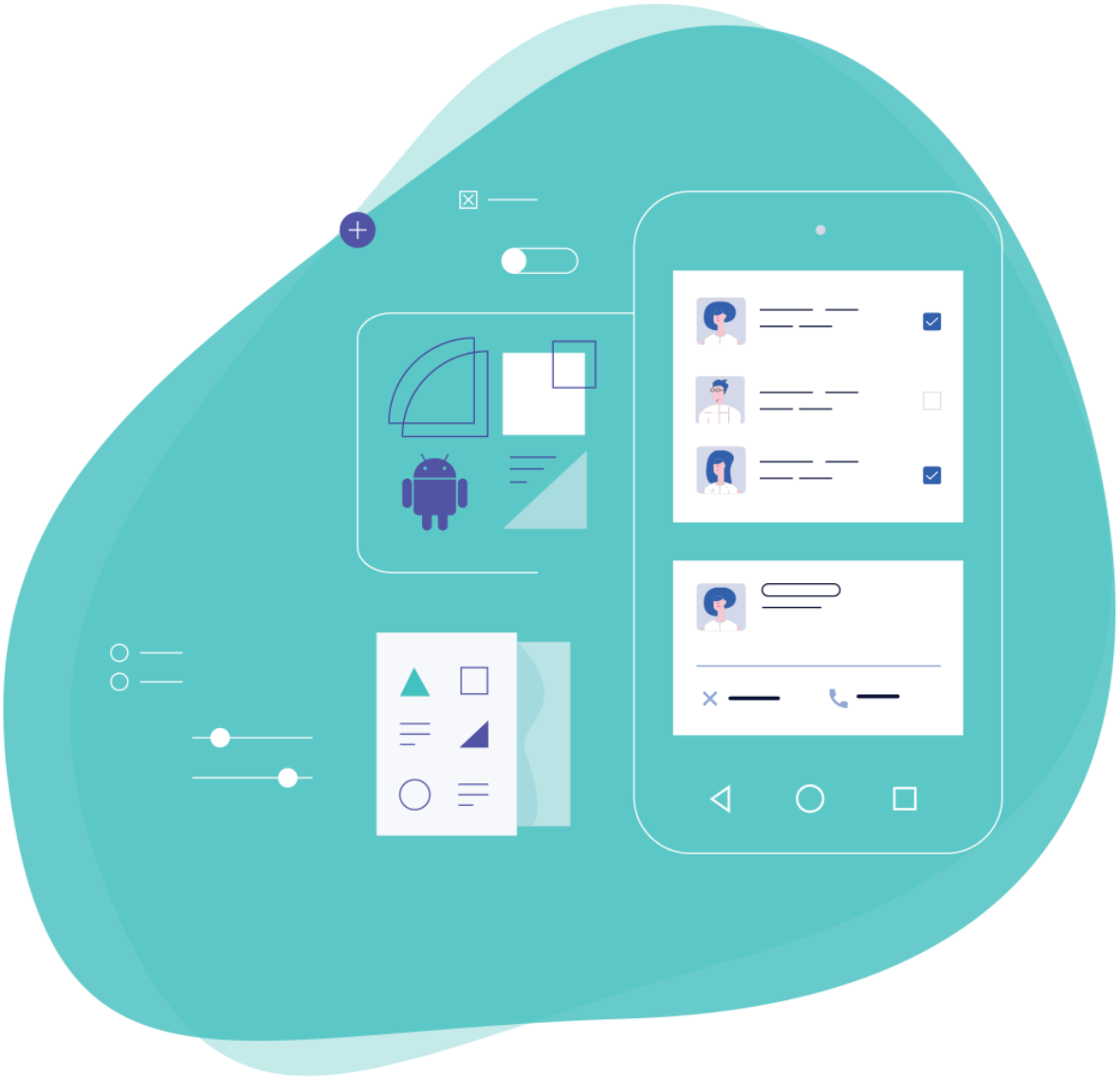


ABSTRACT

1. ABSTRACT

This project “ANDROID MOBILE TRACKING SYSTEM” aims to create an Android mobile application that offers advanced GPS tracking features, including real-time location tracking and path recording on a Google Map interface. The app will provide users with the ability to monitor their battery levels during tracking sessions, ensuring they are aware of their device's power status. Additionally, the application will include a feature to detect and display nearby Wi-Fi networks, enhancing connectivity options for users. One of the key goals of this project is to develop a tracking solution that consumes minimal battery compared to existing applications, thus enabling users to track their location for extended periods without significant power drain. Another objective is to implement a feature that allows users to track multiple devices simultaneously, providing a convenient solution for users with multiple devices. The app will be designed with a user-friendly interface, making it easy for users to initiate and manage tracking sessions. It will also offer customization options, allowing users to adjust settings such as tracking intervals and map display preferences. Moreover, the application will prioritize user privacy and data security, ensuring that all location data is stored and transmitted securely. In conclusion, this project seeks to develop an innovative GPS tracking application for Android devices, offering advanced features, minimal battery consumption, and enhanced connectivity options.

INTRODUCTION

2. INTRODUCTION

In today's fast-paced world, the need for reliable and efficient GPS tracking solutions is more significant than ever. This project addresses this need by developing an advanced Android mobile application that leverages the power of Kotlin and Jetpack Compose UI to provide users with a seamless and intuitive tracking experience. By adopting the MVVM architecture, the app ensures a robust and scalable codebase, making it easier to maintain and update. Room Database integration allows for efficient storage and management of location data, while Dagger ensures that the app remains flexible and easy to extend. Firebase integration adds a layer of functionality, enabling real-time location sharing and cloud based storage for location history. This integration, combined with the Google Maps API, allows users to view their live location and track their path on the map with high accuracy. One of the key highlights of this project is its focus on optimizing battery consumption, ensuring that users can rely on the app for extended periods without draining their device's battery. Additionally, the app supports tracking multiple devices simultaneously, making it ideal for a wide range of applications, including fleet management and personal safety. Overall, this project aims to deliver a cutting-edge GPS tracking application that meets the needs of modern users while incorporating the latest technologies and best practices in Android app development.

SYSTEM

SPECIFICATION

3. SYSTEM SPECIFICATION

3.1 HARDWARE SPECIFICATION

System specifications

Processor	:	Intel Core i5 10GEN OR Ryzen 5 5000s
RAM	:	16 GB With 8GB of Free Space
Hard Disk	:	512 GB SSD

Mobile Specification

Processor	:	Any MTK or SD with Min 4 Core Count
Operating System	:	Android Version 7.0 to 14.12.9

3.2 SOFTWARE SPECIFICATION

Front-end	:	Jet Brains's Jetpack compose ,Material 3 Ui Components
Back-end	:	FireBase schema less Database
Architecture	:	MVVM(Model View ViewModel)

3.3 SOFTWARE DESCRIPTION

Kotlin

Kotlin is a statically typed programming language that runs on the Java Virtual Machine (JVM) and can also be compiled to JavaScript or native code. It was created by JetBrains, the company behind popular development tools like IntelliJ IDEA, PhpStorm, and PyCharm. Kotlin was first announced in 2011 and released to the public in 2016.

One of the main goals behind Kotlin's development was to address some of the shortcomings of Java while maintaining compatibility with existing Java codebases. Kotlin is designed to be concise, expressive, and safe. It offers many modern language features such as null safety, type inference, extension functions, and data classes, which can help developers write more robust and maintainable code.

Kotlin has gained popularity in recent years, especially in the Android development community, where it is now officially supported by Google as a first-class language for Android app development. Kotlin's popularity stems from its ability to reduce boilerplate code, improve developer productivity, and provide seamless interoperability with existing Java code.

