Alma Mater Studiorum • Università di Bologna

SCUOLA DI INGEGNERIA

Corso di laurea in Ingegneria Informatica

ANALISI E UTILIZZO DI FRAMEWORK DI LOCALIZZAZIONE IN KOTLIN PER ANDROID

Relatore:

Prof. PAOLO BELLAVISTA

Candidato:

FILIPPO MANFREDA

II Sessione

Anno Accademico 2019-2020

Indice

Introduzione

Da circa dieci anni il mercato dei dispositivi mobili, in particolar modo quello degli smartphone, è in rapida e costante crescita per quanto riguarda le vendite e la distribuzione in tutto il mondo. Questo andamento non sembra diminuire e la conseguenza diretta è la nascita di nuove case produttrici e, con queste, anche nuove software house, ossia aziende specializzate nello sviluppo di software e applicazioni utilizzabili negli smartphone.
Nel mondo degli smartphone molti sono i sistemi operativi installabili su tali dispositivi: Android, iOS, Windows , BlackBarry OS e altri minori. Di questi sono due quelli che occupano la porzione maggiore di mercato, Android e iOS. Infatti dal 2012 ad oggi questi due sistemi operativi sono cresciuti sempre di più, fino a raggiungere una quota di mercato maggiore del 99%[1], con prevalenza del sistema operativo Android.
E' possibile affermare che ormai ognuno di noi possiede uno smartphone, il quale offre elevate capacità di elaborazione considerando le dimensioni e, di maggiore interesse per l'utente "medio", permette di usufruire di un grande numero di servizi, disponibili tramite determinate applicazioni.
Questi servizi appartengono ai più disparati ambiti, come la multimedialità, la comunicazione, il gaming, la finanza, lo studio e molti altri ancora.

Un'aspetto che risulta molto interessante è la capacità degli smartphone di ottenere la propria posizione geografica (geolocalizzazione), la quale è diventata una necessità per poter usufruire di molti servizi legati allo spostamento (vedi Google Maps, Waze etc) e al ritrovamento di determinate informazioni in base al luogo in cui ci si trova. Comprendere questo aspetto risulta quindi molto importante per poter progettare software funzionale e che rispetti le specifiche e le necessità del tempo contemporaneo. Al termine geolocalizzazione si associa subito l'idea al GPS e all'impiego dei satelliti; in realtà è bene notare che la determinazione della posizione tramite satellite non è l'unica tecnica applicabile ai dispositivi mobili. Infatti anche l'utilizzo delle reti WiFi o delle celle delle reti degli operatori mobili permettono la localizzazione, anche se al momento, l'utilizzo del GPS, risulta la tecnica che fornisce il maggiore livello di accuratezza per tutte le applicazioni orientate alla navigazione.

[1] https://www.statista.com

Considerando invece lo sviluppo di applicazioni che offrono funzionalità di localizzazione e possibilità di visualizzare una mappa, si possono utilizzare diverse librerie di codice e API per implementare tale servizio. Tra queste, le Google Maps API risultano essere le più utilizzate, richiedendo tuttavia un processo di registrazione presso Google e ricordando che essendo proprietarie e non open source non è possibile contribuire allo sviluppo di tale software.

Partendo da questo contesto l'elaborato della tesi si focalizzerà sullo studio, l'analisi e l'utilizzo di un framework di localizzazione per sviluppare un'applicazione eseguibile su smartphone Android e, successivamente, effettuare analisi sperimentali sulle performance e consumi di tale applicazione.
In particolare, si è scelto Android poiché è il sistema operativo principalmente utilizzato al momento negli smartphone e di maggiore interesse personale, la cui piattaforma verrà descritta in modo dettagliato nella tesi.
Per sviluppare un'applicazione è poi necessario conoscerne le caratteristiche e i suoi elementi fondamentali e passando alla reale implementazione, si deve scegliere il linguaggio di programmazione da utilizzare per scrivere il codice dell'app. Per quest'ultimo punto è stato scelto Kotlin, un linguaggio di programmazione general purpose, multi-paradigma, open source sviluppato dall'azienda di software JetBrains, le cui caratteristiche saranno descritte e confrontate con quelle del linguaggio Java.
Ponendo l'attenzione sui dettagli implementativi, si è deciso di studiare la classe LocationManager del pacchetto "android.location" per l'accesso ai servizi di localizzazione del sistema e osmdroid, una libreria open source basata sul progetto OpenStreetMap, per rappresentare la mappa del mondo all'interno dell'applicazione.
Prima di rilasciare una versione dell'applicazione per l'uso definitivo è necessario che tutti i requisiti siano soddisfatti e inoltre che la sua esecuzione non comprometta la stabilità del dispositivo. Per questo verrà misurata la latenza, per quanto riguarda il ritrovamento della posizione, e saranno analizzati i consumi in termini di CPU, RAM e batteria.

L'elaborato di tesi è composto di quattro capitoli, il cui contenuto generale è indicato nelle successive righe.
Il primo descriverà inizialmente gli aspetti relativi alla piattaforma Android, includendo anche gli elementi fondamentali di un'applicazione e, successivamente, si occupa di mostrare quali tipologie di localizzazione sono possibili sui dispositivi mobili.

Nel secondo varrà introdotto nella prima parte il linguaggio di programmazione Kotlin, descrivendone le caratteristiche e confrontandolo con il linguaggio Java, mentre nella seconda parte sarà analizzata nel dettaglio la classe LocationManager, utilizzata per implementare la localizzazione.
Il terzo capitolo si occuperà di mostrare in primo luogo il progetto dell'applicazione, descrivendone i requisiti, i suoi macro moduli e l'interfaccia grafica; successivamente verrà spiegato in cosa consiste il progetto OpenStreetMap.
Infine, il quarto capitolo si concentrerà sull'implementazione, descrivendo le parti di codice più interessanti, e sulle analisi sperimentali effettuate durante l'esecuzione dell'applicazione in due diversi dispositivi, per monitorare: latenza, utilizzo di CPU, utilizzo di memoria RAM e consumo di batteria.

Capitolo 1 : Android e la localizzazione nei dispositivi mobili

In questo primo capitolo si introdurrà l'ambito del mondo Android con particolare attenzione al suo sviluppo e organizzazione strutturale come sistema operativo per dispositivi mobili, incentrandosi sugli smartphone e sulle cosiddette "app", software installabile su tali dispositivi. Successivamente si affronterà il tema della geolocalizzazione che è possibile effettuare su questi dispositivi indicando le varie modalità presenti ad oggi.

La parola smartphone, letteralmente telefono intelligente, indica una determinata classe di dispositivi mobili di medie-piccole dimensioni considerabili elaboratori, quindi con capacità di calcolo, memoria e connessione dati.

Il primo vero smartphone ad essere stato commercializzato è stato "Simon", sviluppato dalla azienda IMB, il quale presentava dieci applicazioni pre-installate senza ancora la presenza di un cosiddetto "app store", negozio da cui è possibile scaricare le app presenti nel suo catalogo.

L'infografica a lato mostra l'evoluzione degli smartphone fino alla nascita del primo smartphone con sistema operativo Android.

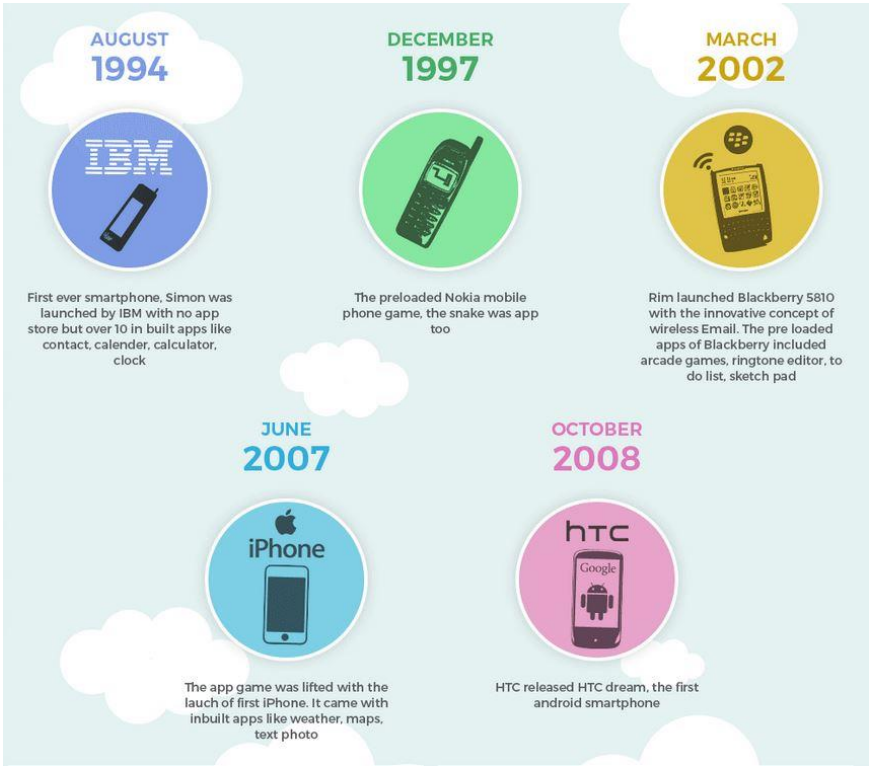

Figura 1 - Infografica sugli smartphone

Nel mondo odierno ormai tutti hanno con sé uno smartphone e un aspetto che risulta sempre più presente, importante e di grande utilizzo legato a questo dispositivo è la possibilità di localizzazione.

