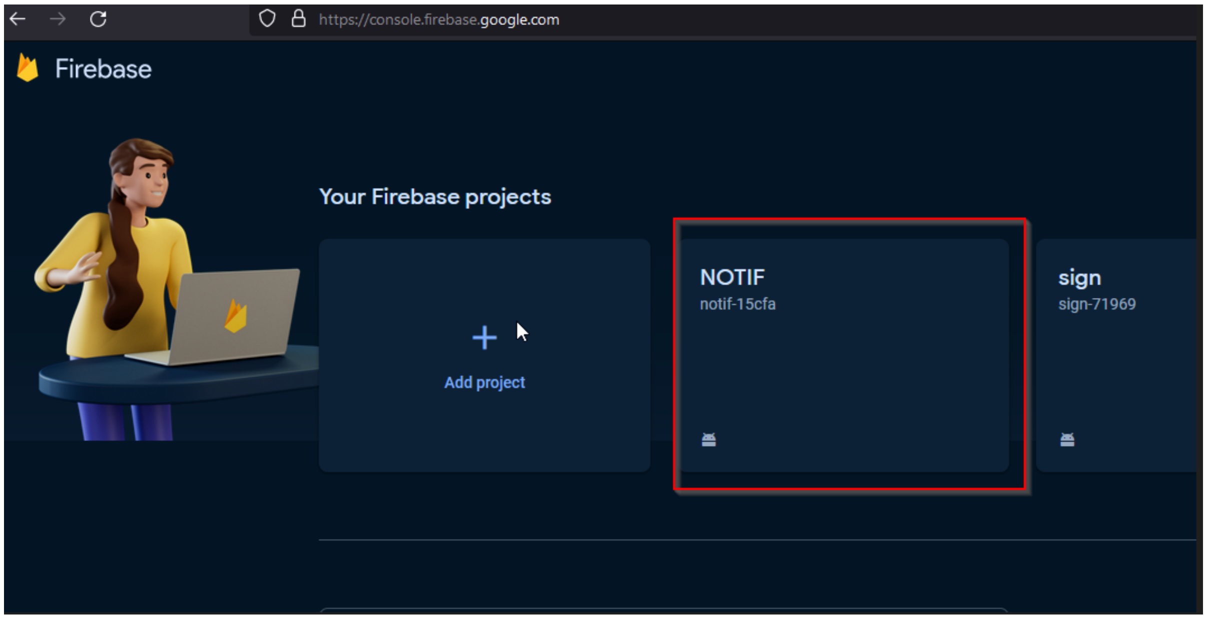
Firebase Realtime Database and Firestore extend collaboration into the realm of data management. These NoSQL cloud databases offer a scalable and real-time solution for storing and synchronizing data across multiple devices. Android Studio developers can leverage Firebase SDK to interact with the database, enabling real-time updates and ensuring that users experience the latest information without the need for manual refresh process. This synchronization capability is invaluable for applications requiring collaborative features, such as messaging or shared document editing.

The collaboration also extends to Firebase Cloud Functions, enabling developers to run backend code in response to events triggered by Firebase features. This serverless architecture enhances the scalability and responsiveness of Android apps by offloading computation-intensive tasks to the cloud. Whether handling authentication events, database triggers, or custom business logic, Cloud Functions seamlessly integrate with the Android Studio development workflow.

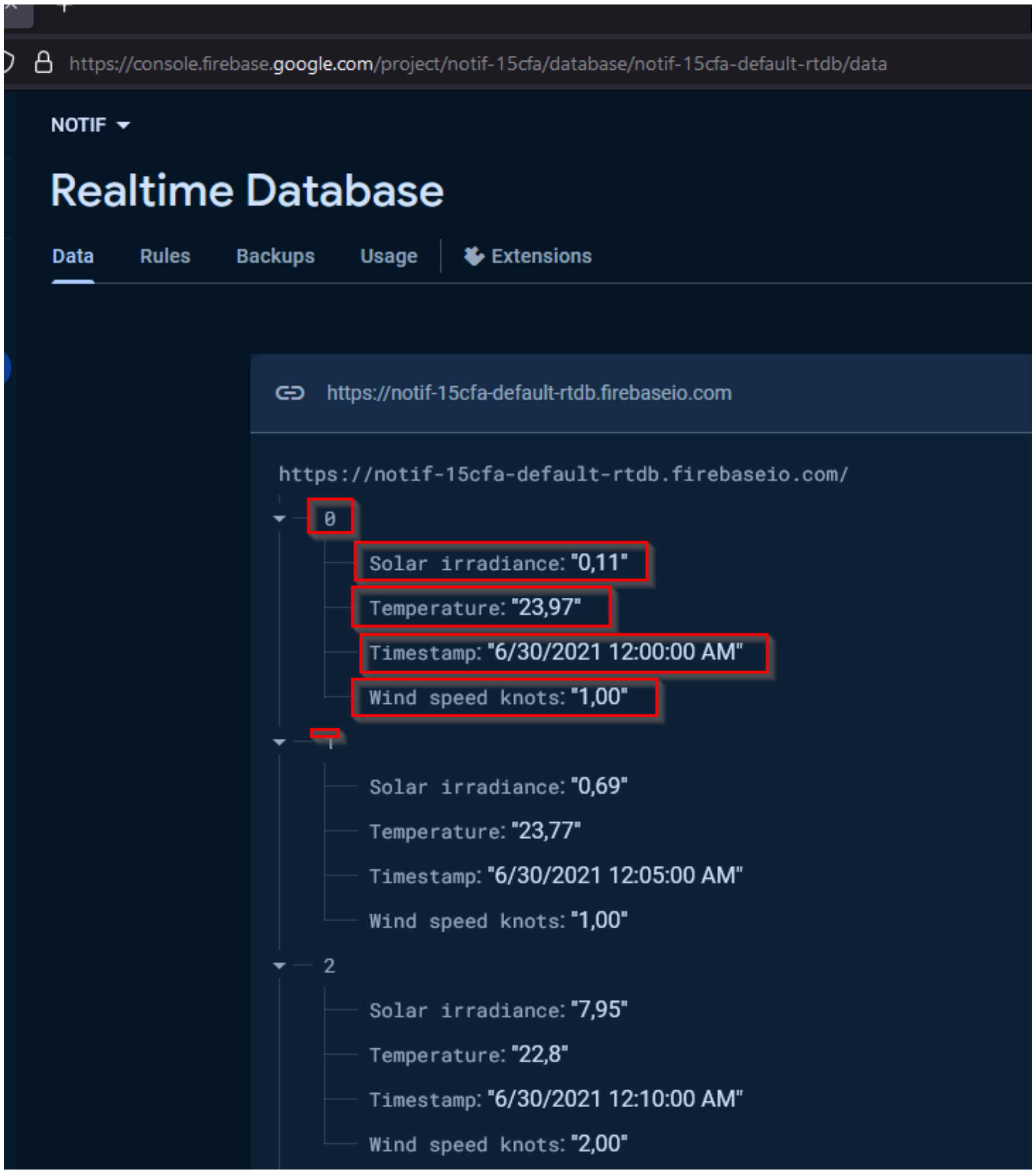
Firebase Cloud Storage further enriches collaboration by providing a secure and scalable solution for handling user-generated content such as images, videos, and other files. Android developers can seamlessly integrate Cloud Storage into their apps, ensuring a smooth and efficient experience for users when uploading or downloading files.

Firebase Analytics, Remote Config, Dynamic Links, and other services contribute to the collaboration by providing valuable insights into user behavior, tailoring app behavior remotely, creating deep links that survive app installations, and more. Android Studio developers gain a comprehensive set of tools that not only streamline development but also contribute to the ongoing success and improvement of their applications.

The below architecture Is used for the needs of the present thesis.



A project with title "NOTIF" created in the Firebase console.


https://console.firebase.google.com/project/notif-15cfa/database/notif-15cfa-default-rtdb/data
NOTIF
Realtime Database
Data
Rules
Backups
Usage
Extensions
https://notif-15cfa-default-rtdb.firebaseio.com

```
https://notif-15cfa-default-rtdb.firebaseio.com/
0
    Solar irradiance: "0,11"
    Temperature: "23,97"
    Timestamp: "6/30/2021 12:00:00 AM"
    Wind speed knots: "1,00"
1
    Solar irradiance: "0,69"
    Temperature: "23,77"
    Timestamp: "6/30/2021 12:05:00 AM"
    Wind speed knots: "1,00"
2
    Solar irradiance: "7,95"
    Temperature: "22,8"
    Timestamp: "6/30/2021 12:10:00 AM"
    Wind speed knots: "2,00"
```

The data that the specific dataset are categorized in four main types:

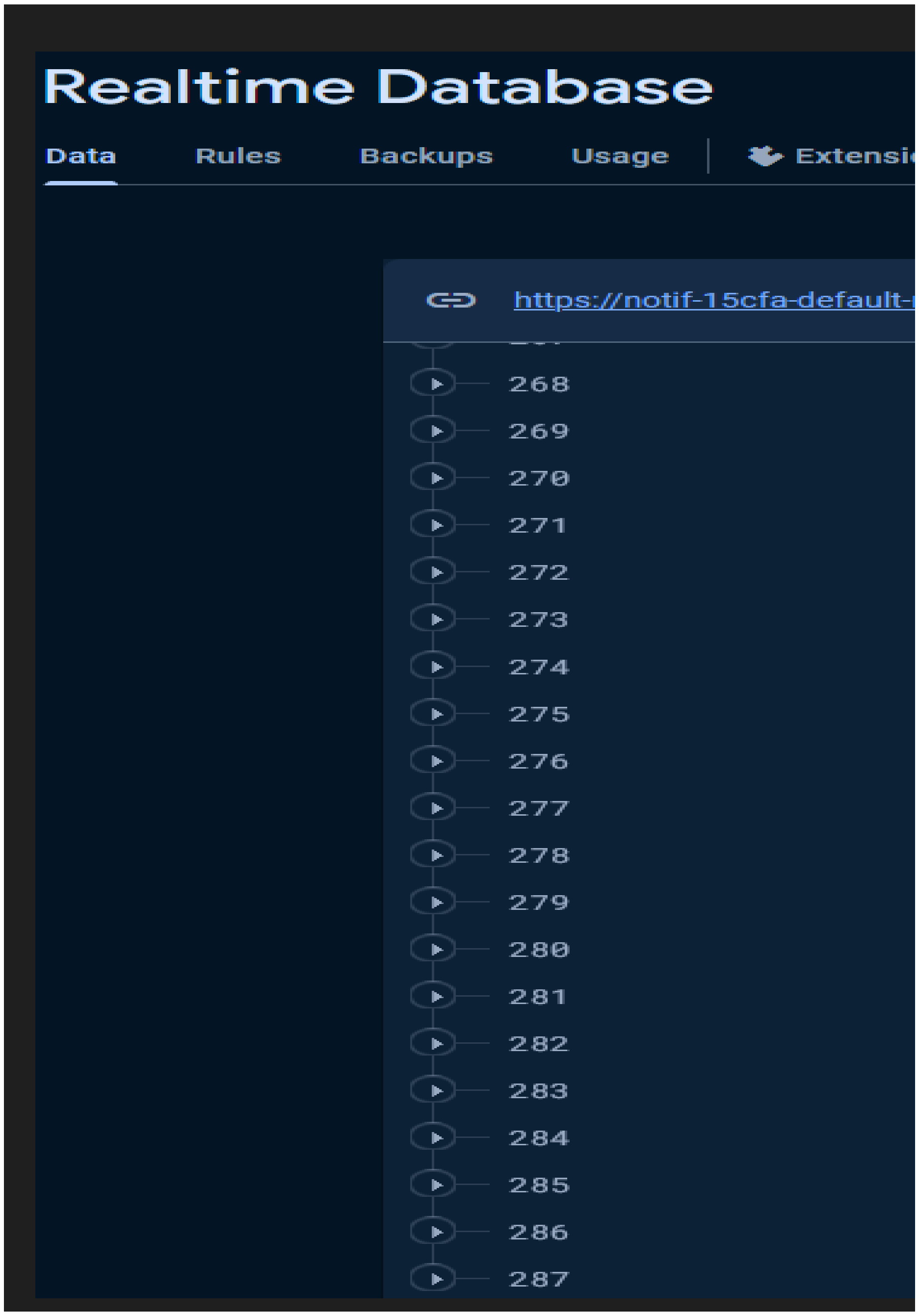
Solar Irradiance : Refers to the amount of solar power received per unit area at a given location on Earth's surface. It is a measure of the solar energy flux received from the Sun per unit of horizontal surface area and is usually expressed in units of watts per square meter (W/m²).

Temrature : Is a measure of the average kinetic energy of the particles in a substance, typically measured in degrees Celsius (°C), in this imlementation, or Fahrenheit (°F). In scientific terms, it is a scalar quantity that represents the thermal energy of an object or a system.

Timestamp: The date/time that the specific data collected.

Wind speed in knots: A knot is a unit of speed commonly used in aviation and maritime contexts. One knot is equal to one nautical mile per hour.

In order to have decreete instances for the need of the application each quadruplet has a unique incremented numbering from 0 to 287.



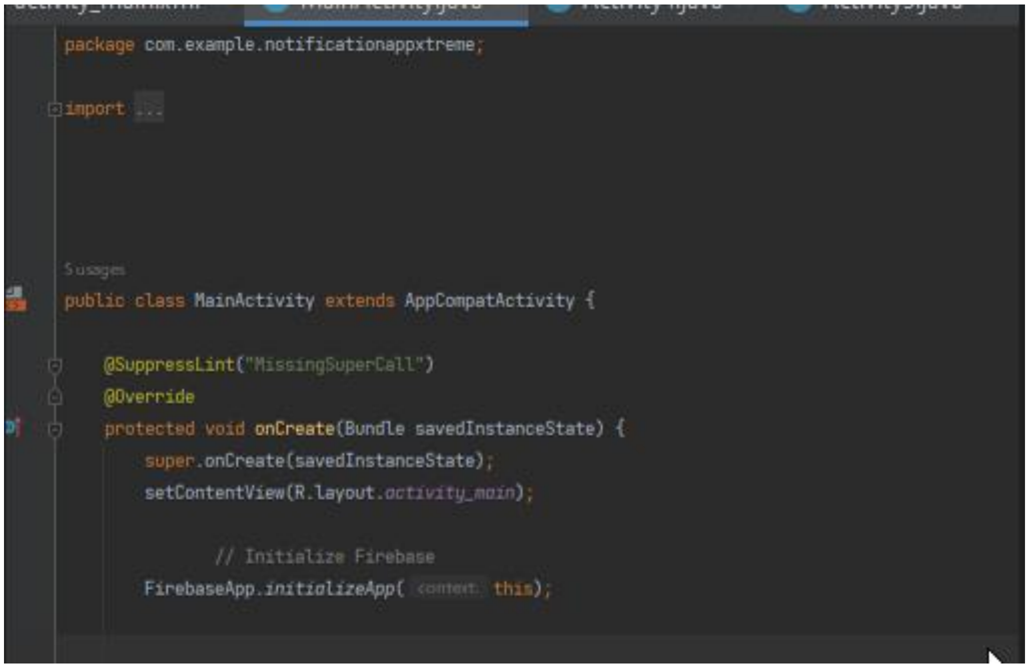
Data feed as time string concept.

The presented mobile application can be used for real time prevention and early noticing for accidents, if it is combined with real time data from sensors. For the needs of the thesis the role of the real time data undertakes a dataset from Firebase, that contains all the necessary information. The timestamp of the data is real and taken every 5 minutes from a set of sensors placed in a solar panel in the region of central Greece. To have consecutive observations the five-minute intervals that the data set contains are fed to the application as a different set of observations every second.

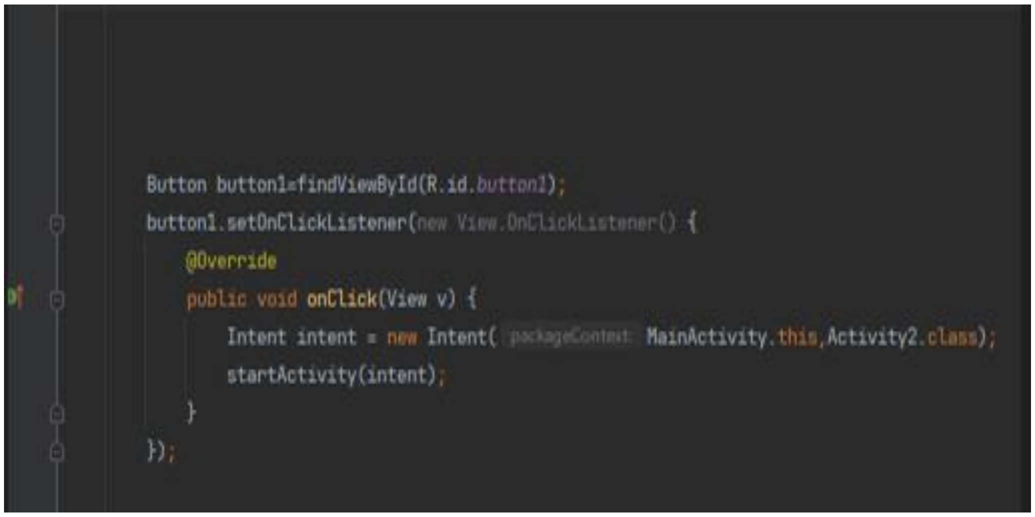
Code documentation.