Traditional Java vs Kotlin

Traditional Java and Kotlin are both programming languages that run on the Java Virtual Machine (JVM) and are used for building a wide range of applications. However, there are several key differences between the two:

1. Null Safety: Kotlin has built-in null safety features, which help prevent null pointer exceptions at runtime. In Java, handling null values requires manual checks, which can be error-prone.
2. Conciseness: Kotlin is often more concise than Java, meaning that developers can achieve the same functionality with less code. This is due to features like type inference, data classes, and extension functions.
3. Interoperability: Kotlin is fully interoperable with Java, which means that Kotlin code can call Java code and vice versa. This makes it easy to migrate existing Java projects to Kotlin gradually.

4. Functional Programming: Kotlin has strong support for functional programming concepts, such as higher-order functions, lambda expressions, and immutability. While Java has some support for these concepts, it is more limited.
5. Tooling and IDE Support: Kotlin is developed by JetBrains, the company behind popular IDEs like IntelliJ IDEA and Android Studio. This means that Kotlin has excellent tooling support, including code completion, refactoring, and debugging features.
6. Adoption and Community: Kotlin has been gaining popularity steadily since its release, especially in the Android development community. However, Java still remains more widely used and has a larger ecosystem and community support.

Kotlin offers several advantages over traditional Java, particularly in terms of conciseness, null safety, and interoperability. However, Java remains a popular choice for many developers, especially for large-scale enterprise applications where stability and compatibility are important factors.

Jetpack Compose

Jetpack Compose is a modern UI toolkit for building native Android apps developed by Google. It revolutionizes the way developers create user interfaces by introducing a declarative approach. Instead of manually defining every aspect of the UI and its behavior, developers describe the UI using composable functions. These functions are lightweight and stateless, making it easier to build and maintain complex UIs.

One of the key advantages of Jetpack Compose is its reactive programming model. UI components automatically update in response to changes in the underlying data, ensuring that the UI always reflects the latest app state. This reactive approach simplifies the development process and reduces the likelihood of bugs related to UI state management.

Jetpack Compose also provides a set of Material Design components, allowing developers to create apps with a consistent and polished look. These components follow the Material Design guidelines and can be customized to fit the app's branding and design requirements.

Another important aspect of Jetpack Compose is its interoperability with existing Android code. Developers can gradually migrate their apps to Jetpack Compose, integrating

new Compose components with their existing codebase. This allows for a smooth transition to the new UI toolkit without the need for a complete rewrite.

Overall, Jetpack Compose offers a modern, efficient, and flexible way to build UIs for Android apps. Its declarative and reactive nature, combined with its interoperability and tooling support, make it a powerful choice for developers looking to create engaging and dynamic user interfaces.

MVVM A RCHITECTURE

MVVM (Model-View-ViewModel) is an architectural pattern that enhances the separation of concerns in software development, particularly in user interface (UI) development. In MVVM, the Model represents the data and business logic, the View represents the UI components, and the ViewModel acts as a mediator between the Model and the View.

The Model holds the application's data and business logic, such as data validation and manipulation. The View is responsible for displaying the UI and interacts with the ViewModel to update the UI based on changes in the data. The ViewModel exposes data from the Model to the View and contains UI-related logic, making it easier to test and maintain.

One of the key advantages of MVVM is its testability. The separation of concerns allows for easier unit testing of the ViewModel, which contains the majority of the application's logic. Additionally, the ViewModel can be reused across different Views, promoting code reusability and maintainability.

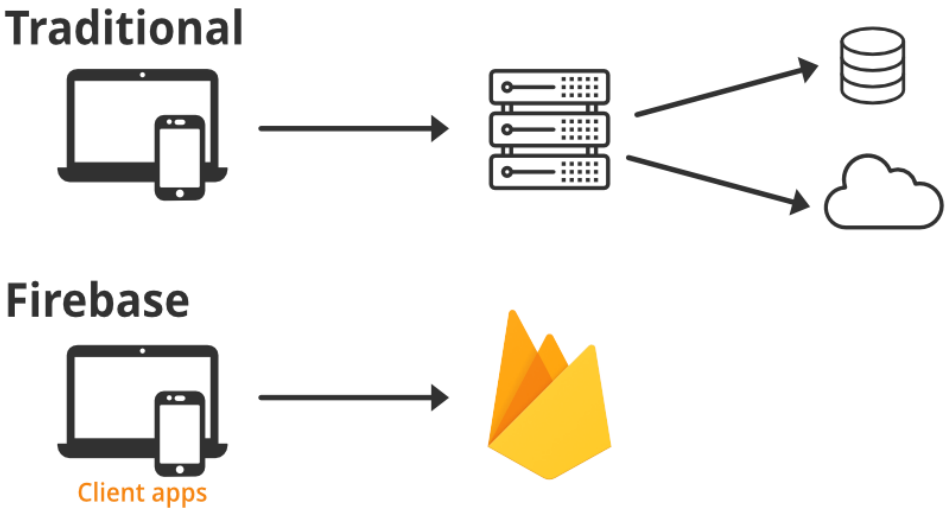
In Android development, MVVM is commonly used with libraries like Jetpack Compose for building UIs, Room Database for data persistence, Dagger for dependency injection, Firebase for backend services, and Google Maps API for location-based services. This architectural pattern helps developers build robust, maintainable, and testable Android applications.

Firebase Database

Firebase is a platform developed by Google for creating mobile and web applications. It provides developers with a variety of tools and services to help them build high-quality apps, improve app quality, and grow their user base.

Some key features of Firebase include:

1. Realtime Database: A cloud-hosted database that supports realtime data synchronization.
2. Authentication: A service that provides easy-to-use authentication methods, including email/password, Google, Facebook, and more.
3. Cloud Firestore: A flexible, scalable database for mobile, web, and server development.
4. Cloud Storage: A powerful, simple, and cost-effective object storage service for storing user-generated content like photos and videos.
5. Firebase Cloud Messaging (FCM): A cross-platform messaging solution that lets you reliably send messages at no cost.
6. Firebase Hosting: A fast and secure way to host web apps and static content.
7. Analytics: Gain insights into user behavior and app performance.
8. Crashlytics: A tool for monitoring app stability and analyzing crash reports.
9. Performance Monitoring: Helps you understand app performance and optimize user experience.



SYSTEM STUDY

4. SYSTEM STUDY

4.1 Existing System

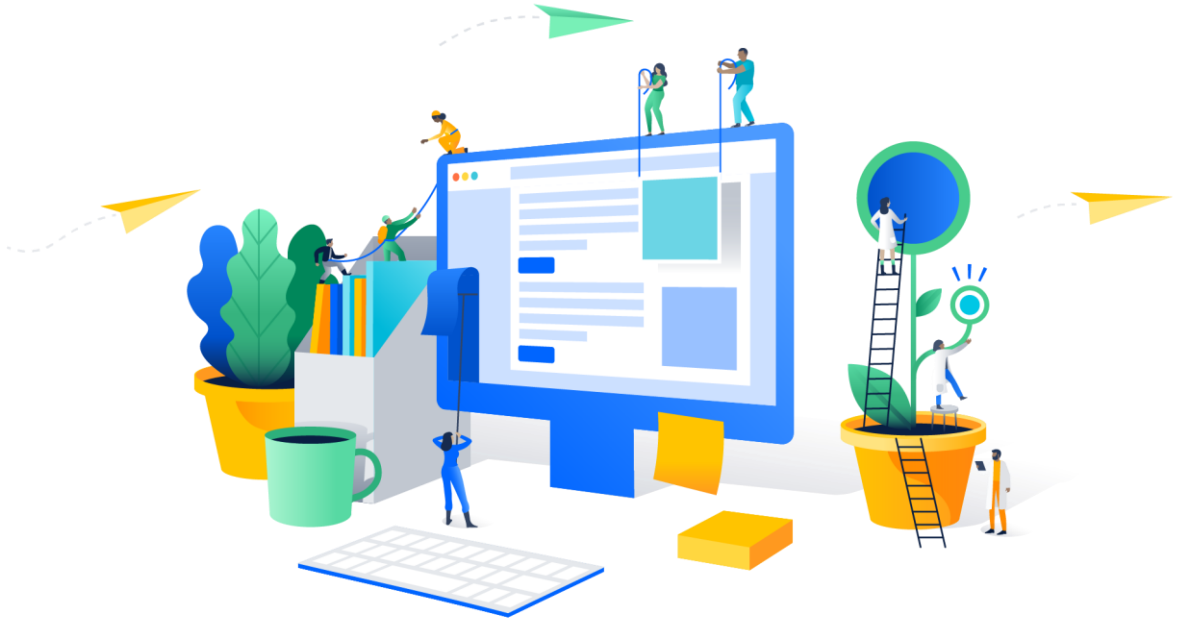
The existing system may lack real-time GPS tracking and path visualization features. It might not have a dedicated battery optimization mechanism, which can lead to excessive battery drain during GPS tracking. The system may also lack the ability to monitor Wi-Fi networks for improved connectivity.