Grazie alla localizzazione sono nati molti servizi che oggi vengono costantemente utilizzati da tutti noi: è per esempio possibile trovare attività e servizi nelle vicinanze, come bar, ristoranti, cinema, negozi e altri; è possibile poi farci guidare dal nostro smartphone per arrivare ad una determinata meta (es. Google Maps); è possibile condividere con amici e parenti la nostra posizione corrente per informarli; un altro utilizzo è quello delle applicazioni in realtà aumentata, le quali si basano principalmente sull'utilizzo delle informazioni provenienti dalla fotocamera del nostro dispositivo ma, la conoscenza della nostra posizione permette di personalizzare l'esperienza di utilizzo di tali app; infine tale capacità risulta molto utile nel caso in cui una persona si perda e guardandosi intorno non riesce più a capire dove si trova, situazione che può presentarsi quando si fanno dei trekking in montagna o in altre località in cui i segnali non sempre sono presenti e costanti.

Ovviamente questi sono solo alcuni degli utilizzi che oggi vengono fatti con la conoscenza della nostra posizione corrente, molti altri poi stanno nascendo.

Con la crescita esponenziale che si è verificata in questo ambito è cresciuta anche la necessità di proteggere tali informazioni, poiché un individuo con scopi malevoli potrebbe risultare facilitato nel suo intento grazie alla conoscenza di queste informazioni.

Risulta quindi importante conoscere e apprendere come tali tecnologie funzionano e come possono essere utilizzate, per questo nel capitolo presente si affronteranno questi due temi appena introdotti.

1.1 Android

Android è un sistema operativo sviluppato da Google LLC e basato sul kernel Linux, progettato principalmente per sistemi embedded quali smartphone e tablet, utilizzato inoltre per televisori (Android TV), automobili (Android Auto), orologi da polso (Wear OS) e altri dispositivi.
Distribuito sotto i termini della licenza libera Apache 2.0 riservandosi di non-includere software coperto da licenze copyleft, è possibile usare il software per ogni scopo, distribuirlo, modificarlo e distribuire versioni modificate di esso.
Il suo sviluppo non è terminato ma prosegue attraverso l'Android Open Source Project, il quale rappresenta un software libero con l'esclusione di diversi firmware non-liberi inclusi per i produttori di dispositivi e delle cosiddette "Google Apps", come ad esempio Google Play.

Figura 2 Logo Android

L'idea alla base di Android nasce nell'ottobre 2003 quando Andy Rubin insieme ad altre 3 persone fondano una società, la Android Inc. per lo sviluppo di quello che Rubin definì «... dispositivi cellulari più consapevoli della posizione e delle preferenze del loro proprietario».[2]
Il 17 agosto 2005 Google LLC ha acquisito l'azienda, in vista del fatto che la società di Mountain View desiderava entrare nel mercato della telefonia mobile. È in questi anni che il team di Rubin comincia a sviluppare un sistema operativo per dispositivi mobili basato sul kernel Linux. La presentazione ufficiale di Android, anche detto "robottino verde" visto il suo logo, avvenne il 5 novembre 2007 dalla OHA (Open Handset Alliance), un consorzio di aziende del settore Hi Tech.

Il primo dispositivo equipaggiato con Android che venne lanciato sul mercato fu l'HTC Dream, il 22 ottobre 2008.
Dal 2008 vengono rilasciati costantemente aggiornamenti per migliorarne le prestazioni, aggiungere nuove funzionalità e per eliminare eventuali problemi di sicurezza delle precedenti versioni.

[2] https://it.wikipedia.org/wiki/Android

Figura 3 - Versioni Android ad oggi

1.1.1 Interfaccia e Hardware

L'interfaccia utente di Android è basata sul concetto di manipolazione diretta per cui si utilizzano gli ingressi mono e multi-touch come strisciate, tocchi e pizzichi sullo schermo per manipolare gli elementi visibili sullo stesso. La risposta all'input dell'utente è stata progettata per essere immediata e tentare di fornire un'interfaccia fluida. Sensori hardware interni come accelerometri, giroscopi e sensori di prossimità sono utilizzati da alcune applicazioni per rispondere alle azioni da parte dell'utente.

La cosiddetta Homescreen (in italiano "Schermata principale", identificata con un'icona che rappresenta una casa) è simile al desktop di Windows ed è la schermata principale in cui ci si trova appena il dispositivo è stato avviato, oppure premendo il tasto Home. La schermata principale di Android è in genere occupata sia dalle icone delle applicazioni sia dai widget, una sorta di gadgets con varie funzioni.

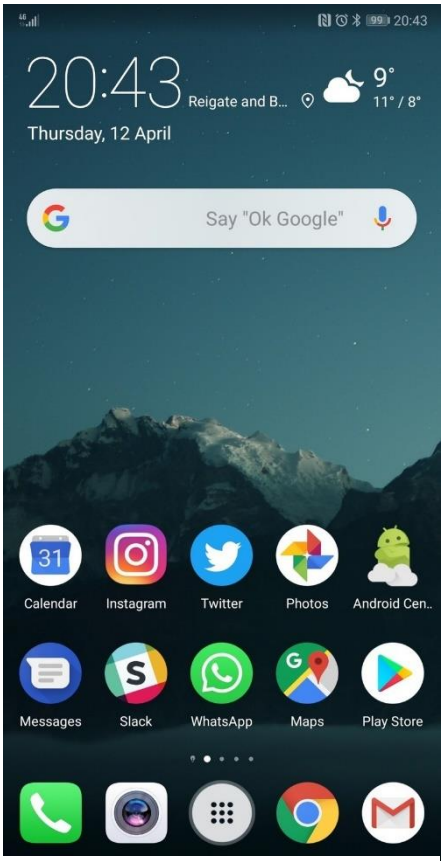

Figura 4 - Homescreen

Componente classico del mondo Android è il Launcher (letteralmente "lanciatore") ovvero l'applicazione di sistema che sovrintende e gestisce essenzialmente la schermata principale e, secondariamente, le scorciatoie (shorcut), il cassetto delle applicazioni (app drawer), la barra inferiore e la barra di stato, il menù notifiche e le impostazioni rapide. Oltre a quello predefinito esistono numerosi launcher di terze parti che offrono una vasta gamma di personalizzazioni.

La piattaforma usa la libreria dedicata SGL (Standard Graphic Library) per la grafica bidimensionale e supporta lo standard OpenGL ES 2.0 per la grafica tridimensionale.

Per quanto riguarda l'hardware, viene utilizzata l'architettura a microprocessore ARM grazie al suo basso consumo di batteria in rapporto alle prestazioni che offre.

I dispositivi Android incorporano molti componenti hardware, alcuni diventati standard per determinate classi di dispositivi, tra cui fotocamere o videocamere, GPS, sensori di orientamento, controlli di gioco dedicati, accelerometri, giroscopi, barometri, magnetometri, sensori di prossimità, sensori di pressione e altri.

1.1.2 Architettura della Piattaforma

Android è un software con un'architettura a più livelli, open source, basato su Linux, creato per un'ampia gamma di dispositivi. Il diagramma sottostante mostra i componenti principali della piattaforma Android.