Documentation for Main Activity Class

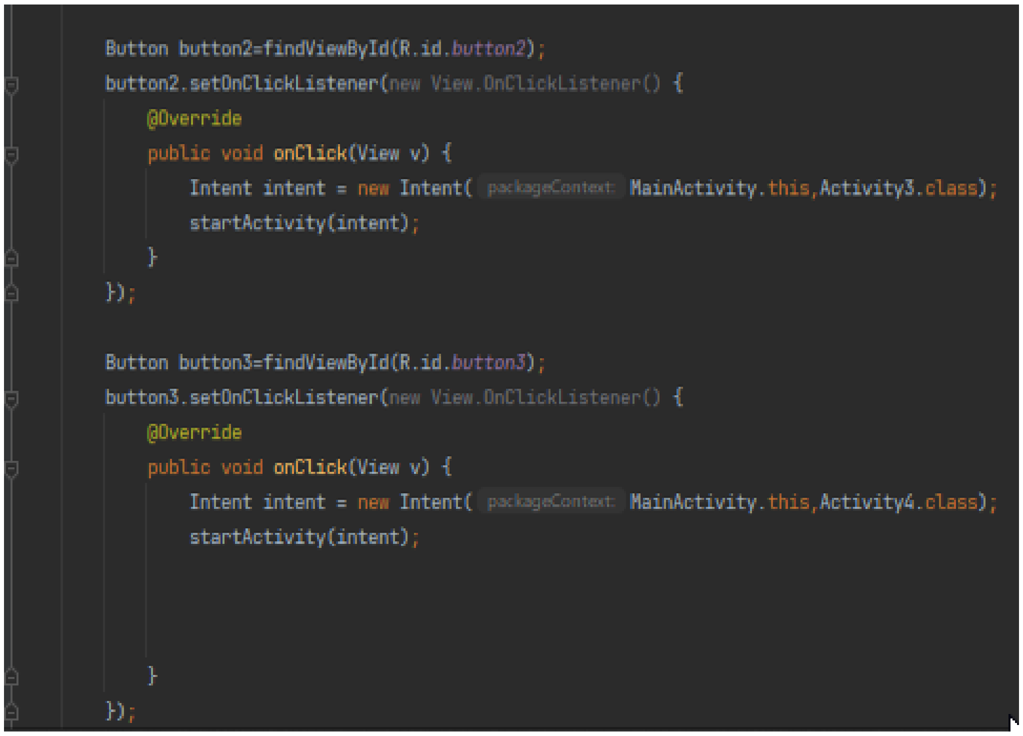
Code implemented in Main Activity.



```java
package com.example.notificationappxtreme;

import ...




5 usages
public class MainActivity extends AppCompatActivity {

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

                // Initialize Firebase
        FirebaseApp.initializeApp( context: this);
```



```
Button button1=findViewById(R.id.button1);
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: MainActivity.this,Activity2.class);
        startActivity(intent);
    }
});
```



```
Button button2=findViewById(R.id.button2);
button2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: MainActivity.this,Activity3.class);
        startActivity(intent);
    }
});

Button button3=findViewById(R.id.button3);
button3.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: MainActivity.this,Activity4.class);
        startActivity(intent);



    }
});
```

Code Interpretation for Main Activity.

Class Declaration and Initialization:

This class extends AppCompatActivity, indicating that it is an activity within an Android application.

onCreate Method:

Overrides the onCreate method, which is called when the activity is first created.Sets the content view to the layout defined in activity_main.xml. Initializes Firebase for the application. Configures click listeners for three buttons (button1, button2, button3), each launching a different activity when clicked.

Button Click Listeners:

Button1 Click:Creates an Intent to navigate from MainActivity to Activity2 when button1 is clicked.

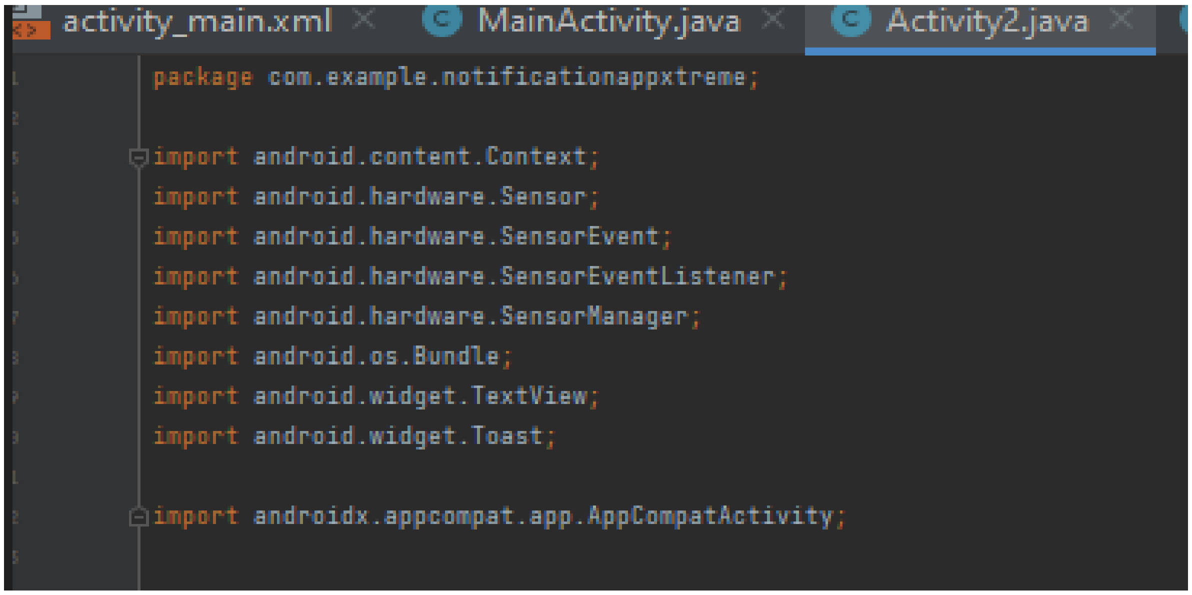
Button2 Click:Creates an Intent to navigate from MainActivity to Activity3 when button2 is clicked.

Button3 Click: Creates an Intent to navigate from MainActivity to Activity4 when button3 is clicked.

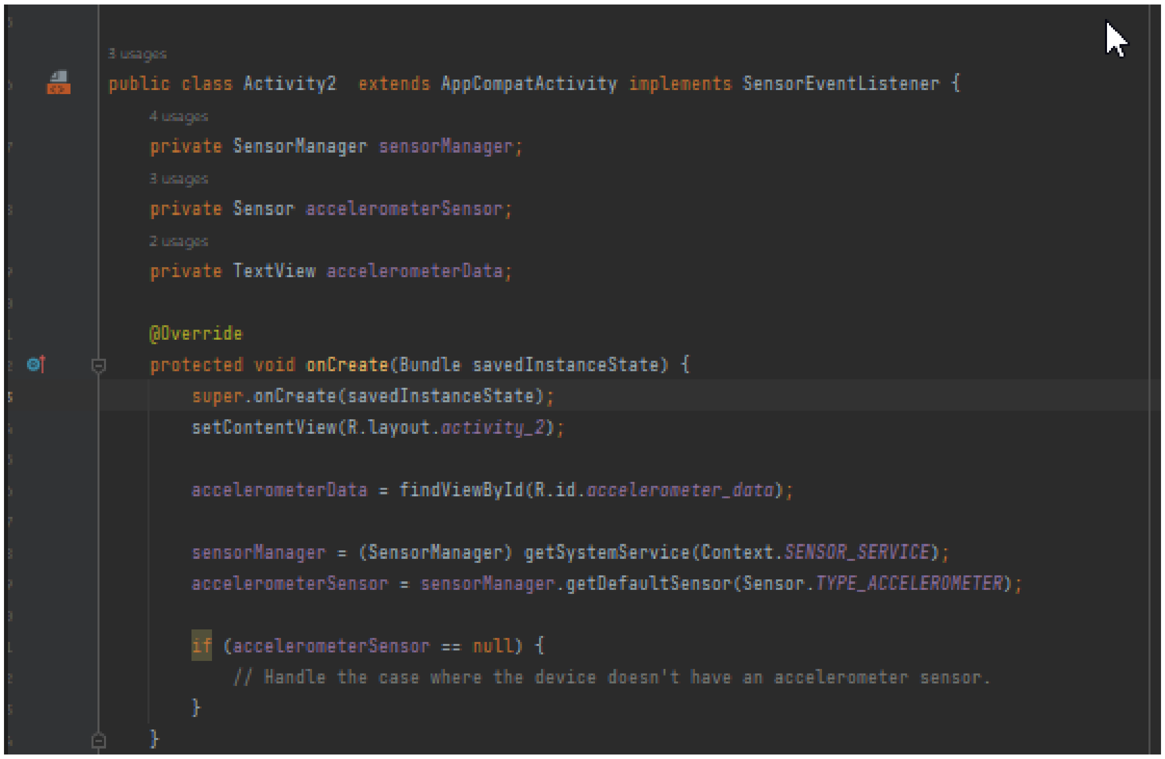
MainActivity sets up the user interface from activity_main.xml, initializes Firebase, and defines click listeners for three buttons. Each button click launches a different activity, enabling navigation within the application.

Documentation for Activity2 Class

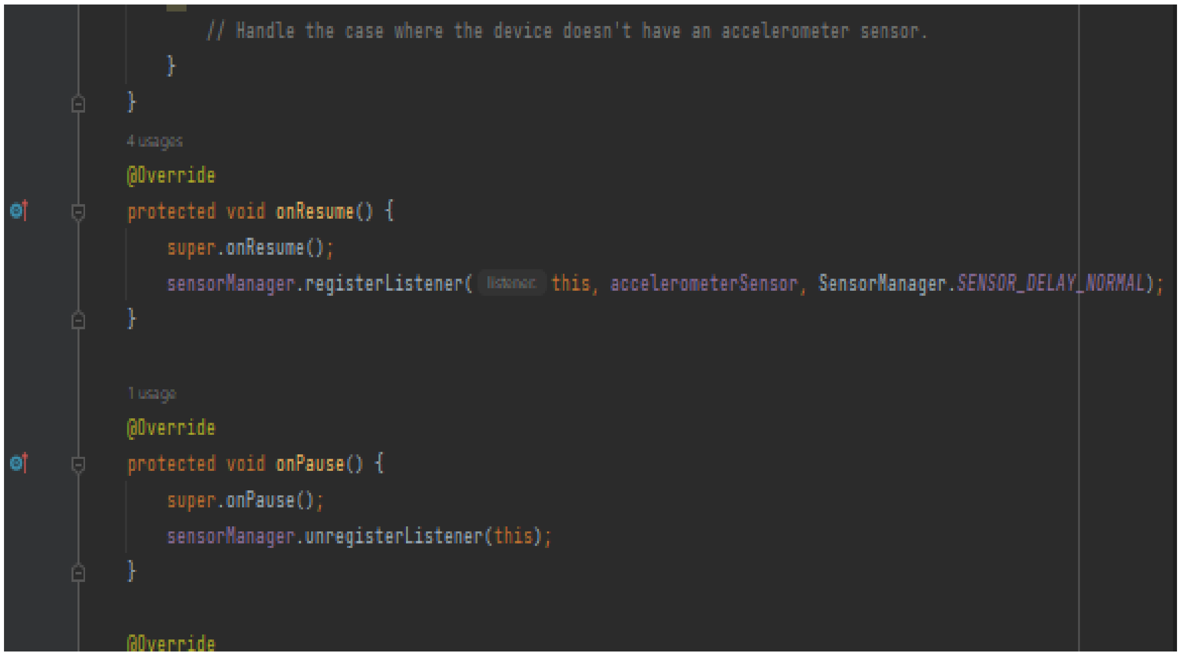
Code implemented in Activity2.