4.2 Proposed System

The proposed system aims to address the limitations of the existing system by introducing real-time GPS tracking with path visualization on a Google Map interface. It includes a battery optimization feature to minimize battery consumption during tracking. Additionally, the system offers Wi-Fi network monitoring to assist users in finding nearby networks for improved connectivity.

4.3 Features

1. Real-time GPS Tracking: Users can track their location in real-time on a Google Map interface.
2. Path Visualization: The system displays the path traveled by the user on the map for easy reference.
3. Battery Optimization: The system optimizes battery usage during GPS tracking to minimize drain.
4. Wi-Fi Network Monitoring: Users can view nearby Wi-Fi networks to improve connectivity.
5. Live Location Sharing: Users can share their live location with others for safety or convenience.
6. Customizable Alerts: Users can set up custom alerts for specific locations or events.
7. Location History: The system stores the user's location history for future reference.
   Multi-device Tracking: The system supports tracking multiple devices simultaneously



SYSTEM DESIGN

5. SYSTEM DESIGN

defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements. It is a solution that fulfills the needs of users while meeting the constraints of the project, such as performance, reliability, scalability, maintainability, and cost-effectiveness. System design involves multiple phases, including requirements analysis, feasibility study, architectural design, detailed design, implementation, testing, deployment, and maintenance.

5.1 Logical Design

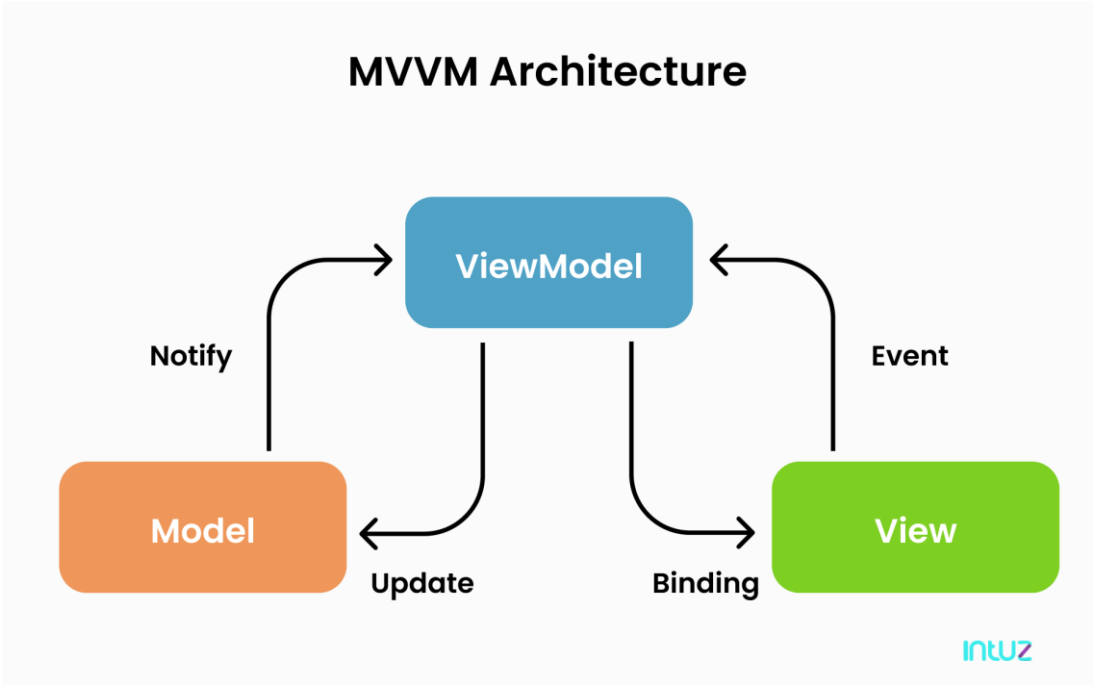
1. **Database Design**: Utilize Firebase Realtime Database to store location data and other relevant information. Use appropriate data structures to efficiently store and retrieve data.
2. **Application Flow**: Implement a structured application flow using MVVM architecture to separate concerns and improve maintainability.
3. **User Interface Design**: Design a user-friendly interface using Jetpack Compose UI, following Material Design guidelines for a modern and intuitive user experience.
4. **Location Tracking Logic**: Implement logic to continuously track the device's location using Google Maps API and update the UI in real-time.