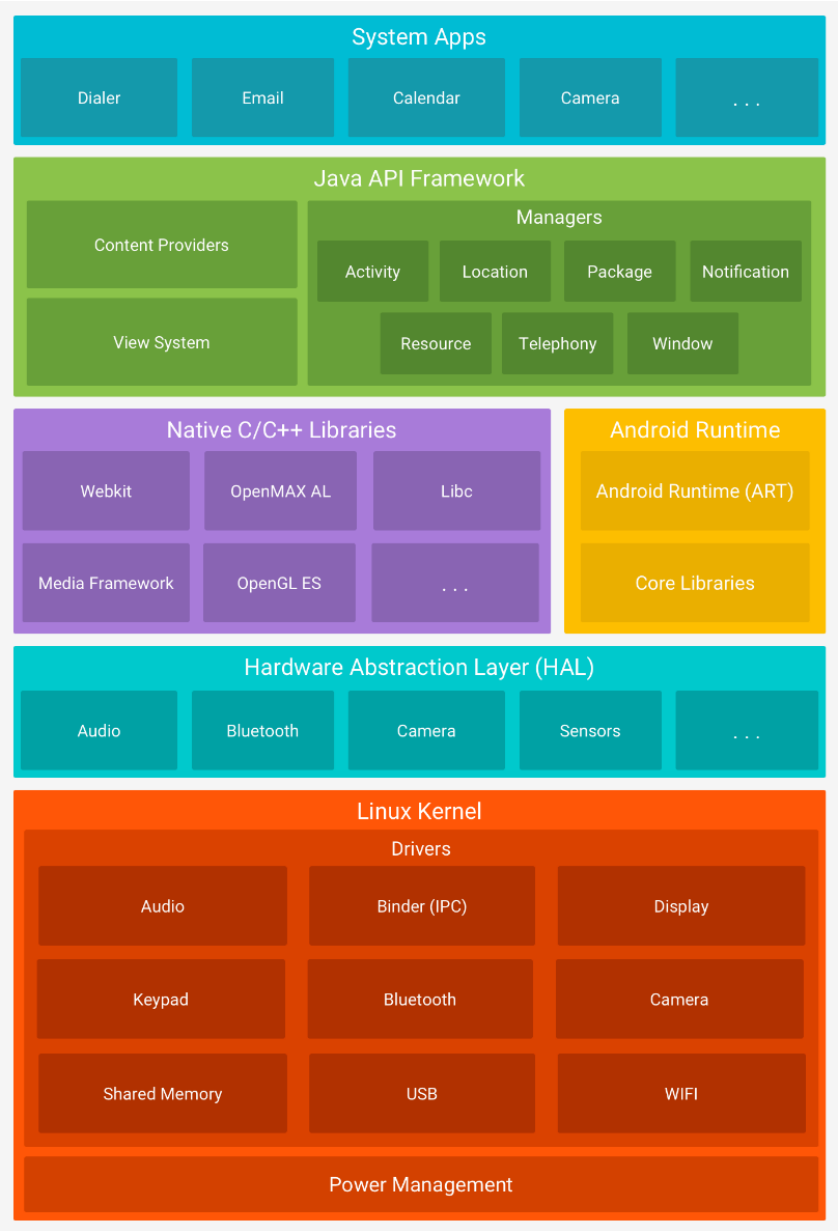

Figura 5 - Piattaforma Android

1.1.2.1 Kernel Linux

Il fondamento della piattaforma Android è il kernel Linux. Android Runtime (ART) si affida al kernel Linux per l'utilizzo di funzionalità sottostanti come il threading e la gestione della memoria di basso livello.[3]

L'uso di un kernel Linux consente ad Android di sfruttare le principali funzionalità di sicurezza messe a disposizione e consente ai produttori di dispositivi di sviluppare driver hardware per un kernel noto, quale è Linux.

[3] https://developer.android.com/guide/platform

1.1.2.2 Hardware Abstraction Layer (HAL)

Il livello di astrazione hardware (HAL) fornisce interfacce standard che espongono i componenti hardware del dispositivo al framework API Java di livello superiore. L'HAL è costituito da più moduli di libreria, ognuno dei quali implementa un'interfaccia per un tipo specifico di componente hardware, per esempio il modulo WiFi o videocamera. Quando un'API del framework effettua una chiamata per accedere all'hardware del dispositivo, il sistema Android carica il modulo della libreria per quel componente hardware e lo rende così utilizzabile al livello superiore.

1.1.2.3 Android Runtime

Per i dispositivi che utilizzano Android versione 5.0 (livello API, Application Programming Interface, 21) o superiore, ogni app viene eseguita con la propria istanza di Android Runtime (ART), il componente che si occupa dell'esecuzione delle operazioni richieste. ART è scritto per eseguire più macchine virtuali su dispositivi a bassa memoria eseguendo file DEX, un formato bytecode progettato appositamente per Android e ottimizzato per un minimo utilizzo di memoria. Con ART si compilano i sorgenti Java in bytecode DEX, i quali possono essere eseguiti sulla piattaforma Android.

Alcune delle principali caratteristiche di ART sono:

- Compilazione ahead-of-time (AOT), che esegue l'intera compilazione del codice durante l'installazione dell'app e non durante l'esecuzione stessa del software
- Compilazione just-in-time (JIT), tramite la quale ogni applicazione viene compilata solo in parte dallo sviluppatore e sarà poi di volta in volta compito dell'interprete software eseguire il codice e compilarlo definitivamente in linguaggio macchina in tempo reale, per ogni esecuzione dell'app stessa
- Garbage Collection (GC), la modalità automatica di gestione della memoria, ottimizzata per il rilascio delle risorse
- Su Android 9 (livello API 28) e versioni successive, conversione dei file in formato eseguibile Dalvik (DEX) di un pacchetto di app in codice macchina più compatto.
- Supporto per il debug, eccezioni diagnostiche dettagliate, report sugli arresti anomali e la possibilità di impostare punti di controllo per monitorare campi specifici.

Prima di Android versione 5.0 (livello API 21), Dalvik era “l’esecutore” di Android, il quale utilizzava esclusivamente la compilazione Just-In-Time. Se un’applicazione funziona bene su ART, dovrebbe funzionare anche su Dalvik, ma potrebbe non essere vero il contrario. Android include anche una serie di librerie core di runtime che forniscono la maggior parte delle funzionalità del linguaggio di programmazione Java, incluse alcune funzionalità del linguaggio Java 8.

1.1.2.4 Librerie Native C/C++

All’Interno di Android sono presenti molti componenti e servizi di base scritti in codice che richiede l’utilizzo di librerie native scritte in C e C++.

La piattaforma Android fornisce un framework di API Java per esporre le funzionalità di alcune di queste librerie native alle applicazioni. Ad esempio, si può utilizzare OpenGL ES tramite l'API OpenGL Java del framework Android per aggiungere supporto per il disegno e la manipolazione di grafica 2D e 3D nelle applicazioni.
Se si sviluppa un’applicazione che richiede codice C o C ++, si può utilizzare Android NDK(Native Development Kit) per accedere direttamente ad alcune di queste librerie della piattaforma nativa.

1.1.2.5 Framework Java API

L'intero set di funzioni del sistema operativo Android è disponibile tramite API scritte in linguaggio Java; queste API costituiscono gli elementi fondamentali necessari per creare app Android, semplificando il riutilizzo di componenti e servizi di sistema.

Tali API includono quanto segue:

- Un sistema di visualizzazione ricco ed estensibile che si può utilizzare per creare l'interfaccia utente di una applicazione, inclusi elenchi, griglie, caselle di testo, pulsanti e persino un browser Web incorporabile
- Un Resource Manager, che fornisce accesso a risorse come stringhe, grafica e file di layout localizzati

- Un gestore delle notifiche che consente a tutte le app di visualizzare avvisi personalizzati nella barra di stato
- Un Activity Manager che gestisce il ciclo di vita delle applicazioni e fornisce un “back stack” di navigazione comune
- Provider di contenuti che consentono alle applicazioni di accedere ai dati di altre applicazioni, per permettere quindi la condivisione

1.1.2.6 Applicazioni di sistema

Android viene fornito con una serie di applicazioni di sistema come quelle per e-mail, messaggi SMS, calendari, navigazione in Internet, contatti e altro. Le applicazioni incluse con la piattaforma non hanno uno “privilegio” speciale rispetto alle applicazioni che l'utente sceglie di installare. Quindi un’applicazione di terze parti può diventare il browser Web predefinito, di messaggistica SMS o anche la tastiera predefinita (si applicano alcune eccezioni, come l'applicazione per le impostazioni del sistema che non è possibile cambiare).
Le applicazioni di sistema funzionano sia come applicazioni per gli utenti sia per fornire funzionalità chiave a cui gli sviluppatori possono accedere dalla propria applicazione. Ad esempio, se una applicazione desidera recapitare un messaggio SMS, non è necessario sviluppare tale funzionalità: si può invece invocare qualsiasi applicazione SMS già installata per recapitare un messaggio al destinatario specificato.

1.1.3 Il Mondo delle Applicazioni

Le applicazioni (o app) sono la forma più generica per indicare i software applicativi installabili su Android. Esse possono essere scaricate sia dal catalogo ufficiale Google Play, sia da altri cataloghi, come l'Amazon Appstore, o F-Droid che contiene solo software su licenza libera. Le applicazioni Android possono essere installate anche direttamente a partire da un file APK(Android Package) fornito dal distributore del software.

Per motivi di sicurezza informatica, le applicazioni sono fornite di un sistema di certificazione che verifica l'integrità del pacchetto da installare e la paternità rispetto a un distributore di software accreditato presso Google, anche se è possibile installare applicazioni senza una certificazione dopo consenso esplicito dell'utente.

Le app Android possono essere scritte usando i linguaggi Kotlin, Java e C ++. Gli strumenti Android SDK compilano il codice insieme a tutti i file di dati e risorse in un APK, un pacchetto Android, che è un file di archivio con un suffisso .apk. Un file APK contiene tutti i dati di un'app Android ed è il file utilizzato dai dispositivi Android per installare quest'ultima.

Per quanto riguarda la sicurezza, ogni app Android vive nel proprio "sandbox", protetto dalle seguenti funzionalità:

- Il sistema operativo Android è un sistema Linux multiutente in cui ogni app è rappresentata come un utente diverso del sistema.
- Per impostazione predefinita, il sistema assegna a ciascuna app un ID utente Linux univoco (l'ID viene utilizzato solo dal sistema ed è sconosciuto all'app).
- Ogni processo ha la propria macchina virtuale (VM), quindi il codice di un'app viene eseguito separatamente da altre app.
- Per impostazione predefinita, per ogni esecuzione di un'app si crea un nuovo processo Linux. Il sistema Android avvia il processo quando è necessario eseguire uno dei componenti dell'app, quindi interrompe il processo quando non è più necessario o quando il sistema deve recuperare memoria per altre app.