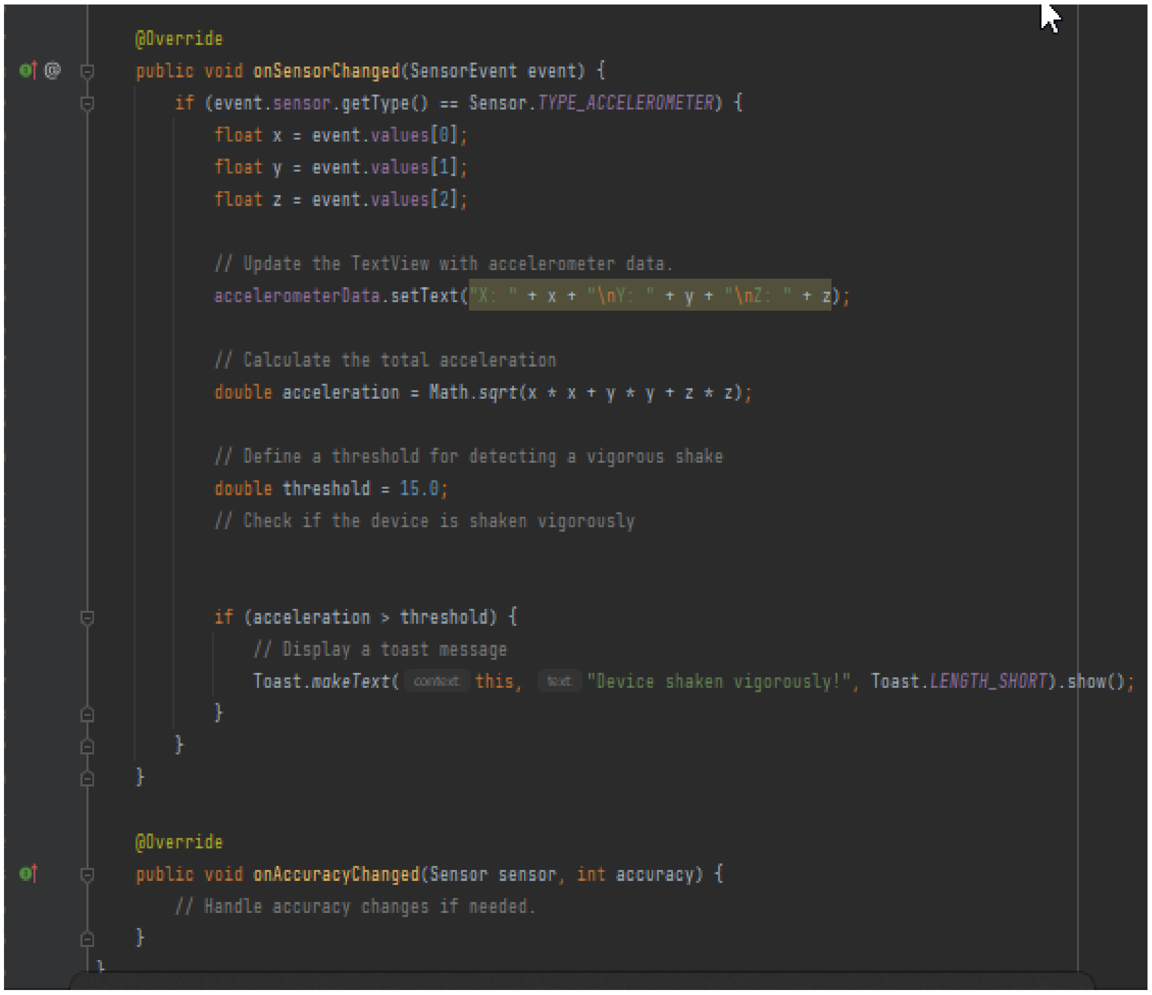
```java
package com.example.notificationappxtreme;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
```



```
3 usages
public class Activity2  extends AppCompatActivity implements SensorEventListener {
    4 usages
    private SensorManager sensorManager;
    3 usages
    private Sensor accelerometerSensor;
    2 usages
    private TextView accelerometerData;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_2);

        accelerometerData = findViewById(R.id.accelerometer_data);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        if (accelerometerSensor == null) {
            // Handle the case where the device doesn't have an accelerometer sensor.
        }
    }
```



```
            // Handle the case where the device doesn't have an accelerometer sensor.
        }
    }
    4 usages
    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener( listener: this, accelerometerSensor, SensorManager.SENSOR_DELAY_NORMAL);
    }

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

    @Override
```



```
    @Override
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            float x = event.values[0];
            float y = event.values[1];
            float z = event.values[2];

            // Update the TextView with accelerometer data.
            accelerometerData.setText("X: " + x + "\nY: " + y + "\nZ: " + z);

            // Calculate the total acceleration
            double acceleration = Math.sqrt(x * x + y * y + z * z);

            // Define a threshold for detecting a vigorous shake
            double threshold = 15.0;
            // Check if the device is shaken vigorously


            if (acceleration > threshold) {
                // Display a toast message
                Toast.makeText( context: this, text: "Device shaken vigorously!", Toast.LENGTH_SHORT).show();
            }
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Handle accuracy changes if needed.
    }
}
```

Code Interpretation for Activity2.

Class Declaration and Fields:The class extends AppCompatActivity and implements SensorEventListener to listen for sensor events.It has fields for SensorManager, accelerometerSensor, and Text View to display accelerometer data.

onCreate Method: Initializes the layout and views.

Retrieves the accelerometer sensor from the system service.Checks if the device has an accelerometer sensor and handles the case where it doesn't.

Sensor Registration: Registers and unregisters the sensor listener when the activity is resumed and paused, respectively.

onSensorChanged Method:Monitors changes in the accelerometer sensor data.

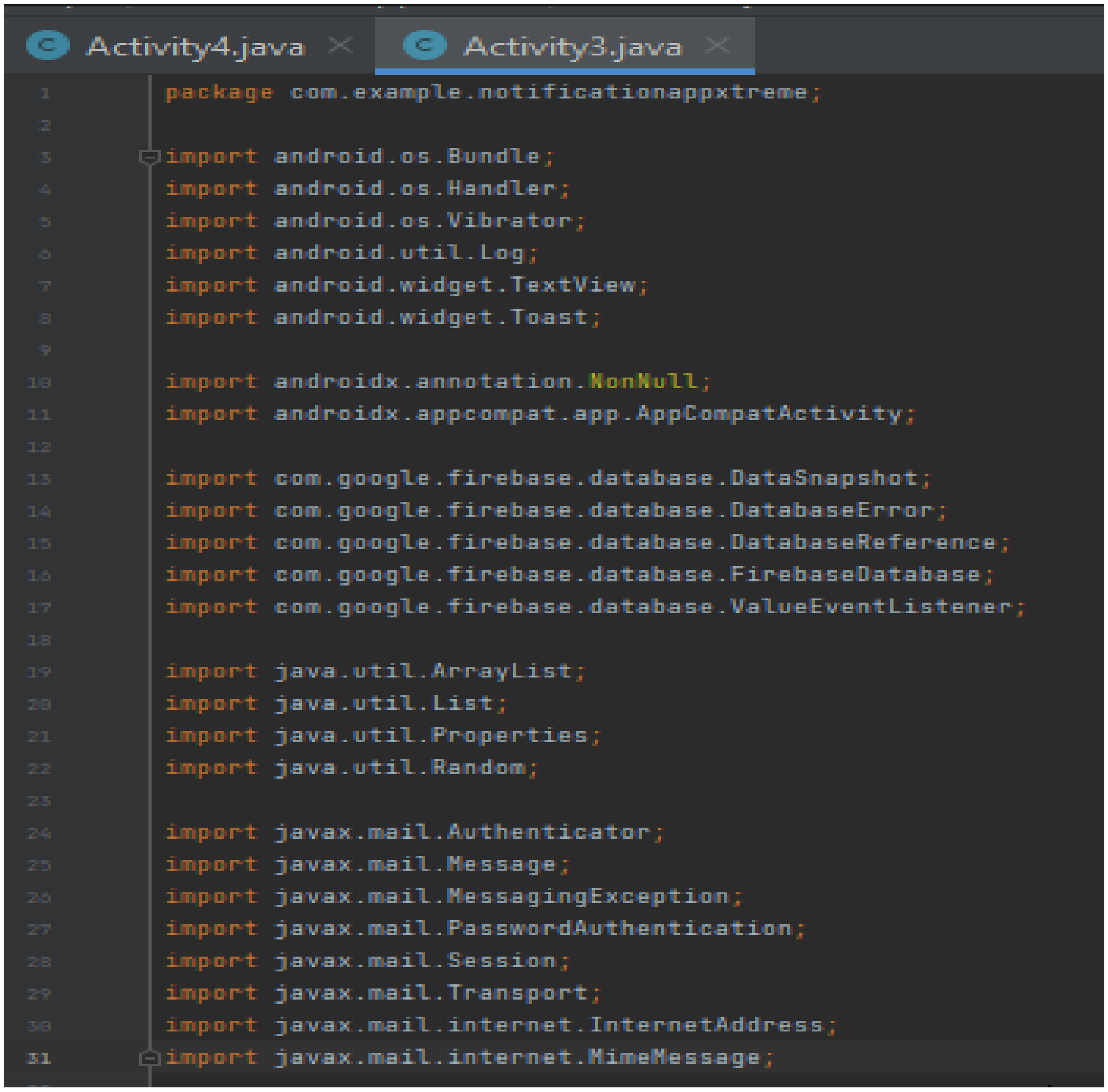
Calculates total acceleration and compares it to a predefined threshold. If the acceleration exceeds the threshold, a toast message is displayed, indicating a vigorous shake.

onAccuracyChanged Method: Placeholder method to handle changes in sensor accuracy if necessary.