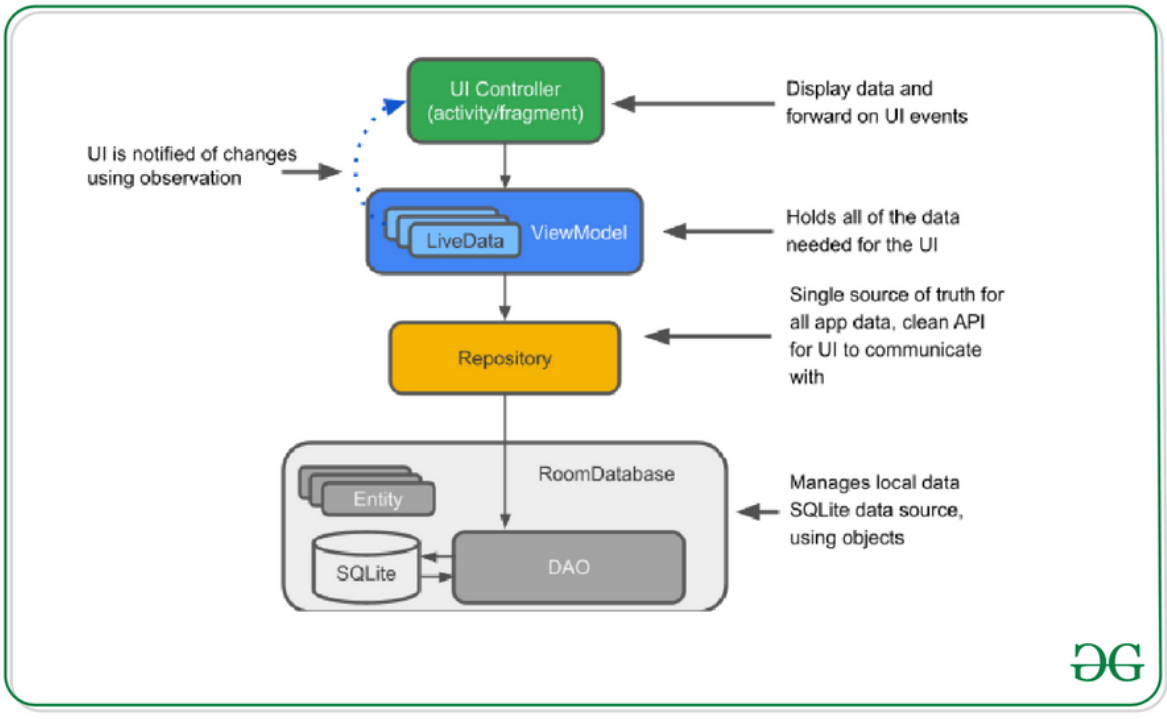
5.2 Physical Design

1. **Server Infrastructure**: Utilize Firebase for the backend, providing scalable and reliable cloud infrastructure for storing and processing data.
2. **Device Compatibility**: Ensure compatibility with a wide range of Android devices by optimizing code and resources for different screen sizes and hardware specifications.
3. **Battery Optimization**: Implement strategies to optimize battery usage, such as limiting location updates frequency and using efficient algorithms for data processing.
4. **Network Connectivity**: Utilize Firebase Cloud Messaging (FCM) for efficient communication between the server and the app, ensuring real-time updates and notifications.

The system design focuses on providing a robust and efficient solution for GPS tracking, ensuring a seamless user experience and optimal performance across various devices and network conditions.

5.3 ARCHITECTURAL DESIGN





The components which we will be using inside the application are listed above with a detailed explanation

LiveData

Live Data is a data holder class that can be observed. It holds as well as caches the latest version of the data and notifies our observer whenever the data is being updated or changed. Live Data automatically manages all of this since it is aware of the relevant lifecycle status changes while observing.

ViewModel

View Modal acts as a communication center between repository and UI. The UI no longer needs to worry about the origin of the data. The ViewModel instances survive Activity/Fragment recreation.

Repository

Repository is a class that is mainly used to manage multiple sources of data.

Entity

Entity is an annotated class that is used to describe a database table when we are working with Room.

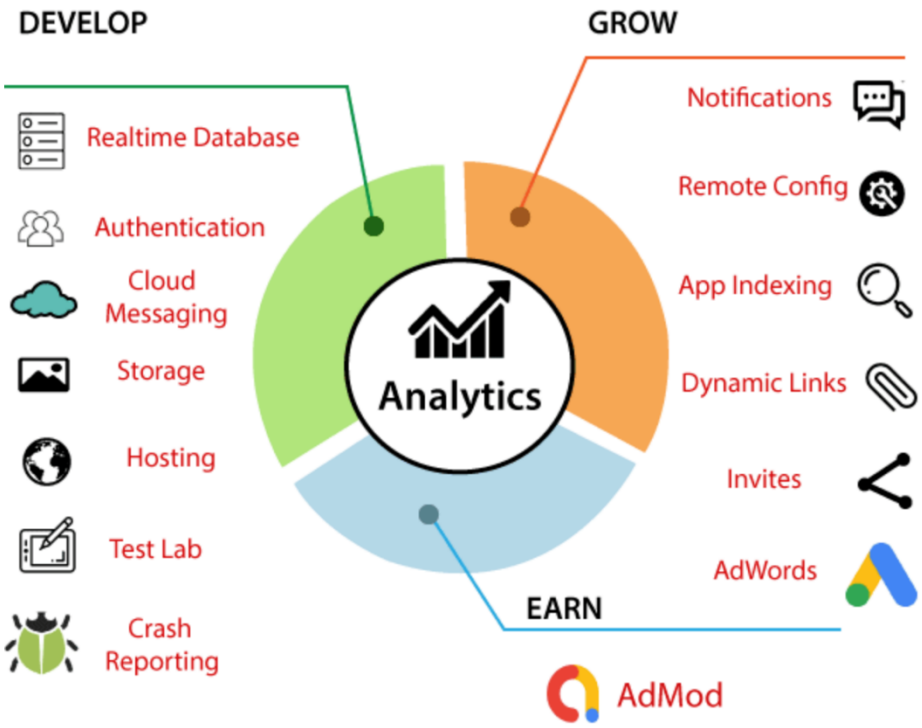
Room Database

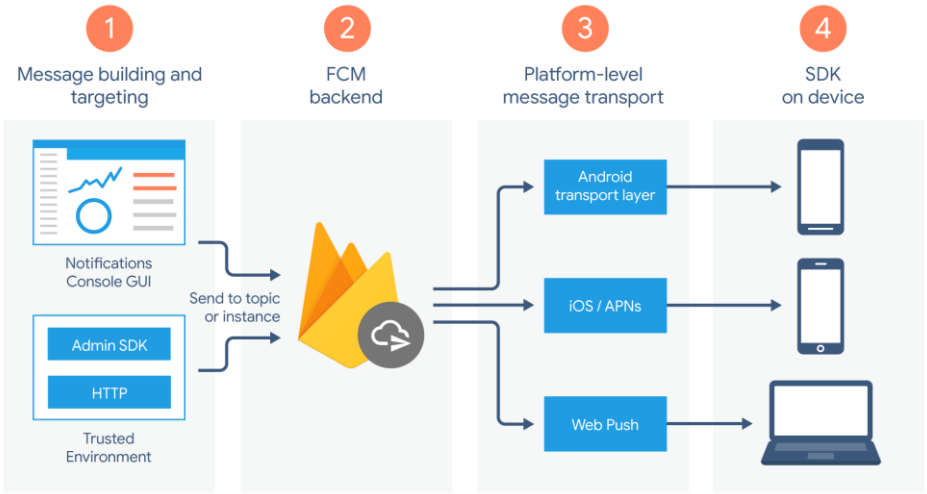
Room Database is an improvised version of SQLite Database. It simplifies the database work and serves as an access point to the underlying SQLite database. The room uses DAO to issue queries to the SQLite database.

DAO

DAO is a Data Access Object which is used for mapping SQL queries to functions.

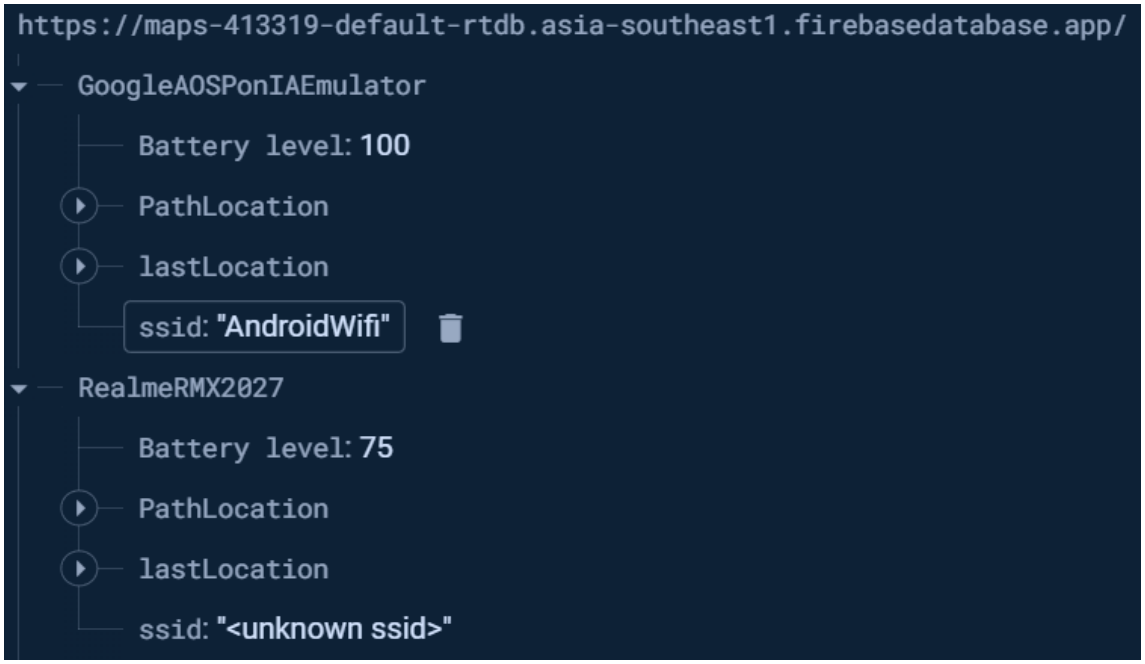
5.4 FireBase Design





In Firebase, a connection is a user interacting with Firebase, and a connection opens when a user opens the app, and closes when the user leaves the app. This means that to hit the 100 connection cap, 100 users must have the app open at the same time.