Il sistema Android implementa il principio del privilegio minimo: ogni app, per impostazione predefinita, ha accesso solo ai componenti necessari per svolgere il proprio lavoro e non ad altro. Ciò crea un ambiente molto sicuro in cui un'app non può accedere a parti del sistema per le quali non è stata autorizzata. Tuttavia, ci sono modi in cui un'app può condividere dati con altre app e modi per un'app di accedere ai servizi di sistema: è possibile disporre che due app condividano lo stesso ID utente, nel qual caso sono in grado di accedere reciprocamente ai file collegati a tale ID. Per conservare le risorse di sistema, le app con lo stesso ID utente possono anche organizzare l'esecuzione di esse nello stesso processo Linux e condividere la stessa VM. In questo caso le app devono essere firmate con lo stesso certificato per poter essere utilizzate.
Un'app può richiedere l'autorizzazione per accedere ai dati del dispositivo, come la posizione geografica di quest'ultimo, la fotocamera e la connessione Bluetooth. L'utente deve concedere esplicitamente queste autorizzazioni.

Altri concetti importanti per quanto riguarda il mondo delle applicazioni sono:

- I componenti principali del framework che definiscono una applicazione.
- Il file Manifest in cui si dichiarano i componenti e le funzionalità del dispositivo richieste per il funzionamento dell'app.
- Risorse separate dal codice dell'applicazione che consentono di ottimizzarne il comportamento in modo corretto per una varietà di configurazioni del dispositivo.

1.1.3.1 Componenti di un'applicazione

Ogni applicazione Android, indipendentemente dalla finalità che si prefigge, affida le sue funzionalità a diversi componenti.

Questi componenti sono i mattoni fondamentali di un'app Android. Ogni componente è un punto di accesso attraverso il quale il sistema o un utente possono accedere alla app. Alcuni componenti dipendono da altri ed esistono affinché tale applicazione possa integrarsi alla perfezione nell'ecosistema Android.

Esistono quattro diversi tipi di componenti:

- Activity
- Service
- Broadcast receiver
- Content provider

Ogni tipo di componente ha uno scopo distinto e ha un ciclo di vita distinto che definisce come il componente viene creato e distrutto. Le seguenti sezioni descrivono i quattro tipi di componenti dell'applicazione.

Activity

L'activity(o attività) è il componente fondamentale per l'interazione con l'utente, rappresenta una singola schermata con un'interfaccia utente.
Ogni volta che si usa un'applicazione generalmente si interagisce con una o più "pagine" mediante le quali si consultano dati o si immettono input. Le activity nel loro complesso costituiscono il flusso in cui l'utente si inoltra per sfruttare le funzionalità messe a disposizione dall'applicazione.[4]

Questo richiede pertanto non solo la realizzazione delle interfacce grafiche stesse, ma anche una corretta progettazione della navigazione tra di esse offrendo la possibilità di sfogliarne i contenuti e risalirli gerarchicamente in maniera coerente.

Una activity permette di gestire in modo semplificato le interazioni tra il sistema e l'applicazione, in particolare permette di:

- Tenere traccia di ciò che l'utente attualmente necessita (ciò che è sullo schermo) per garantire che il sistema continui a eseguire il processo che ospita tale activity.
- Sapere che i processi utilizzati in precedenza contengono elementi a cui l'utente può tornare (activity interrotte) e quindi dare priorità più alta per mantenere tali processi.

[4] https://www.html.it/guide/guida-android/

- Aiutare l'app a gestire le interazioni precedenti in modo che l'utente possa tornare alle attività con lo stato precedente ripristinato.
- Fornire un modo alle app di implementare i flussi utente tra le diverse activity e per permettere al sistema di coordinare questi flussi.

Nella pratica una Activity si implementa come sottoclasse della classe “android.app.Activity”.

Service

Un servizio è un componente per mantenere un'app in esecuzione in background, per motivi legati al su uso e funzionamento. In genere è utilizzato per eseguire operazioni di lunga durata o eseguire operazioni per processi remoti e non fornisce un'interfaccia utente. Ad esempio, un servizio potrebbe riprodurre musica in background mentre l'utente si trova in un'altra app, oppure potrebbe recuperare dati sulla rete senza bloccare l'interazione dell'utente con una activity. Un altro componente, ad esempio una activity, può avviare il servizio e lasciarlo funzionare o associarsi ad esso per interagirci in un secondo momento.

Esiste una suddivisione semantica dei servizi: servizi avviati e servizi associati.

I servizi avviati indicano al sistema di mantenerli in esecuzione fino al completamento della loro esecuzione: questo potrebbe essere necessario per sincronizzare alcuni dati in background o riprodurre musica anche dopo che l'utente ha lasciato l'app. La sincronizzazione dei dati in background o la riproduzione di musica rappresentano anche due diversi tipi di servizi avviati che modificano il modo in cui il sistema li gestisce:

- La riproduzione musicale è qualcosa di cui l'utente è direttamente a conoscenza, quindi l'app informa il sistema che essa deve rimanere in esecuzione, notificando direttamente anche l’utente; un servizio di questo tipo deve rimanere in esecuzione e non può essere bloccato perché esplicitamente richiesto dall’utente.

- Un normale servizio in background non è qualcosa di cui che l'utente è direttamente a conoscenza come in esecuzione, quindi il sistema ha più libertà nella gestione del suo processo. Potrebbe accadere che esso venga arrestato (e quindi riavviato qualche tempo dopo) se il sistema avesse bisogno di RAM per operazioni con priorità maggiore esplicitamente chieste dall'utente.

I servizi associati vengono eseguiti poiché un'altra app (o il sistema) ha bisogno delle operazioni svolte da tale servizio. Il sistema sa che esiste una dipendenza tra questi processi, quindi se per esempio un processo A è associato ad un servizio di un processo B, sa che deve mantenere il processo B (e il suo servizio) in esecuzione per A; inoltre, se il processo A è qualcosa di direttamente richiesto dall'utente, anche il processo B assume maggiore priorità.

Grazie alla loro flessibilità, i servizi si sono rivelati un elemento davvero importante per l'esecuzione corretta e semplificata di alcuni "task": sfondi animati, listener di notifiche, screen saver, metodi di input, servizi di accessibilità e molte altre funzionalità di sistema di base sono tutte costruite come servizi implementati dalle applicazioni e il sistema si occupa di eseguirle quando necessario.

Nella pratica un servizio è implementato come sottoclasse di "android.app.Service".

Si noti inoltre che questo settore è stato rivoluzionato negli anni, in particolare con la nascita dei JobScheduler che, come indica il nome stesso, svolgono operazioni pianificate in background con un uso parsimonioso della batteria.

Broadcast receiver

Un broadcast receiver è un componente che consente al sistema di informare le applicazioni del succedersi di eventi. Poiché i broadcast receiver sono componenti a sé stanti delle applicazioni, il sistema può informare anche le applicazioni che non sono attualmente in esecuzione. Molte trasmissioni provengono dal sistema, ad esempio una trasmissione che annuncia lo spegnimento dello schermo, che la batteria è scarica o che è stata catturata un'immagine.
Questi componenti come si può immaginare sono particolarmente utili per la gestione istantanea di determinate circostanze speciali. I broadcast receiver non utilizzano interfaccia grafica sebbene

possano inoltrare notifiche alla barra di stato per avvisare l'utente del succedersi di un determinato evento.

Anche le app stesse possono avviare delle trasmissioni, ad esempio per far sapere ad altre app che alcuni dati sono stati scaricati sul dispositivo e quindi sono disponibili per l'uso. Più comunemente, tuttavia, un broadcast receiver è solo un punto di passaggio per altri componenti ed è destinato a svolgere una quantità minima di lavoro, delegando a Service o JobScheduler eventuali operazioni da attivare.

Content provider

Un content provider si occupa di gestire un set condiviso di dati dell'app che è possibile archiviare nel file system, in un database SQLite, sul Web o in qualsiasi altro percorso di archiviazione persistente a cui l'app può accedere. Tramite il provider di contenuti, altre app possono eseguire query (interrogazioni alla base di dati) o modificare i dati se il provider di contenuti lo consente. Ad esempio, il sistema Android fornisce un content provider che gestisce le informazioni dei contatti nella rubrica; pertanto, qualsiasi app con le autorizzazioni appropriate può richiedere al content provider di leggere e scrivere informazioni di un determinato contatto.

È possibile pensare ad un content provider come un'astrazione su un database, tuttavia per il sistema un content provider è un punto di ingresso in un'app per la pubblicazione di dati rappresentati da oggetti con un particolare nome, identificati da uno schema URI. Pertanto un'app può decidere come mappare i dati che contiene su uno spazio di nomi URI, distribuendo tali URI ad altre entità che a loro volta possono usarli per accedere ai dati. Esistono alcuni elementi particolari che consentono al sistema di gestire un'app:

- La conoscenza di un URI non richiede che l'app in cui è stato definito rimanga in esecuzione, quindi gli URI possono persistere dopo la chiusura delle applicazioni in cui sono definiti. Il sistema deve solo assicurarsi che l'app sia in esecuzione quando deve recuperare i dati relativi all'URI in questione.
- Questi URI forniscono anche un importante modello di sicurezza a grana fine, poiché è possibile concedere l'autorizzazione di leggere determinati dati solo in modo temporaneo e in relazione al singolo "oggetto".