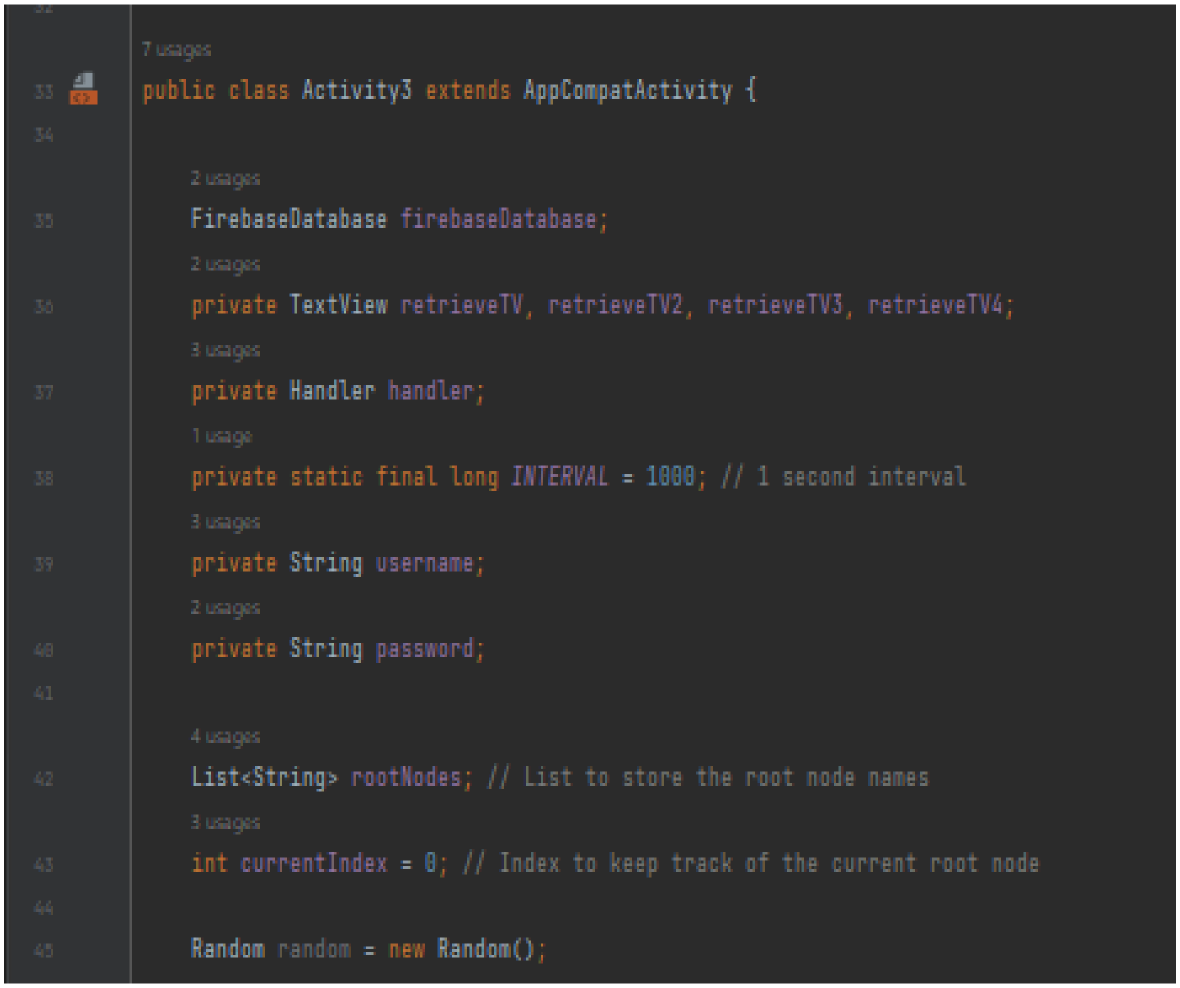
Activity2 demonstrates a basic Android application that leverages the accelerometer sensor to detect vigorous shakes of the device.

Documentation for Activity3 Class

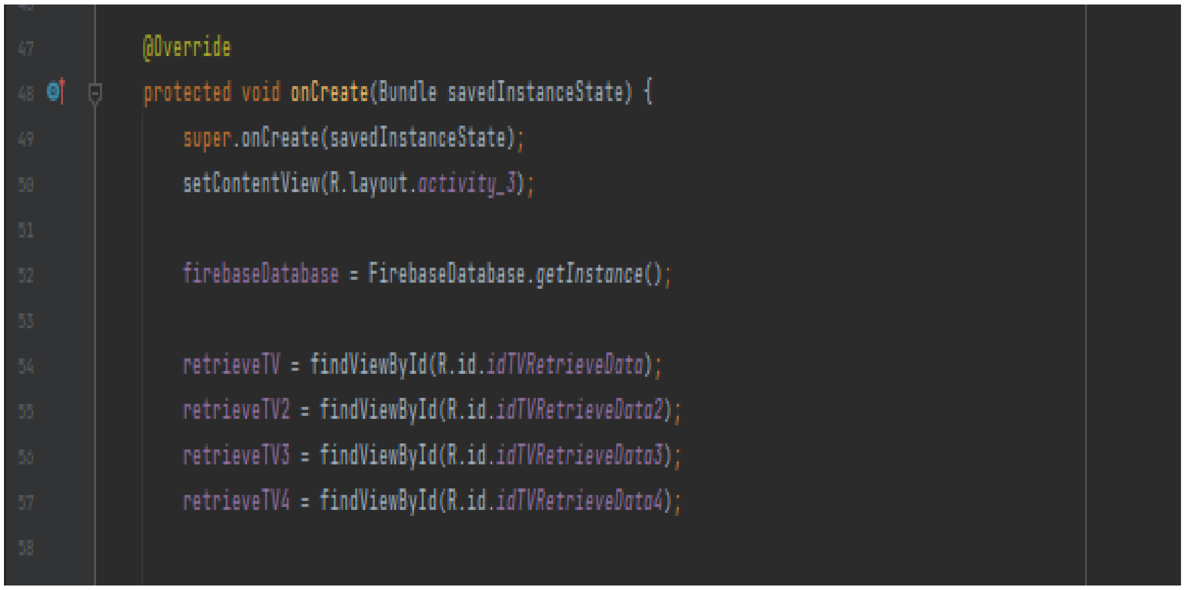
Code implemented in Activity3.



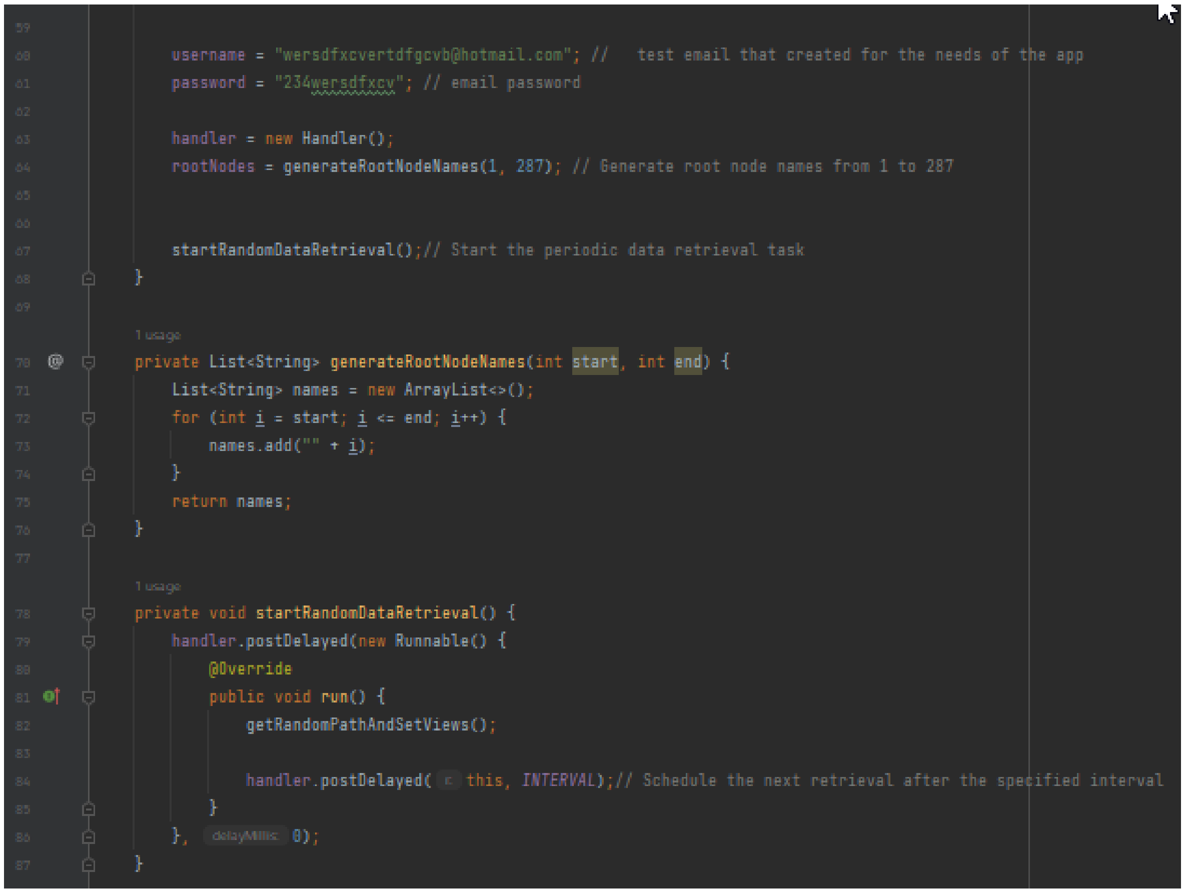
```java
package com.example.notificationappxtreme;

import android.os.Bundle;
import android.os.Handler;
import android.os.Vibrator;
import android.util.Log;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.Random;

import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
```