5.5 DATABASE TREE



Root Node is name of the Device that stores :

Path Location : series of Co-ordinates

Last Location : single LatLng that stores last Location of device

SSID : It defines the Device connected WiFi SSID details

Battery Level : indicates the battery percentage

PROJECT DESCRIPTION

6.PROJECT DESCRIPTION

6.1 Description

The Android Mobile GPS Tracking Application is a cutting-edge solution that transforms how users track their locations and monitor their paths. By leveraging GPS technology, the app provides real-time location tracking with unparalleled accuracy. Users can easily view the paths they have traveled at any time, allowing them to revisit their journeys and explore new routes with confidence.

One of the key features of the application is its focus on minimizing battery consumption, ensuring that users can track their locations without worrying about draining their device's battery. Additionally, the app offers a simple and intuitive user interface, making it easy for users to navigate and access its features.

With live location updates, users can stay informed about their whereabouts, making it ideal for travelers, outdoor enthusiasts, and anyone who wants to keep track of their movements. The app also prioritizes data security, encrypting user data to ensure privacy and confidentiality.

Overall, the Android Mobile GPS Tracking Application is a powerful tool that offers accurate location tracking, minimal battery consumption, and a user-friendly interface, making it an essential companion for anyone who values tracking their location and exploring new paths.

6.2 Key Features:

1. **Live Location Tracking** Users can track their current location in real-time on a Google Map interface within the app.
2. **Path Tracking** The app records the path traveled by the user, allowing them to view their entire route on the map.
3. **Battery Optimization** Special emphasis has been placed on optimizing battery consumption, ensuring that the app does not drain the device's battery excessively.

4. **Wi-Fi Network Finder** The app includes a feature to find nearby Wi-Fi networks, providing users with additional connectivity options.
5. **Multiple Device Tracking** The application supports tracking multiple devices simultaneously, allowing users to monitor the location of multiple devices from a single interface.

6.3 Technology Stack

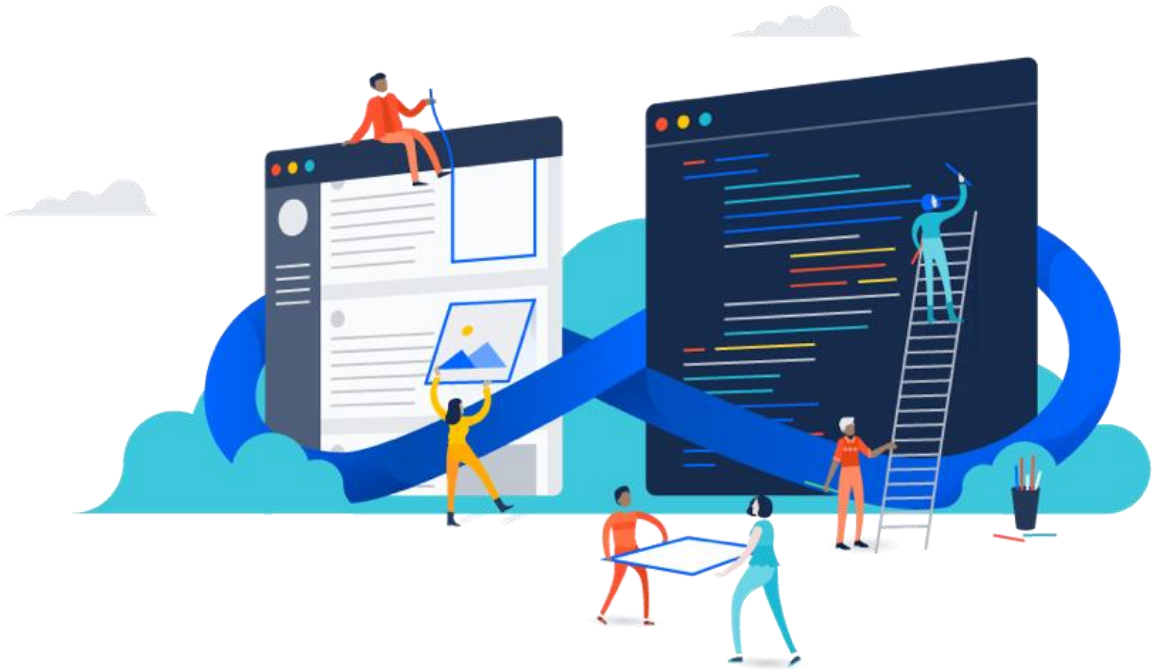
- **Development Language:** Kotlin
- **User Interface:** Jetpack Compose UI
- **Architecture Pattern:** MVVM (Model-View-ViewModel)
- **Database:** Room Database
- **Dependency Injection:** Dagger
- **Backend Services:** Firebase
- **Maps Integration:** Google Maps API

6.4 Target Audience:

The Android Mobile GPS Tracking Application is designed for individuals who require a reliable and efficient GPS tracking solution for personal or professional use. It is suitable for a wide range of use cases, including outdoor activities, fleet management, and location-based services.

6.5 Benefits:

- Provides real-time location tracking and path monitoring.
- Optimizes battery consumption for prolonged use.
- Supports tracking of multiple devices simultaneously.
- Enhances connectivity with Wi-Fi network finder feature.
- Offers a user-friendly and intuitive interface.



FUTURE ENHANCEMENT

7.Future Enhancements

• Integration with additional sensors for enhanced tracking capabilities, such as gyroscope and accelerometer for more precise movement tracking.

• Support for custom maps and overlays, allowing users to customize their tracking experience with personalized maps and markers.

• Enhanced security features for data protection, including advanced encryption algorithms and multi-factor authentication options.

• Integration with social media platforms for sharing location updates, enabling users to share their routes and experiences with friends and followers.

• Access to the camera by the user on another mobile device, enabling them to view live video feeds of the tracked location.

• Access to the microphone, allowing users to listen to ambient sounds at the tracked location.

• Mobile motion detection, which can detect movements of the tracked device and provide alerts or notifications.

• Stolen device communication with nearby devices, where the stolen device can communicate with nearby devices and notify them that it was stolen from its real user, potentially aiding in recovery efforts.

SYSTEM TESTING

8. SYSTEM TESTING

System Testing for the Android Mobile GPS Tracking Application was conducted to ensure the application meets specified requirements and functions correctly. The testing process covered various aspects of the application's functionality, performance, compatibility, usability, security, integration, load handling, stress management, and acceptance by end-users.

Functionality Test

Functionality Testing included verifying that the live location tracking accurately displays the user's current location on the map, ensuring the path tracking feature records and displays the user's path correctly, and confirming that the battery optimization feature reduces battery consumption compared to other tracking apps.

Performance Test

Performance Testing involved testing the application's performance under various conditions, such as low battery, poor network connectivity, and high device load, as well as measuring the app's response time for tracking updates and map rendering.