I content provider sono utili anche per leggere e scrivere dati privati per un’app e non condivisi.

Nella pratica un content provider è implementato come sottoclasse di “android.app.ContentProvider” e deve implementare un set standard di API per consentire ad altre app di eseguire transazioni.

Poiché il sistema esegue ciascuna app in un processo separato con autorizzazioni per i file che limitano l'accesso ad altre app, un’app non può attivare direttamente un componente di un'altra app. Tuttavia, il sistema Android fornisce la possibilità di richiedere l’esecuzione di un componente di un’altra app, recapitando un messaggio al sistema che specifica l'intenzione di avviare un particolare componente. Il sistema attiva quindi il componente per l’applicazione che l’ha richiesto.[5]

Grazie a questo metodo qualsiasi app può avviare, indirettamente previa autorizzazione, un componente di un'altra app. Ad esempio, se si desidera dare la possibilità che l'utente scatti una foto con la fotocamera del dispositivo, la nuova app può utilizzare l’app di sistema che effettua tale operazione invece di sviluppare un'attività per catturare una foto. Non è necessario incorporare o addirittura collegare il codice dall'app della fotocamera, basta semplicemente avviare l'attività nell'app della fotocamera che cattura una foto. Al termine, la foto viene restituita alla nuova app in modo da poterla utilizzare. In questo modo per l'utente sembra che la videocamera sia effettivamente parte della nuova applicazione.
Quando il sistema avvia un componente, avvia il processo per quell'app se non è già in esecuzione e crea un'istanza delle classi necessarie per il componente. Ad esempio, se un’applicazione avvia l'attività nell'app della fotocamera che cattura una foto, tale attività viene eseguita nel processo che appartiene all'app della fotocamera, non nel processo dell'app che lo richiede. Pertanto, a differenza delle applicazioni sulla maggior parte degli altri sistemi, le app Android non hanno un unico punto di ingresso (in pratica non esiste una funzione main()).

[5] https://developer.android.com/guide/components/fundamentals

1.1.3.2 Attivazione dei componenti

Come detto sopra un componente può attivarne un altro mediante apposite invocazioni di sistema.
Tre dei quattro tipi di componenti - activity, service e broadcast receiver - sono attivati da un messaggio asincrono chiamato Intent. Gli Intent associano i singoli componenti tra loro in fase di esecuzione. Si possono considerare come i messaggeri che richiedono un'azione da altri componenti, indipendentemente dal fatto che il componente appartenga all'applicazione corrente o ad un'altra.
A livello di codice un Intent viene creato con un oggetto Intent, che definisce un messaggio per attivare un componente specifico (Intent esplicito) o un tipo specifico di componente (Intent implicito).
Per attività e servizi, un Intent definisce l'azione da eseguire e può specificare l'URI dei dati su cui agire, ad esempio per trasmettere una richiesta ad un'attività di mostrare un'immagine o aprire una pagina web o inviare dei dati.
In alcuni casi, è possibile avviare un'attività per ricevere un risultato, nel qual caso l'attività restituisce anche il risultato in un Intent.
Ad esempio, è possibile emettere un Intent che consente di scegliere un contatto personale e di riceverlo. L'Intent di ritorno include un URI che punta al contatto scelto.

Per i broadcast receiver, l'Intent definisce semplicemente l'annuncio da trasmettere. Ad esempio, una trasmissione per indicare che la batteria del dispositivo è scarica include solo una stringa di azione nota che indica che la batteria è scarica.

A differenza di activity, service e broadcast receiver i content provider non sono attivati da Intent. Piuttosto vengono attivati quando sono scelti come target da una richiesta di un ContentResolver (risolutore dei contenuti). Il ContentResolver gestisce tutte le transazioni dirette con il content provider in modo tale che il componente che sta eseguendo transazioni con esso debba esclusivamente chiamare metodi sull'oggetto ContentResolver. Ciò crea un livello di astrazione tra il content provider e il componente che richiede informazioni, senza la diretta conoscenza reciproca.

1.1.3.3 Il file Manifest

Prima che il sistema Android possa avviare un componente dell'applicazione, il sistema deve sapere che il componente esiste leggendo il file Manifest dell'app, “AndroidManifest.xml”. L'app deve dichiarare tutti i suoi componenti in questo file, che deve trovarsi nella radice della directory del progetto dell'app.

Il Manifest fa diverse cose oltre a dichiarare i componenti dell'app:

- Identifica le autorizzazioni utente richieste dall'app, come ad esempio l'accesso a Internet o l'accesso in lettura ai contatti dell'utente.
- Dichiara il livello minimo di API richiesto dall'app, in base alle API utilizzate dall’app stessa.
- Dichiara le funzionalità hardware e software utilizzate o richieste dall'app, come una fotocamera, servizi bluetooth o uno schermo multitouch.
- Dichiara le librerie API a cui l'app deve essere collegata (diverse dalle API del framework Android), come ad esempio può essere la libreria di Google Maps.

Dichiarazione dei componenti

Il compito principale del file Manifest è informare il sistema sui componenti dell'app. Ad esempio, un file Manifest può dichiarare una activity come segue:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

Nell'elemento `<application>`, l'attributo `android: icon` punta alla risorsa che rappresenta l'icona utilizzata per identificare l'app.
Nell'elemento `<activity>`, l'attributo `android: name` specifica il nome completo della specifica classe Activity e l'attributo `android: label` specifica una stringa da usare come etichetta visibile dall'utente per l'attività.

E' necessario dichiarare tutti i componenti dell'app usando i seguenti elementi:
•Elementi `<activity>` per le Activity.
•Elementi`<service>` per i Service.
•Elementi`<receiver>` per i Broadcast Receiver.
• Elementi `<provider>` per i Content provider.

Le Activity, i service e i content provider che si includono nel codice ma che non si dichiarano nel file Manifest non sono visibili al sistema e, di conseguenza, non possono mai essere eseguiti.
Tuttavia, i Broadcast Receiver possono essere dichiarati nel file Manifest o creati dinamicamente nel codice come oggetti BroadcastReceiver e registrati nel sistema chiamando "registerReceiver()".

Dichiarazione dei requisiti dell'applicazione

Esistono diversi dispositivi basati su Android e non tutti offrono le stesse funzionalità e hanno le stesse capacità. Per impedire l'installazione dell'app corrente su dispositivi privi delle funzionalità necessarie a quest'ultima, è importante definire chiaramente un profilo per i tipi di dispositivi supportati dall'app dichiarando i requisiti del dispositivo nel file Manifest. La maggior parte di queste dichiarazioni sono solo a scopo informativo e il sistema non le legge, ma i servizi esterni come Google Play le leggono per fornire filtri agli utenti quando cercano app da installare sul proprio dispositivo.

Ad esempio, se l'app richiede una videocamera e utilizza API introdotte in Android 2.1 (API Level 7), è necessario dichiararle come requisiti nel file Manifest come mostrato nell'esempio seguente:

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
    ...
</manifest>
```

Con le dichiarazioni mostrate nell'esempio, i dispositivi che non dispongono di una fotocamera o hanno una versione di Android precedente alla 2.1 non possono installare la relativa applicazione da Google Play.

1.1.3.4 Risorse dell'applicazione

Un'applicazione Android è composta da più di semplice codice: richiede risorse separate dal codice sorgente, come immagini, file audio e qualsiasi altro elemento relativo alla presentazione visiva dell'applicazione. Ad esempio, è possibile definire animazioni, menù, stili, colori e il layout delle interfacce utente delle activity con i file XML.
Per ogni risorsa che si utilizza nel progetto Android dell'applicazione, gli strumenti di compilazione dell'SDK (Software Development Kit) definiscono un ID intero univoco, che si può utilizzare per fare riferimento alla relativa risorsa dal codice dell'applicazione o da altre risorse definite in XML. [6]
Uno degli aspetti più importanti della fornitura di risorse separate dal codice sorgente è la possibilità di fornire risorse alternative per diverse configurazioni del dispositivo. Ad esempio, definendo le stringhe dell'interfaccia utente in XML, è possibile tradurre le stringhe in altre lingue e salvarle in file separati. Di conseguenza Android applica le stringhe di lingua appropriate all'interfaccia utente in base a un qualificatore di lingua che si aggiunge al nome della directory delle risorse (come res/valori-fr/ per i valori di stringa francese) e alle impostazioni della lingua del dispositivo. Android supporta molti qualificatori diversi per le risorse alternative dichiarate. Il qualificatore è una

[6] https://developer.android.com/guide/topics/manifest/manifest-intro

stringa breve che viene inclusa nel nome delle directory delle risorse per definire la configurazione del dispositivo per cui tali risorse devono essere utilizzate.

Un altro esempio di rilevante importanza è il caso in cui è necessario creare layout diversi per le activity a seconda dell'orientamento e delle dimensioni dello schermo del dispositivo. Quando lo schermo del dispositivo è in orientamento verticale, è possibile che un layout con i pulsanti sia verticale, ma quando lo schermo è in orientamento orizzontale, i pulsanti potrebbero essere allineati orizzontalmente. Per modificare il layout in base all'orientamento, è possibile definire due diversi layout e applicare il qualificatore appropriato al nome della directory di ciascun layout. Quindi, il sistema, durante l'esecuzione, applica automaticamente il layout appropriato in base all'orientamento attuale del dispositivo.

1.2 Localizzazione nei dispositivi mobili

Al giorno d'oggi la conoscenza della propria posizione corrente è diventata una necessità per poter usufruire di molti servizi legati allo spostamento (vedi Google Maps, Waze etc) e al ritrovamento di determinate informazioni in base al luogo in cui ci si trova. Comprendere questo aspetto risulta quindi molto importante per poter progettare software funzionale e che rispetti le specifiche e le necessità del tempo contemporaneo.

Sentendo la parola "geolocalizzazione", si associa subito l'idea al GPS e all'impiego dei satelliti. In realtà è bene notare che la determinazione della posizione tramite satellite non è l'unica tecnica possibile applicabile ai dispositivi mobili, dato che anche l'utilizzo delle reti WiFi o delle celle delle reti degli operatori mobili permettono la localizzazione; è però al momento, l'utilizzo del GPS, la tecnica che fornisce il maggiore livello di accuratezza per tutte le applicazioni orientate alla navigazione.

1.2.1 Global Positioning System

GPS è la sigla di Global Positioning System**,** un sistema di posizionamento e navigazione satellitare militare statunitense[7], ed è ormai entrato nel linguaggio comune come il sistema che consente di calcolare la posizione di un punto sulla superficie terrestre grazie al calcolo della latitudine, longitudine e altitudine.

Attraverso una rete dedicata di satelliti artificiali in orbita, fornisce ad un terminale mobile o ricevitore GPS informazioni sulle sue coordinate geografiche e sul suo orario in ogni condizione meteorologica, ovunque sulla Terra o nelle sue immediate vicinanze dove vi sia un contatto, tra il ricevitore e il satellite, privo di ostacoli con almeno quattro satelliti del sistema. La localizzazione avviene tramite la trasmissione di un segnale radio da parte di ciascun satellite e l'elaborazione dei segnali ricevuti da parte del ricevitore.

[7] https://www.wikipedia.org/wiki/Geolocalizzazione

È gestito dal governo degli Stati Uniti d'America ed è liberamente accessibile da chiunque sia dotato di un ricevitore GPS. Il suo grado attuale di accuratezza è dell'ordine di pochi metri, in dipendenza dalle condizioni meteorologiche, dalla disponibilità e dalla posizione dei satelliti rispetto al ricevitore, dalla qualità e dal tipo di ricevitore, dagli effetti di radiopropagazione del segnale radio in ionosfera e troposfera (es. rifrazione) e dagli effetti della relatività.

Ad oggi oltre a tale sistema di posizionamento ci sono altri operatori che hanno lanciato satelliti per la georeferenziazione in ambito sia civile che militare. In particolare sono già operativi il sistema GLONASS (russo), GALILEO (europeo), BEIDOU (cinese) e QZSS (giapponese).

L'hardware dei moderni dispositivi mobili è in grado di sfruttare le informazioni combinate da più sistemi satellitari, in modo da aumentare le possibilità di rilevamento con la massima accuratezza.

Installando e usando l'applicazione "GPS Test" della Chartcross Ltd disponibile su Play Store, è possibile rilevare la posizione corrente e visualizzare in modo grafico, semplice e intuitivo le informazioni sui satelliti e le loro posizioni nel cielo. Un esempio dei dati visualizzati è riportato nella figura sottostante, dove si possono vedere le posizioni dei satelliti visibili sul cielo marcati in base al tipo di operatore e i dati relativi alla posizione corrente. Inoltre è possibile decidere quale sistema di posizionamento utilizzare e quali impostazioni utilizzare per il rilevamento della posizione.

Figura 6 – Schermate relative all'applicazione "GPS Test"

Il principio su cui si basa il calcolo della posizione di un punto nello spazio è molto simile a quello della trilaterazione in cui fissato un sistema di riferimento, se conosciamo la distanza di un punto *p*, di posizione sconosciuta, da altri noti, possiamo, per intersezione delle superfici sferiche con centro nei punti noti e raggio le distanze calcolate con *p*, determinarne le coordinate.

1.2.1.1 Trilaterazione rispetto a 4 punti noti

Se si conosce la distanza tra noi e almeno 4 satelliti è possibile calcolare la nostra precisa posizione nello spazio, poiché:

- con 1 non sapremmo dove siamo posizionati in tutta la superficie sferica che rappresenta la distanza tra noi e il satellite;