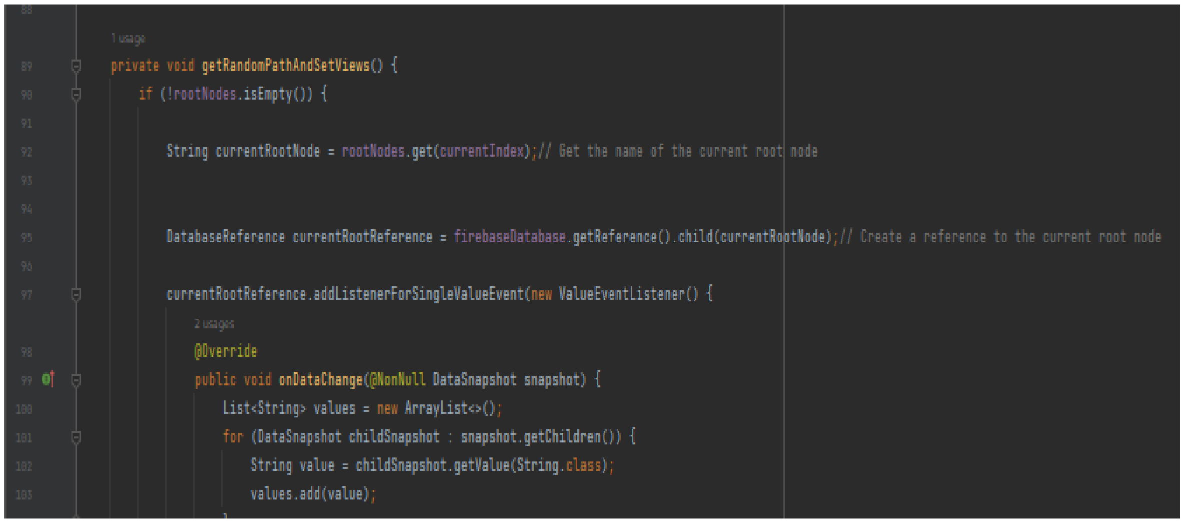
```
7 usages
public class Activity3 extends AppCompatActivity {

    2 usages
    FirebaseDatabase firebaseDatabase;
    2 usages
    private TextView retrieveTV, retrieveTV2, retrieveTV3, retrieveTV4;
    3 usages
    private Handler handler;
    1 usage
    private static final long INTERVAL = 1000; // 1 second interval
    3 usages
    private String username;
    2 usages
    private String password;

    4 usages
    List<String> rootNodes; // List to store the root node names
    3 usages
    int currentIndex = 0; // Index to keep track of the current root node

    Random random = new Random();
```



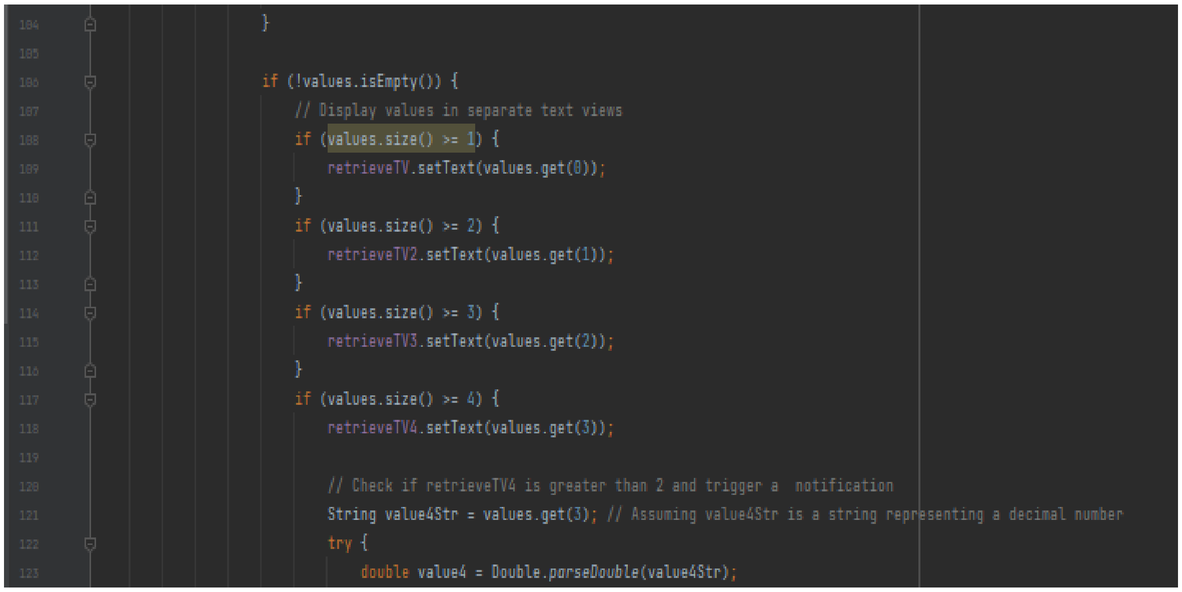
```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_3);

        firebaseDatabase = FirebaseDatabase.getInstance();

        retrieveTV = findViewById(R.id.idTVRetrieveData);
        retrieveTV2 = findViewById(R.id.idTVRetrieveData2);
        retrieveTV3 = findViewById(R.id.idTVRetrieveData3);
        retrieveTV4 = findViewById(R.id.idTVRetrieveData4);

```



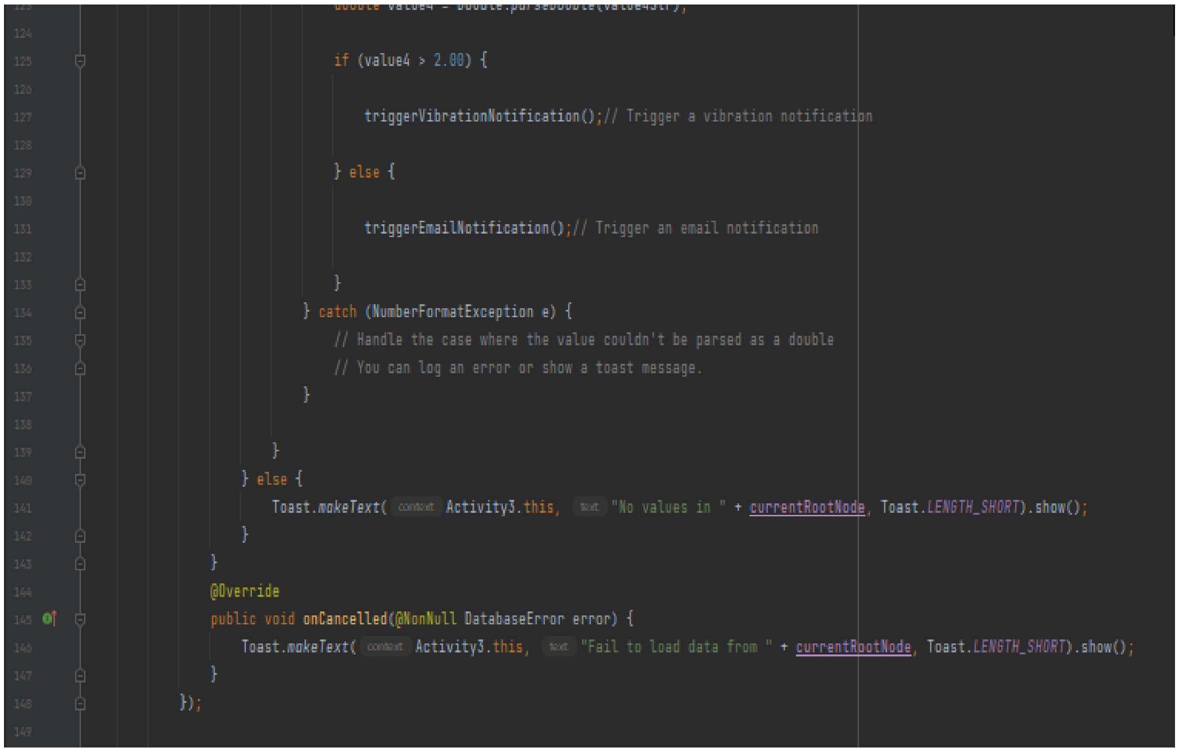
```
        username = "wersdfxcvertdfgcvb@hotmail.com"; //   test email that created for the needs of the app
        password = "234wersdfxcv"; // email password

        handler = new Handler();
        rootNodes = generateRootNodeNames(1, 287); // Generate root node names from 1 to 287


        startRandomDataRetrieval();// Start the periodic data retrieval task
    }

    1 usage
    private List<String> generateRootNodeNames(int start, int end) {
        List<String> names = new ArrayList<>();
        for (int i = start; i <= end; i++) {
            names.add("" + i);
        }
        return names;
    }

    1 usage
    private void startRandomDataRetrieval() {
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                getRandomPathAndSetViews();

                handler.postDelayed( this, INTERVAL);// Schedule the next retrieval after the specified interval
            }
        }, delayMillis: 0);
    }
```