Compatibility Test

Compatibility Testing ensured the application works on different Android devices, screen sizes, resolutions, and hardware configurations, and is compatible with different versions of the Android operating system.

Usability Test

Usability Testing evaluated the app's user interface for ease of use, intuitiveness, and consistency, and tested the app's navigation and user interaction elements for user-friendliness.

Security Test

Security Testing verified that user location data is transmitted securely and is not accessible to unauthorized users, and tested the app for vulnerabilities such as data leaks or unauthorized access.

Integration Test

Integration Testing tested the integration of Firebase for backend services, Google Maps API for map rendering, and other third-party services for proper functionality.

Load Test

Load Testing simulated a large number of users to test the app's performance under heavy load and evaluated its ability to handle multiple tracking requests simultaneously.

Stress Test

Stress Testing tested the app's performance under extreme conditions, such as prolonged use or high data load, to identify potential issues and bottlenecks.

Acceptance Test

Acceptance Testing was conducted with end-users to ensure the app meets their requirements and expectations.

BIBLIOGRAPHPY

9. **BIBLIOGRAPHY**

1. W3Schools. (n.d.). HTML Tutorial. Retrieved from https://www.w3schools.com/html/
2. W3Schools. (n.d.). CSS Tutorial. Retrieved from https://www.w3schools.com/css/
3. MDN Web Docs. (n.d.). JavaScript. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript
4. SQLCourse. (n.d.). SQLCourse - Interactive Online SQL Training for Beginners. Retrieved from https://www.sqlcourse.com/
5. PHP: Hypertext Preprocessor. (n.d.). PHP: Hypertext Preprocessor. Retrieved from https://www.php.net/
6. MDN Web Docs. (n.d.). AJAX - Introduction. Retrieved from https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started
7. Stack Overflow. (n.d.). Retrieved from https://stackoverflow.com/

CONCLUSION

10. CONCLUSION

The Android Mobile GPS Tracking Application is a robust and user-friendly solution that offers real-time location tracking, path monitoring, and battery optimization features. Throughout the development process, we focused on creating a highly accurate and efficient tracking app that minimizes battery consumption compared to other similar apps. Our app allows users to view their travel paths at any time, monitor battery levels, and securely encrypt their data for privacy. With a simple and intuitive user interface, users can easily access and utilize the app's features. The accuracy of our tracking system ensures that users can rely on it for precise location information.

In conclusion, the Android Mobile GPS Tracking Application provides a valuable tool for users seeking a reliable and efficient tracking solution for their mobile devices. Its innovative features and focus on user experience make it a standout choice in the field of GPS tracking applications**.**

CODINGS

11. CODINGS

Manifest.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-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.FOREGROUND_SERVICE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"/>


    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@drawable/mapicon"
        android:label="@string/app_name"
        android:roundIcon="@drawable/mapicon"
```

```
        android:supportsRtl="true"
        android:theme="@style/Theme.Maps"
        tools:targetApi="31">

        <meta-data
            android:name = "com.google.android.geo.API_KEY"
            android:value = "@string/google_map_api_key"
        />

        <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>
```

MainActivity.kt

```
package com.example.maps



import android.Manifest
import android.content.Context
import android.content.pm.PackageManager
import android.location.Location
import android.net.wifi.WifiInfo
import android.net.wifi.WifiManager
import android.os.BatteryManager
import android.os.Build
import android.os.Bundle
import android.os.Looper
import android.util.Log
import android.view.MenuItem
import android.widget.Toast
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.PopupMenu
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.drawerlayout.widget.DrawerLayout
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationCallback
import com.google.android.gms.location.LocationRequest
import com.google.android.gms.location.LocationResult
import com.google.android.gms.location.LocationServices
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.OnMapReadyCallback
import com.google.android.gms.maps.SupportMapFragment
```

```
import com.google.android.gms.maps.model.BitmapDescriptorFactory
import com.google.android.gms.maps.model.JointType
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.MarkerOptions
import com.google.android.gms.maps.model.Polyline
import com.google.android.gms.maps.model.PolylineOptions
import com.google.android.gms.maps.model.RoundCap
import com.google.android.material.bottomnavigation.BottomNavigationView
import com.google.android.material.navigation.NavigationView
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


class MainActivity : AppCompatActivity(), OnMapReadyCallback {

    private lateinit var fusedLocationClient: FusedLocationProviderClient
    private var mGoogleMap: GoogleMap? = null
    private var lastKnownLocation: Location? = null
    private var isTrackingEnabled: Boolean = false
    private lateinit var locationCallback: LocationCallback
    private lateinit var reference: DatabaseReference
    private var polyline: Polyline? = null

    private lateinit var popupMenu: PopupMenu
    private lateinit var bottomNavigationView: BottomNavigationView

    lateinit var drawtoggle: ActionBarDrawerToggle
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val drawerLayout = findViewById<DrawerLayout>(R.id.drawer_Layout)
        val navView = findViewById<NavigationView>(R.id.navView)


        drawtoggle = ActionBarDrawerToggle(this,
drawerLayout,R.string.open,R.string.close)
        drawerLayout.addDrawerListener(drawtoggle)
        drawtoggle.syncState()
        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        navView.setNavigationItemSelectedListener {
            when(it.itemId){
                R.id.item1 -> Toast.makeText(applicationContext,"its item
1",Toast.LENGTH_SHORT).show()

                R.id.item2 -> Toast.makeText(applicationContext,"its item
2",Toast.LENGTH_SHORT).show()

                R.id.item3 -> Toast.makeText(applicationContext,"its item
3",Toast.LENGTH_SHORT).show()
            }
            true
        }


        bottomNavigationView = findViewById(R.id.bottomNavigationView)
```

```
        bottomNavigationView.setOnItemSelectedListener { item ->
            when (item.itemId) {
                R.id.maptype -> {
                    // Handle map type selection
                    true
                }
                R.id.ring -> {
                    // Handle ring selection
                    true
                }

                R.id.Batterypower -> {
                    val ref = FirebaseDatabase.getInstance().getReference("RealmeRMX2027")

                    getBatteryAndSSID(ref) { batteryLevel, ssid ->
                        // Use the batteryLevel and ssid values here
                        Log.d("FirebaseData", "Battery Level: $batteryLevel, SSID: $ssid")
                        item.title = "$batteryLevel %"
                    }

                    true
                }

                R.id.NetWork -> {
                    // Handle network selection
                    val ref =
FirebaseDatabase.getInstance().getReference("GoogleAOSPonIAEmulator")

                    getBatteryAndSSID(ref) { batteryLevel, ssid ->
                        // Use the batteryLevel and ssid values here
```

```
                Log.d("FirebaseData", "Battery Level: $batteryLevel, SSID: $ssid")
                item.title = "$ssid"
            }

            true
            true
        }
        R.id.devices -> {

            true
        }
        else -> false
    }
}

fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
val deviceName = getDeviceName().replace("[^a-zA-Z0-9]".toRegex(), "")
reference = FirebaseDatabase.getInstance().getReference(deviceName)


if (ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_FINE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_COARSE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED
) {
    // Request permissions if not granted
    ActivityCompat.requestPermissions(
        this,
```

```
            arrayOf(
                Manifest.permission.ACCESS_FINE_LOCATION,
                Manifest.permission.ACCESS_COARSE_LOCATION
            ),

            REQUEST_LOCATION_PERMISSION
        )
    } else {
        getCurrentLocation(this)
    }

    val mapFragment = supportFragmentManager.findFragmentById(R.id.mapFragment)
as SupportMapFragment
    mapFragment.getMapAsync(this)