- con 2 avremmo 2 superfici sferiche che si intersecano e potremmo avere come soluzione una circonferenza di punti nello spazio;
- con 3 la superficie sferica intersecherebbe la circonferenza di prima in al più 2 punti;
- per risolvere l'ultima ambiguità, serve un quarto satellite.

In realtà già considerando 3 satelliti avremmo che i due punti, che sono la soluzione al problema, sono posizionati lungo un asse ortogonale al piano formato dai tre satelliti: uno rivolto verso la Terra e l'altro rivolto verso lo spazio esterno. Chiaramente il punto esterno è da scartare perché si troverebbe per assurdo sopra le orbite dei satelliti dove non può arrivare nemmeno il segnale trasmesso, dato che le antenne sono rivolte verso la Terra.

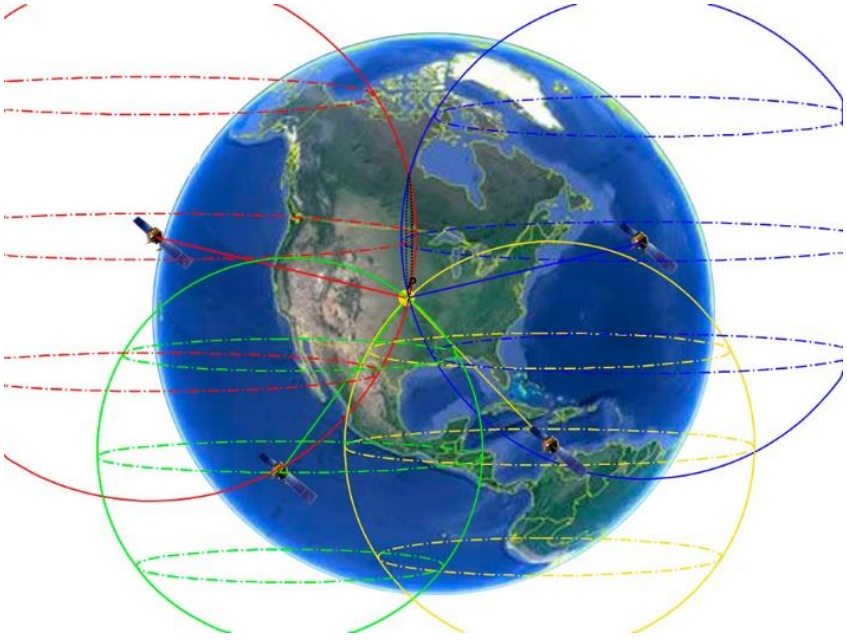

Figura 7 - Esempio grafico del calcolo

1.2.1.2 Il punto sulla superficie terrestre

Il passo successivo è quello di portare il punto nel sistema di riferimento terrestre a noi noto, ossia latitudine, longitudine e altitudine**.** Questo è un altro passaggio geometrico che ha lo scopo di proiettare il punto su una superficie ellissoidale in modo da ricavarne le coordinate sul nuovo sistema di riferimento.

La superficie ellissoidale in questione serve ad approssimare al meglio la superficie della Terra considerando i punti sulla superficie come posti allo zero altimetrico.

Negli scorsi due secoli, ogni area del globo faceva riferimento a un ellissoide differente, ma ai giorni nostri esiste ormai da qualche decennio un ellissoide di riferimento adottato in tutto il mondo per approssimare la forma della superficie terrestre. Questo standard è il WGS84 (World Geodetic System 1984) introdotto nel 1984 e questa sigla spesso ritorna quando si parla di GPS o anche semplicemente di rappresentazione di punti su mappe.

1.2.1.3 Misurare le distanze

La chiave per determinare con sufficiente precisione la distanza tra un punto e un satellite sta nella misura del tempo intercorso tra la trasmissione di un segnale emesso dal satellite e la ricezione al punto di misura.

Per fare questo, ogni satellite invia un segnale rappresentato da una semplice sequenza binaria che appare del tutto casuale ma in realtà è una sequenza ben nota che si ripete con una lunga periodicità (pseudo-random).

Questa sequenza è assolutamente caratteristica di ciascun satellite per cui, dopo un determinato periodo di osservazione, che serve al rilevatore per determinare a che punto della sequenza il satellite è arrivato con la trasmissione, si è in grado di stabilire l'orario a cui il satellite lo avrebbe emesso; confrontandolo con l'orologio del rilevatore si è in grado di determinare per differenza il tempo Δt intercorso tra trasmissione e ricezione.

Il controllo dell'emissione della sequenza dal satellite è affidato a degli orologi atomici a bordo dei satelliti che riescono a misurare il tempo con una precisione di gran lunga inferiore al nanosecondo, mentre i rilevatori GPS dei dispositivi mobili in genere utilizzano frequenze di riferimento tra le decine e centinaia di MHz.

1.2.1.4 Migliorare la precisione

Sembra però che uno dei grandi vincoli sia quello di avere l'orologio del dispositivo sempre accuratamente preciso. In realtà anche questo limite è risolto con qualche trucco; infatti, se per esempio l'orologio del rilevatore fosse leggermente "indietro" le distanze calcolate apparirebbero "accorciate". In questi casi può risultare però un errore nel calcolo delle figure geometriche utilizzate per determinare la posizione e quindi il rilevatore GPS procede nel ricalcolare il tempo quanto basta per risolvere il problema. L'errore introdotto nel variare il tempo in questo modo, avendo un buon numero di satelliti, non supera la precisione limite dello strumento.

Questo passaggio può impiegare molto tempo se non c'è certezza assoluta sull'orologio del rilevatore. Per questo motivo molti rilevatori GPS installati sui dispositivi mobili utilizzano le risorse di rete per ottenere almeno un orario più preciso e una vaga posizione che consenta di determinare quali satelliti dovremmo avere "visibili" sulla volta celeste. Le informazioni sono reperibili dalla rete degli operatori mobili considerando che di ogni antenna è perfettamente nota la posizione. Questo espediente, noto come Assisted-GPS **(**AGPS**)**, ha lo scopo di aiutare il rilevatore GPS ad essere più rapidamente in grado di rilevare una posizione ma comunque non è in grado di aumentarne la precisione.

Infine, altro vincolo, sembra essere quello di conoscere esattamente la posizione dei satelliti al momento della trasmissione del segnale per proiettare correttamente il punto determinato sull'ellissoide di riferimento.

Questo problema è risolto dal satellite stesso inviando, su un differente canale, ogni 30 secondi i propri dati telemetrici.

Calcolare la posizione, in pratica

1.2.1.5 GNSS

GNSS (Global Navigation Satellite System) implementa in pratica l'algoritmo descritto in precedenza senza aggiungere tanto altro.

Inoltre, una volta tarato il processo, la posizione è calcolabile di continuo istantaneamente anche se si è in movimento e quindi è un metodo valido anche per calcolare la velocità istantanea e relativa direzione.

Non consente tuttavia una particolare precisione perché, anche se lo strumento ne fosse in grado, ci sono dei fattori non eliminabili da questo metodo che portano a considerare la precisione offerta non migliore dei 5 metri di approssimazione.

Questi fattori sono essenzialmente dovuti al passaggio del segnale nella ionosfera e nella troposfera.

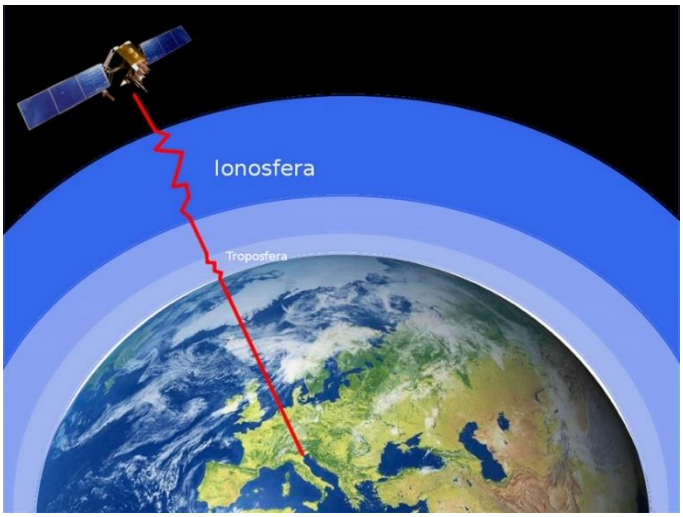

Figura 8 – Ritardo del segnale del satellite che passa attraverso i vari strati dell'atmosfera

In queste parti dell'atmosfera terreste il segnale può subire deviazioni casuali che introducono un ritardo sensibile. Il fattore di ritardo non è costante o sempre uguale per ogni satellite, per cui il calcolo finale può non essere perfettamente corretto.

Limitandosi a queste precisioni, i rilevatori GPS che usano questo metodo sono anche relativamente economici non avendo bisogno di componenti estremamente precisi e quindi costosi.

1.2.1.6 DGNSS

DGNSS (Differential GNSS) si basa sulla considerazione che due rilevatori GPS a terra abbastanza vicini (nell'ordine di pochi km) avranno visibili sulla volta celeste gli stessi satelliti e quindi l'influenza derivata dal passaggio dei segnali per la ionosfera e la troposfera si fa sentire su entrambi allo stesso modo.[8]

Se di uno dei due rilevatori è esattamente nota a priori la posizione, la stessa differenza di lettura tra la posizione rilevata e quella nota si deve applicare anche all'altro rilevatore.

In questo modo il secondo rilevatore può, applicando tale correzione, determinare il punto con maggiore precisione, questa volta dipendente quasi esclusivamente dalla bontà del rilevatore GPS che, come abbiamo visto, può aggirarsi attorno al metro.