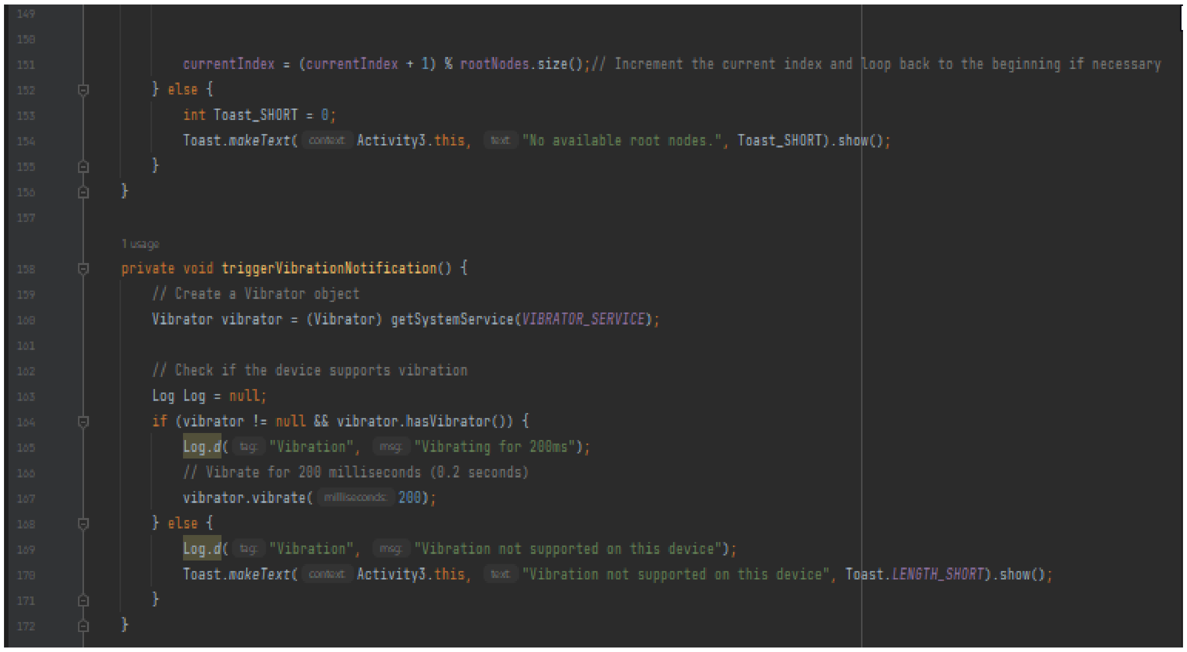
```
    1 usage
    private void getRandomPathAndSetViews() {
        if (!rootNodes.isEmpty()) {

            String currentRootNode = rootNodes.get(currentIndex);// Get the name of the current root node


            DatabaseReference currentRootReference = firebaseDatabase.getReference().child(currentRootNode);// Create a reference to the current root node

            currentRootReference.addListenerForSingleValueEvent(new ValueEventListener() {
                2 usages
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    List<String> values = new ArrayList<>();
                    for (DataSnapshot childSnapshot : snapshot.getChildren()) {
                        String value = childSnapshot.getValue(String.class);
                        values.add(value);
```



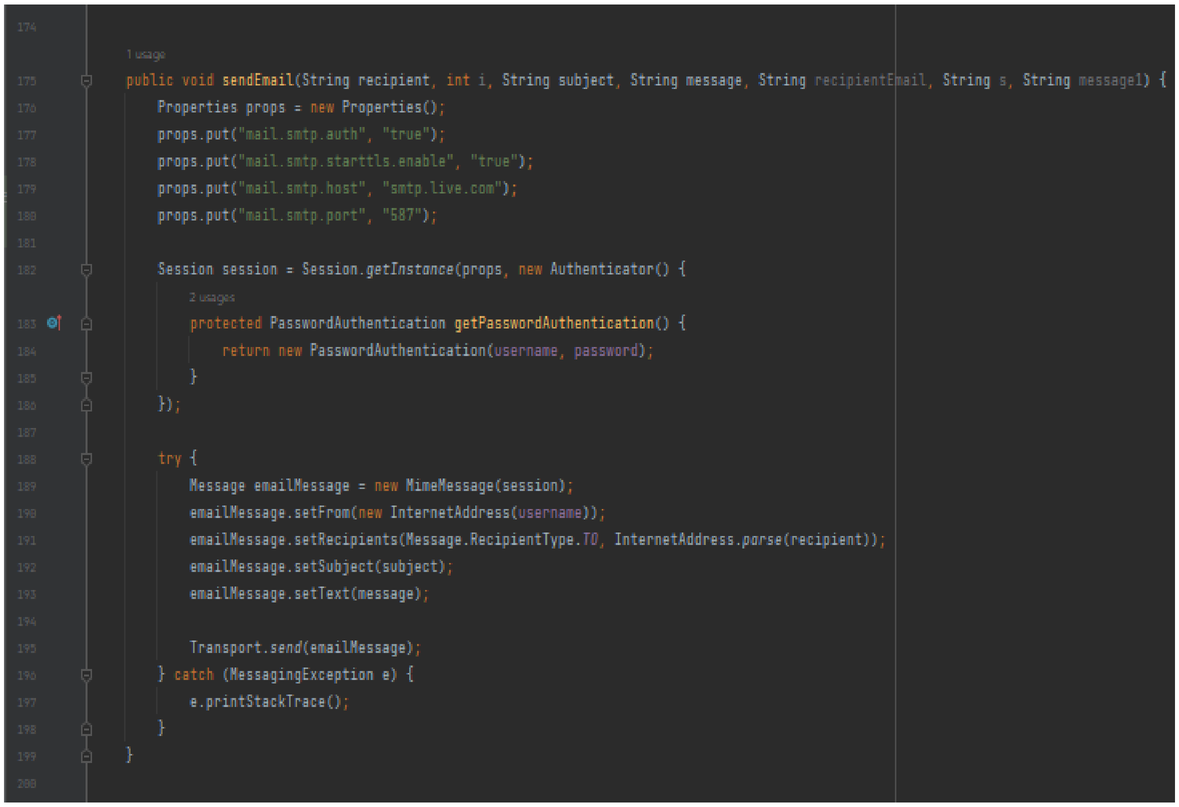
```java
                }

                if (!values.isEmpty()) {
                    // Display values in separate text views
                    if (values.size() >= 1) {
                        retrieveTV.setText(values.get(0));
                    }
                    if (values.size() >= 2) {
                        retrieveTV2.setText(values.get(1));
                    }
                    if (values.size() >= 3) {
                        retrieveTV3.setText(values.get(2));
                    }
                    if (values.size() >= 4) {
                        retrieveTV4.setText(values.get(3));

                        // Check if retrieveTV4 is greater than 2 and trigger a  notification
                        String value4Str = values.get(3); // Assuming value4Str is a string representing a decimal number
                        try {
                            double value4 = Double.parseDouble(value4Str);
```