}


override fun onOptionsItemSelected(item: MenuItem): Boolean {

    if(drawtoggle.onOptionsItemSelected(item)){
        return true
    }

    return super.onOptionsItemSelected(item)
```

```
    }


    private fun createLocationCallback() {
        locationCallback = object : LocationCallback() {
            override fun onLocationResult(locationResult: LocationResult) {
                locationResult ?: return
                val batteryPercentage = getBatteryPercentage(applicationContext)
                val wifiSSID = getWifiSSID(applicationContext)

                for (location in locationResult.locations) {
                    if (isTrackingEnabled) {
                        saveLocationToFirebase(reference,location.latitude,
location.longitude,batteryPercentage,wifiSSID)
                    }
                }
            }
        }
    }

    private fun showPopupMenu() {
        val databaseReference =
FirebaseDatabase.getInstance().reference.child("RealmeRMX2027")

        databaseReference.addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                popupMenu.menu.clear()
                for (nodeSnapshot in dataSnapshot.children) {
                    val nodeName = nodeSnapshot.key ?: ""
                    popupMenu.menu.add(nodeName)
                }
                popupMenu.show()
```

```
        }

        override fun onCancelled(databaseError: DatabaseError) {
            // Handle database error
        }
    })
}


override fun onResume() {
    super.onResume()
    startLocationUpdates()
}

override fun onPause() {
    super.onPause()
    stopLocationUpdates()
}

private fun startLocationUpdates() {
    isTrackingEnabled = true
    if (ActivityCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_COARSE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        return
    }
```

```
    createLocationCallback()

    val locationRequest = LocationRequest.create().apply {
        interval = 4000
        fastestInterval = 2000
        priority = LocationRequest.PRIORITY_HIGH_ACCURACY
    }

    fusedLocationClient.requestLocationUpdates(
        locationRequest,
        locationCallback,
        Looper.getMainLooper()
    )
}

private fun stopLocationUpdates() {
    isTrackingEnabled = false
    fusedLocationClient.removeLocationUpdates(locationCallback)
}

override fun onMapReady(googleMap: GoogleMap) {
    mGoogleMap = googleMap

    if (lastKnownLocation != null) {
        val latLng = LatLng(lastKnownLocation!!.latitude, lastKnownLocation!!.longitude)
        mGoogleMap?.addMarker(MarkerOptions().position(latLng).title("You").icon(
            BitmapDescriptorFactory.fromResource(R.drawable.pin)))
        mGoogleMap?.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, 15f))
    }

    drawPolylineFromFirebase()
```

```
    val ref = FirebaseDatabase.getInstance().getReference("RealmeRMX2027")

    getLocationData(ref) { latitude, longitude ->
        // Inside this block, you have access to the latitude and longitude values
        Log.d("LocationData", "Latitude: $latitude, Longitude: $longitude")

        // Call your function to mark the location here
        markLocation(mGoogleMap!!,latitude, longitude)
    }

}

private fun drawPolylineFromFirebase() {
    val database = FirebaseDatabase.getInstance()
    val reference = database.getReference("locations").child("PathLocation")


    reference.addListenerForSingleValueEvent(object : ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            val coordinatesList = mutableListOf<LatLng>()

            for (snapshot in dataSnapshot.children) {
                val latitude = snapshot.child("latitude").getValue(Double::class.java)
                val longitude = snapshot.child("longitude").getValue(Double::class.java)

                if (latitude != null && longitude != null) {
                    coordinatesList.add(LatLng(latitude, longitude))
                }
            }

            if (coordinatesList.size >= 2) {
                val polylineOptions = PolylineOptions().clickable(true)
```

```
                for (coordinate in coordinatesList) {
                    polylineOptions.add(coordinate)
                }

                polyline?.remove()
                polyline = mGoogleMap?.addPolyline(polylineOptions
                    .endCap(RoundCap())
                    .startCap(RoundCap())
                    .color(ContextCompat.getColor(this@MainActivity, R.color.black))
                    .width(12f)
                    .jointType(JointType.ROUND)
                )
            }
        }

        override fun onCancelled(databaseError: DatabaseError) {
            Log.w(TAG, "loadPost:onCancelled", databaseError.toException())
        }
    })
}

private fun getCurrentLocation(mainActivity: MainActivity) {
    if (ActivityCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_COARSE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        return
```

```
        }
        fusedLocationClient.lastLocation
            .addOnSuccessListener { location: Location? ->
                location?.let {
                    lastKnownLocation = location
                    val latitude = location.latitude
                    val longitude = location.longitude
                    Toast.makeText(
                        this,
                        "Latitude: $latitude, Longitude: $longitude",
                        Toast.LENGTH_SHORT
                    ).show()
                }
            }
            .addOnFailureListener { e ->
                Toast.makeText(
                    this,
                    "Failed to get location: ${e.message}",
                    Toast.LENGTH_SHORT
                ).show()
            }
    }

    fun getBatteryPercentage(context: Context): Int {
        val batteryManager = context.getSystemService(Context.BATTERY_SERVICE) as
BatteryManager
        return
batteryManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY)
    }

    fun getWifiSSID(context: Context): String {
        val wifiManager =
```

```
context.applicationContext.getSystemService(Context.WIFI_SERVICE) as WifiManager
        val wifiInfo: WifiInfo = wifiManager.connectionInfo
        return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
            wifiInfo.ssid
        } else {
            wifiInfo.ssid.replace("\"", "")
        }
    }


    fun getDeviceName(): String {
        val manufacturer = Build.MANUFACTURER
        val model = Build.MODEL
        return if (model.startsWith(manufacturer)) {
            model.capitalize()
        } else {
            manufacturer.capitalize() + " " + model
        }
    }

    fun markLocation(map: GoogleMap, latitude: Double, longitude: Double) {
        val location = LatLng(latitude, longitude)
        map.addMarker(MarkerOptions().position(location))
        map.moveCamera(CameraUpdateFactory.newLatLngZoom(location, 15f))
    }
    data class LocationData(val latitude: Double, val longitude: Double)

    companion object {
        const val REQUEST_LOCATION_PERMISSION = 1001
        const val TAG = "MainActivity"
    }
}
```

MainActivity_Layout .Xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.drawerlayout.widget.DrawerLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_Layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">



        <androidx.fragment.app.FragmentContainerView
            android:id="@+id/mapFragment"
            class="com.google.android.gms.maps.SupportMapFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            tools:layout_editor_absoluteX="41dp"
            tools:layout_editor_absoluteY="0dp" />

        <Button
            android:id="@+id/mapmenu"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@null"
            android:text="mapviw"
            app:layout_constraintEnd_toEndOf="parent"
```

```
        app:layout_constraintStart_toStartOf="parent"
        tools:ignore="MissingConstraints"
        tools:layout_editor_absoluteY="603dp" />

    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:layout_width="match_parent"
        android:layout_height="80dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0"
        app:menu="@menu/bottom_menu"
        android:layout_gravity="bottom"
        android:id="@+id/bottomNavigationView"/>

</androidx.constraintlayout.widget.ConstraintLayout>

<com.google.android.material.navigation.NavigationView
    android:id="@+id/navView"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    app:headerLayout="@layout/header"
    app:menu="@menu/nav_menu"
    android:layout_gravity="start"
    android:fitsSystemWindows="true" />
```

Strings.xml

```
<resources>
   <string name="app_name">maps</string>
   <string name="google_map_api_key">AIzaSyBUcpB_uFUzZHA-
3MkRvcxabIeD2E44ihc</string>
   <string name="open">open</string>
   <string name="close">close</string>

</resources>
```

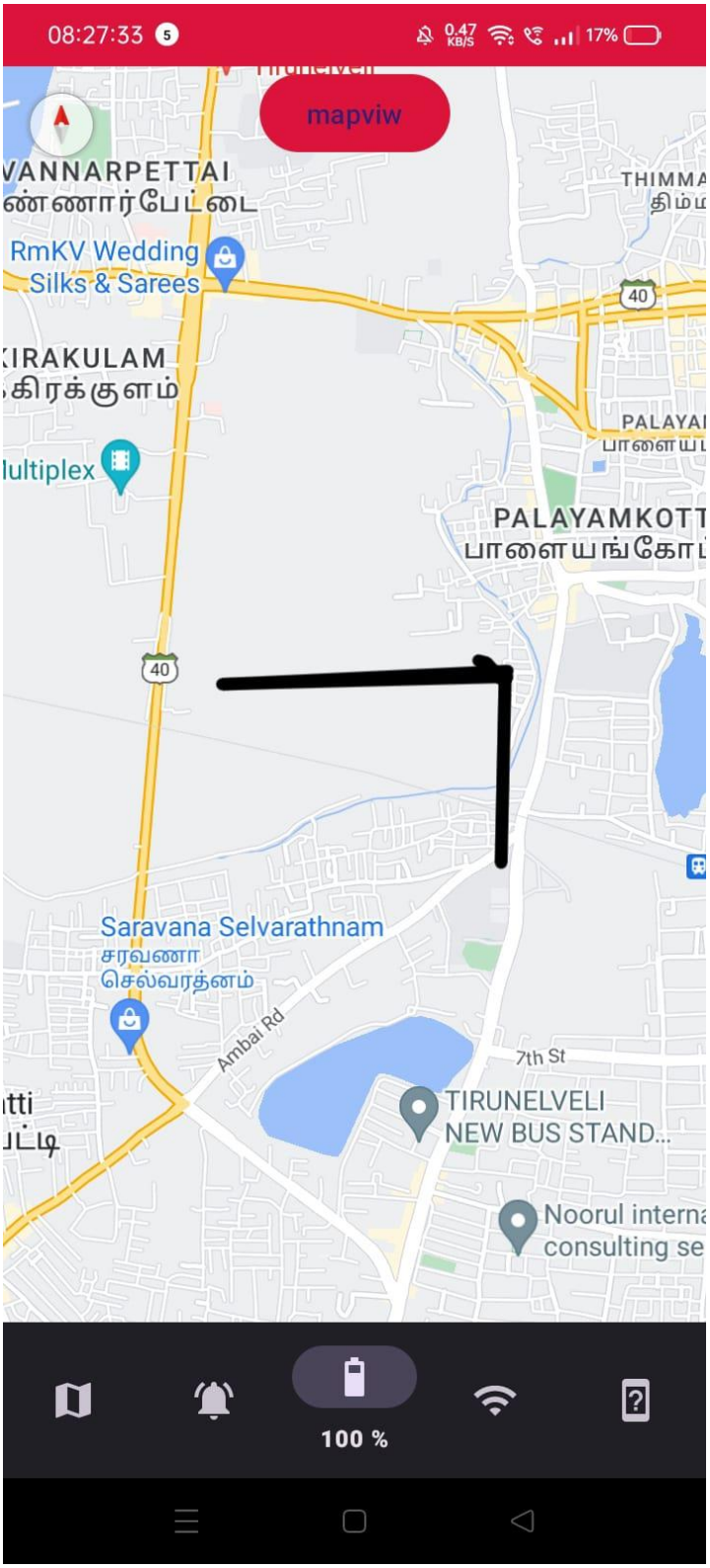
Colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="black">#FF000000</color>
   <color name="white">#FFFFFFFF</color>
   <color name="aqua">#66CDAA</color>
   <color name="grey">#95B9C7</color>
</resources>
```

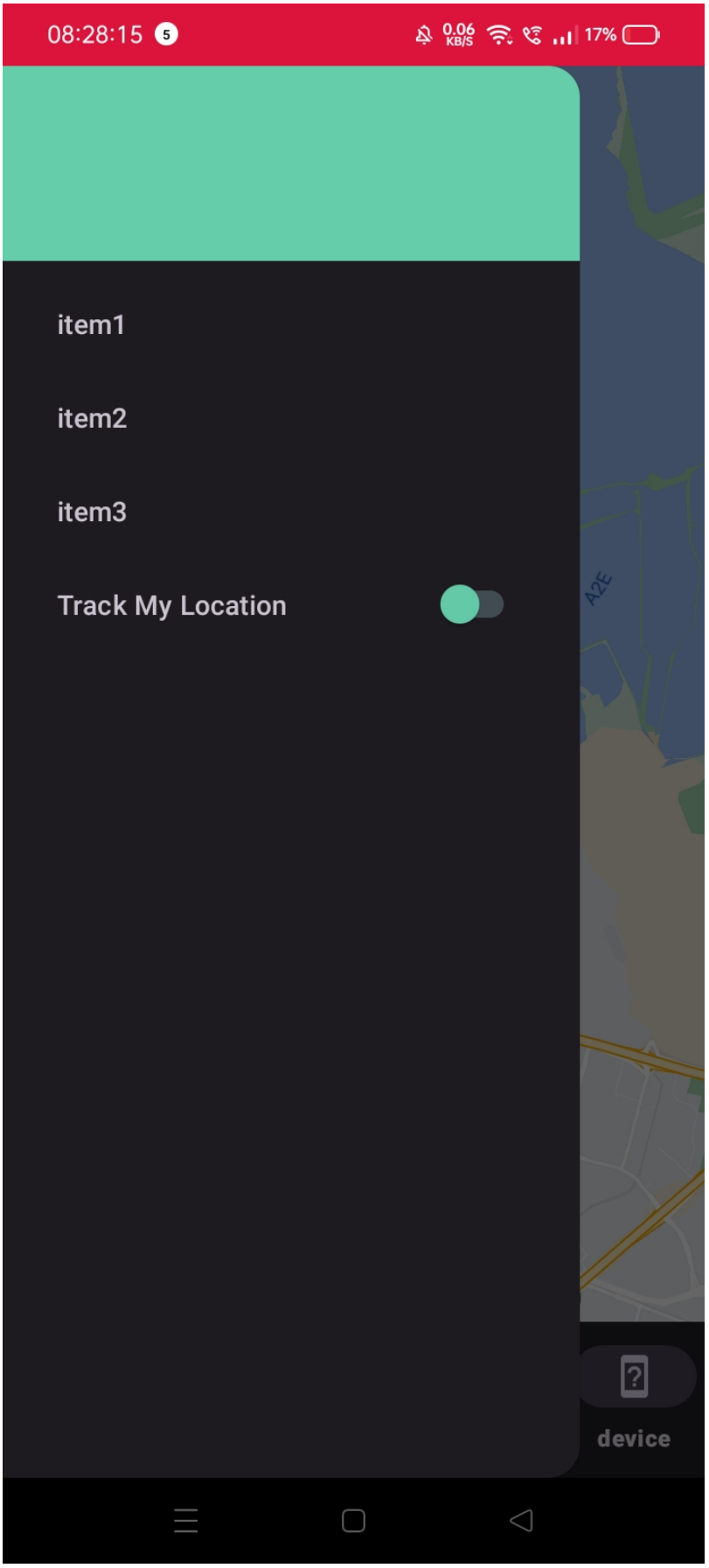
Hearder.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="100dp"
   android:background="@color/aqua"/>
```

12. SAMPLE SCREENSHOTS




08:27:51
17%
mapviw
Shoreline at Mountain View
Shoreline Golf Links
Kite Lot
Shoreline Amphitheatre
Parking Lot C
N Shoreline Blvd
Googleplex
Google Visitor Experience
E Charleston Rd
Shashi Hotel Mountain View, an Urban Resort
Zareen's
Old Middlefield Way
Spring St
NetWork


08:28:15
5
0.06
KB/S
17%
item1
item2
item3
Track My Location
A2E
device


thank
you