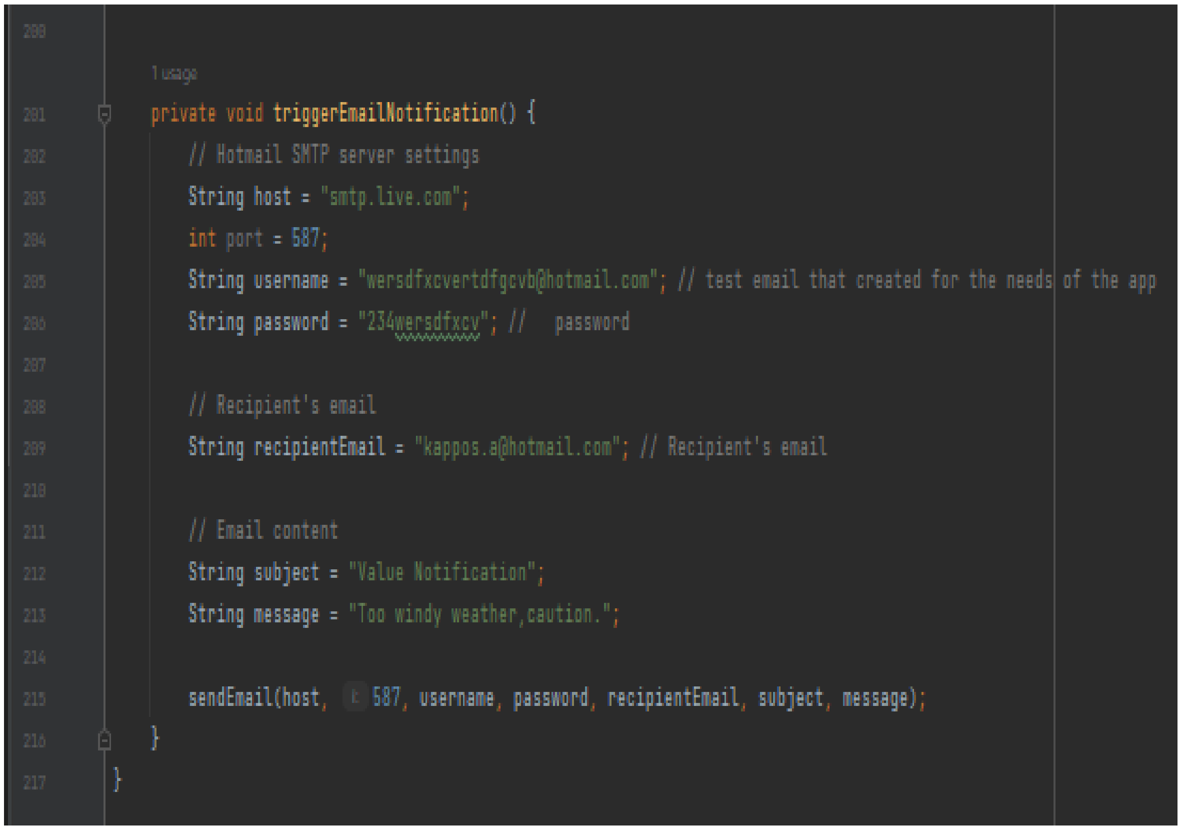
```java

                            if (value4 > 2.00) {

                                triggerVibrationNotification();// Trigger a vibration notification

                            } else {

                                triggerEmailNotification();// Trigger an email notification

                            }
                        } catch (NumberFormatException e) {
                            // Handle the case where the value couldn't be parsed as a double
                            // You can log an error or show a toast message.
                        }

                    }
                } else {
                    Toast.makeText(Activity3.this, "No values in " + currentRootNode, Toast.LENGTH_SHORT).show();
                }
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Toast.makeText(Activity3.this, "Fail to load data from " + currentRootNode, Toast.LENGTH_SHORT).show();
            }
        });
```



```
            currentIndex = (currentIndex + 1) % rootNodes.size();// Increment the current index and loop back to the beginning if necessary
        } else {
            int Toast_SHORT = 0;
            Toast.makeText( context: Activity3.this,  text: "No available root nodes.", Toast_SHORT).show();
        }
    }

    1 usage
    private void triggerVibrationNotification() {
        // Create a Vibrator object
        Vibrator vibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);

        // Check if the device supports vibration
        Log Log = null;
        if (vibrator != null && vibrator.hasVibrator()) {
            Log.d( tag: "Vibration",  msg: "Vibrating for 200ms");
            // Vibrate for 200 milliseconds (0.2 seconds)
            vibrator.vibrate( milliseconds: 200);
        } else {
            Log.d( tag: "Vibration",  msg: "Vibration not supported on this device");
            Toast.makeText( context: Activity3.this,  text: "Vibration not supported on this device", Toast.LENGTH_SHORT).show();
        }
    }
```



```

    1 usage
    public void sendEmail(String recipient, int i, String subject, String message, String recipientEmail, String s, String message1) {
        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", "smtp.live.com");
        props.put("mail.smtp.port", "587");

        Session session = Session.getInstance(props, new Authenticator() {
            2 usages
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username, password);
            }
        });

        try {
            Message emailMessage = new MimeMessage(session);
            emailMessage.setFrom(new InternetAddress(username));
            emailMessage.setRecipients(Message.RecipientType.TO, InternetAddress.parse(recipient));
            emailMessage.setSubject(subject);
            emailMessage.setText(message);

            Transport.send(emailMessage);
        } catch (MessagingException e) {
            e.printStackTrace();
        }
    }

```



```
    1 usage
    private void triggerEmailNotification() {
        // Hotmail SMTP server settings
        String host = "smtp.live.com";
        int port = 587;
        String username = "wersdfxcvertdfgcvb@hotmail.com"; // test email that created for the needs of the app
        String password = "234wersdfxcv"; //   password

        // Recipient's email
        String recipientEmail = "kappos.a@hotmail.com"; // Recipient's email

        // Email content
        String subject = "Value Notification";
        String message = "Too windy weather,caution.";

        sendEmail(host,  587, username, password, recipientEmail, subject, message);
    }
}
```

Code Interpretation.

Class Declaration and Fields: This class extends AppCompatActivity and encapsulates the main logic for the notification application.

Initialization and Setup: Initializes Firebase, Text Views, email credentials, and other necessary components. Generates a list of root node names from 1 to 287.

Initiates the periodic data retrieval task.

Data Retrieval and Display: Retrieves data from Firebase using a randomly selected root node. Updates Text Views with the retrieved data. Checks a specific condition (retrieveTV4 > 2) and triggers either a vibration or email notification.

Vibration Notification: Utilizes the device's vibrator to trigger a vibration notification.

Checks if the device supports vibration and handles accordingly.

Email Notification: Sends an email notification using Hotmail SMTP settings and predefined content. Email credentials, recipient email, and content are hardcoded for demonstration purposes.

Email Sending Method: Configures email properties and uses JavaMail API to send an email.

Additional Notes: The Handler is employed for periodic data retrieval at a 1-second interval.

The code contains error handling for database queries and email sending operations, displaying appropriate Toast messages in case of failure.The hardcoded email credentials and recipient email should be securely managed in a production environment.

Activity3 is an Android application component that fetches data from Firebase, displays it on the UI, and triggers notifications based on specific conditions. It demonstrates integration with Firebase, device vibration, and email functionality for notification purposes.

Future Expansion of current functionality

An Android application focused on preventing accidents in the maintenance of machinery can play a crucial role in enhancing workplace safety and operational efficiency. Here are some potential future functionalities for such an application:

Maintenance Schedule and Reminders: Allow users to schedule routine maintenance tasks for machinery and receive automated reminders. This ensures timely inspections and reduces the risk of accidents caused by neglected maintenance.

IoT Integration for Real-time Monitoring: Integrate with Internet of Things (IoT) devices to monitor the condition of machinery in real-time. Sensors can provide data on temperature, pressure, vibration, and other relevant parameters, helping detect potential issues before they escalate.

Augmented Reality (AR) Maintenance Guides: Implement AR technology to provide step-by-step maintenance guides overlaid on the physical machinery. This can assist maintenance personnel in performing tasks accurately and safely.

Predictive Maintenance Analytics: Utilize machine learning algorithms to analyze historical maintenance data and predict potential equipment failures. This enables proactive maintenance, reducing downtime and minimizing the risk of accidents due to unexpected breakdowns.

Safety Checklists and Procedures: Include digital checklists for safety inspections and maintenance procedures. Ensure that technicians follow standardized safety protocols during maintenance activities to prevent accidents.

Remote Assistance and Collaboration: Enable remote assistance through video calls or augmented reality to connect on-site technicians with experts. This can facilitate real-time guidance during complex maintenance tasks, improving accuracy and safety.

Hazard Identification and Reporting: Allow users to identify and report potential hazards during maintenance activities. This crowdsourced information can contribute to a safer work environment by addressing and mitigating risks promptly.

Training Modules and Certification Tracking: Provide training modules for maintenance personnel, covering safety protocols, equipment handling, and emergency procedures. Track and manage certifications to ensure that workers are adequately trained for their tasks.

Equipment Tagging and QR Codes: Implement a system for tagging machinery with QR codes that link to digital maintenance records, manuals, and safety information. This simplifies access to critical information during maintenance and inspections.

Environmental Monitoring: Integrate sensors to monitor environmental conditions such as air quality, temperature, and noise levels. This information can contribute to a safer and healthier workplace.

Automated Reporting and Documentation: Streamline the documentation process by automating the generation of maintenance reports, including work performed, parts replaced, and safety checks. This ensures a comprehensive record of maintenance activities.

Emergency Response Integration: Connect the application to emergency response systems, allowing for quick alerts and assistance in case of accidents or unexpected events during maintenance.

Compliance Tracking: Monitor and track compliance with safety regulations and industry standards. Provide alerts and recommendations to ensure that maintenance activities align with safety guidelines.

By incorporating these advanced features, an Android application for machinery maintenance can significantly contribute to accident prevention, worker safety, and overall operational efficiency in industrial settings.

Conclusion.

Summarizing all the above chapters the application is able to give solid and early information to the user and to any relevant person that needs that kind of notification in case of accident(free fall) and the users in addition to the supervisors can have early notice for extreme weather conditions .The measurements and the type of the conditions can vary depending on the environment and the tolerance limits of the accepted values can be modified to whatever needed in order to have realistic alarms, to the device. The combined functionality of the Notification App Extreme provides early warning for lateral and vertical accelerations, that can detect free falls during the ongoing maintenance and can provide insights with early warnings according to the weather conditions of the site, crucial information for any organizations that needs to avoid accidents for its employees.

Acknowledgements

I would like to thank my mentor/professor on this feet, Efthymios Alepis for all his efforts to transfer to us as much knowledge he could with the limited resources that has in his disposal. I will be forever grateful for his assistance.

References

http://www.stackoverflow.com

https://www.udemy.com/home

https://developer.android.com/studio

https://firebase.google.com