Per realizzare questo sistema c'è bisogno che il rilevatore GPS possa ricevere i dati di correzione **(**RTCM**)** da una stazione fissa tramite un canale dedicato per la trasmissione dati.

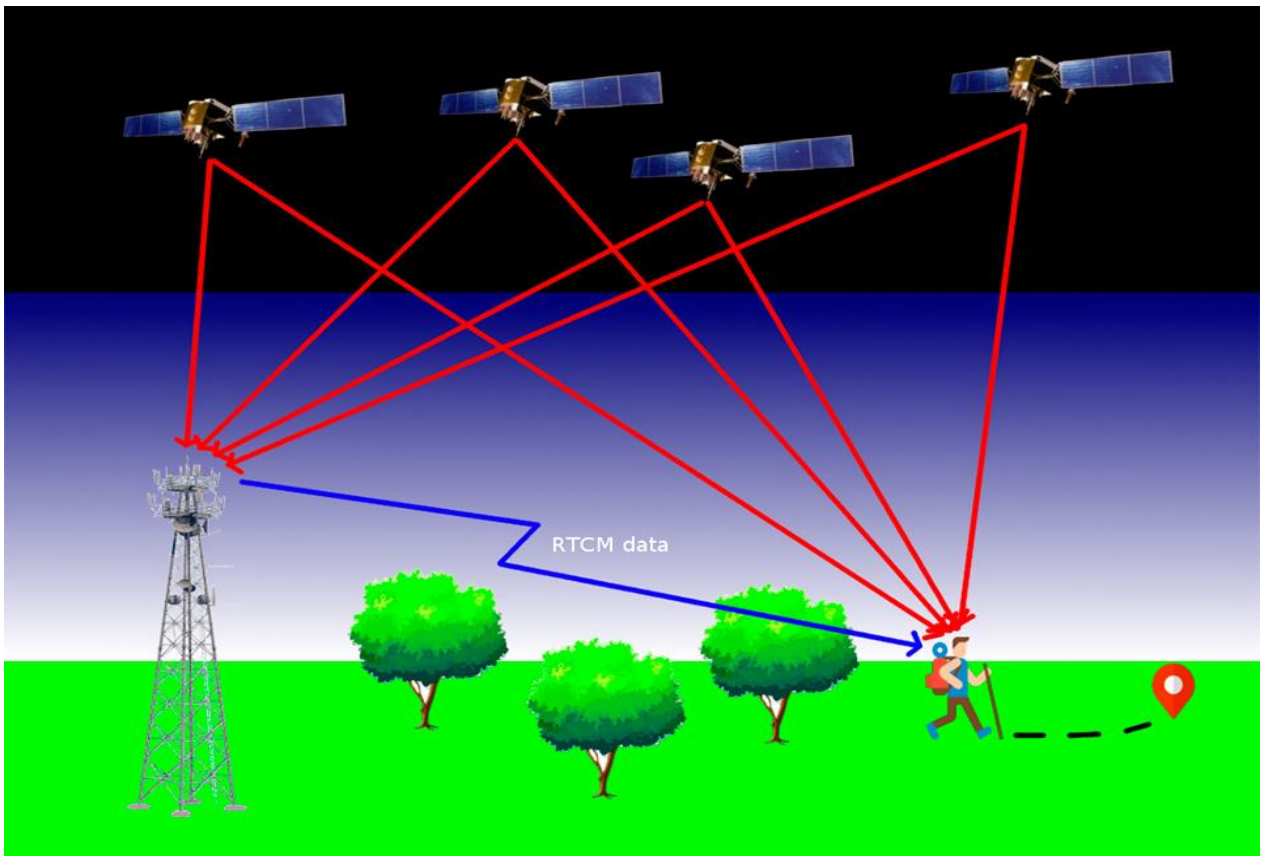

Figura 9 - Sistema DGNSS

[8] http://www.mokabyte.it/2019/03/geolocmobile-1/

Esistono delle implementazioni accessibili pubblicamente tramite Internet che forniscono i dati delle correzioni GNSS.

Nel caso di dispositivi mobili, l'accesso a Internet è una delle funzioni primarie, per cui si potrebbe beneficiare di questa possibilità con un costo bassissimo. Tuttavia, come si evince dal fatto che i servizi di GNSS sono offerti a livello locale e in maniera disomogenea, la possibilità di utilizzare queste risorse per delle applicazioni "GPS-based" di un generico dispositivo mobile su scala mondiale è alquanto difficoltoso.

Se questa funzione venisse invece assolta dagli operatori mobili, la situazione potrebbe essere ben diversa. Infatti per un dispositivo mobile accedere ad una rete di trasmissione è il primo dei suoi scopi e per farlo ha bisogno di agganciare una cella di un operatore. La cella è caratterizzata da una antenna fissa che, per ragioni di qualità del segnale, non può distare mai più di qualche chilometro e la cui posizione potrebbe essere determinata a priori con precisione topografica (pochi cm).

Se l'antenna fosse dotata anche di un rilevatore GPS e trasmettesse a tutti i cellulari agganciati i dati delle correzioni da applicare, si potrebbe avere un comodo sistema di DGNSS a costo praticamente zero su scala globale.

I vantaggi del DGNSS sono gli stessi del GNSS per quanto riguarda — una volta completato il periodo di decine di secondi necessario per l'avvio del processo — il rilevamento istantaneo della posizione e della velocità di moto, ma con in più una precisione di gran lunga migliore.

1.2.2 WiFi RTT

WiFi RTT (Round Trip Time) è stato introdotto con la specifica IEEE 802.11mc nel 2016 ed è ormai implementata da quasi tutti i "vendor" di access point WiFi. Il concetto che sta alla base di WiFi RTT è molto simile a quello della trilaterazione già descritto per il calcolo della posizione da satellite. Il calcolo della distanza dal rilevatore all'access point avviene però mediante la misura del round trip time.

Questa funzione è molto simile a quella ben nota di ping, l'unica differenza è che il rilevatore non opera a livello di IP ma al **l**ivello di rete che è quindi più basso. Infatti il rilevatore ha bisogno di avere i dati da tutti gli access point disponibili nel suo raggio di azione per avere il maggior numero di tempi misurati allo scopo di determinare un punto con precisione.

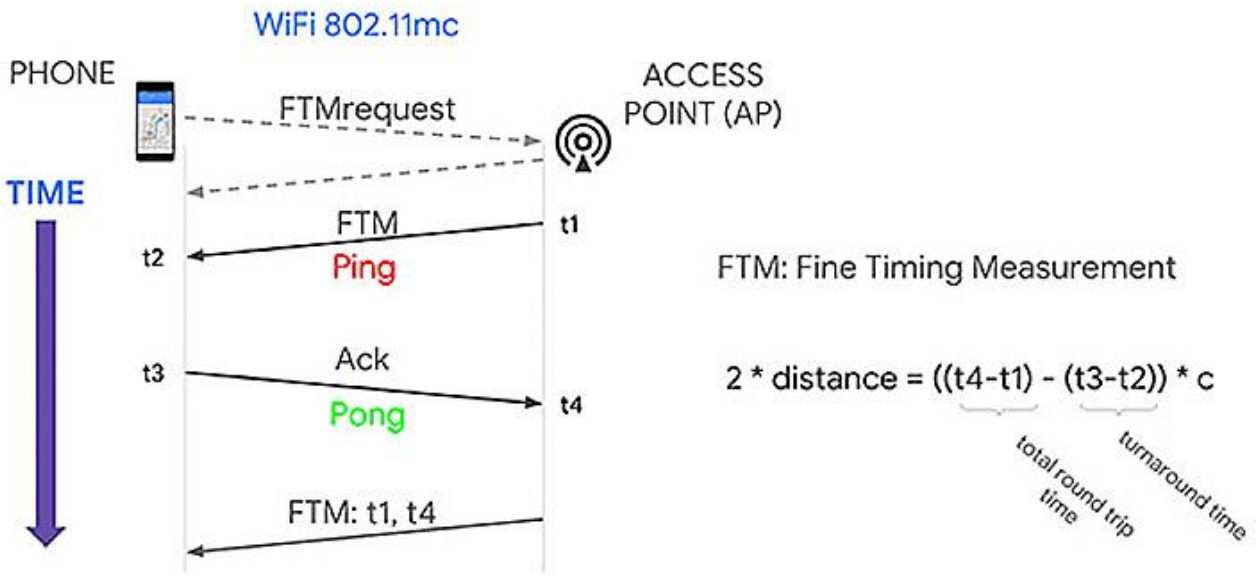

Figura 10 - WIFI RTT

Come nel caso della trilaterazione nel GNSS, ci vogliono almeno 4 access point per determinare un punto, anche se ne potrebbero bastare 3 nel caso si considerino applicazioni solo su superfici limitate e in piano. La misurazione del tempo è molto accurata perché dipende tutto dall'orologio del rilevatore e, come nel caso del calcolo del GNSS, si può arrivare a precisioni dell'ordine del metro.

1.2.3 Localizzazione cellulare

La localizzazione cellulare è meno precisa di quella GPS, ma è possibile utilizzarla anche nei dispositivi mobili che non sono dotati di ricevitore GPS o in ambienti interni in cui non è possibile utilizzare il GPS.

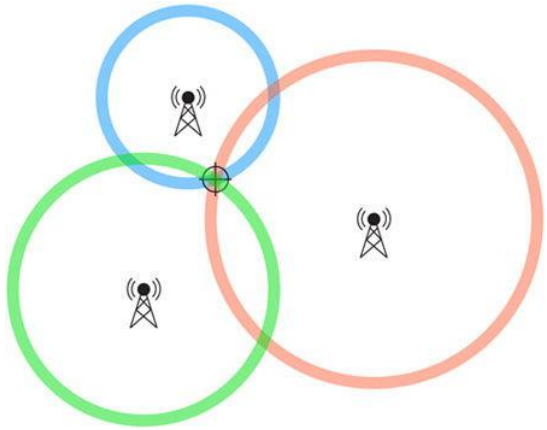

Figura 11 - Esempio di possibile localizzazione tramite celle

Questo metodo di calcolo della posizione, utilizza le antenne radio, in particolare sono necessarie 3 antenne, a cui ogni smartphone è connesso per la ricezione GSM(Global System for MobileCommunications) o UMTS(Universal Mobile Telecommunications System) per i servizi di telefonia mobile.

Le antenne sono posizionate sulle cosiddette celle, le quali hanno un raggio di trasmissione che dipende da molti fattori tra i quali la densità di terminali connessi contemporaneamente e la potenza trasmissiva.

Si analizza la potenza del segnale radio di ogni cella telefonica in relazione alla rispettiva stazione radio base (che ha coordinate geografiche note) collegata con il dispositivo mobile o terminale e ne viene determinata la distanza da questa in base alla conoscenza dell'attenuazione dell'ambiente di radiopropagazione.

1.2.4 Implementazioni e applicazioni nei dispositivi mobili: considerazioni

L'introduzione del WiFi RTT è relativamente recente. È stata integrata negli smartphone Android a partire dalla versione P(API livello 28) uscita a metà del 2018. Sicuramente è una tecnologia di interesse per applicazioni indoor, dove il segnale GPS comunque non è disponibile, e in edifici dove si sia prevista l'installazione di un numero sufficiente di access point. È quindi una tecnologia che si orienta ad applicazioni domotiche o allestimenti multimediali in edifici di grandi dimensioni**,** come ad esempio a musei.

La precisione offerta la renderebbe adatta anche per applicazioni di navigazione veicolare, ma ci sarebbe bisogno di implementare una rete di access point conforme la specifica IEEE 802.11mc anche all'esterno.

Come già detto, sempre nel caso indoor, è possibile utilizzare la localizzazione tramite la rete cellulare, la quale in base al contesto può risultare più o meno precisa dell'utilizzo del wifi;

La localizzazione più precisa resta quella effettuata tramite il GPS; al giorno d'oggi la soluzione migliore è abilitare tutte e tre le tipologie di rilevamento per ottenere la posizione più accurata in base al luogo e alla situazione in cui ci si trova.

Capitolo 2 : Kotlin e il LocationManager

In questo capitolo verrà presentato il linguaggio di programmazione Kotlin, utilizzato per lo sviluppo della applicazione, e spiegata nello specifico la classe “LocationManager”, la quale permette la localizzazione nei dispositivi mobili.

2.1 Utilizzo del linguaggio Kotlin per lo sviluppo Android

Kotlin è un linguaggio di programmazione general purpose, multi-paradigma, open source sviluppato dall'azienda di software JetBrains.

Figura 12 - Logo Kotlin

Kotlin si basa sulla JVM (Java Virtual Machine) ed è ispirato ad altri linguaggi di programmazione tra i quali Scala e lo stesso Java, mentre ulteriori spunti sintattici sono stati presi da linguaggi classici, come il Pascal e moderni come Go o F#. [9]Lo sviluppo è iniziato nel 2011 mentre una prima versione pubblica del compilatore, in versione 0.1, è stata pubblicata a gennaio 2012. Kotlin è un linguaggio particolarmente orientato verso la programmazione a oggetti permettendo peraltro un pieno uso dell'approccio funzionale. Kotlin è strutturato per interoperare con la piattaforma Java Runtime Environment come target principale, il che garantisce il funzionamento delle applicazioni in ogni ambiente che accetti la JVM, ivi compreso Android, ma il compilatore è in grado anche di emettere codice JavaScript.

[9] https://it.wikipedia.org/wiki/Kotlin_(linguaggio_di_programmazione)

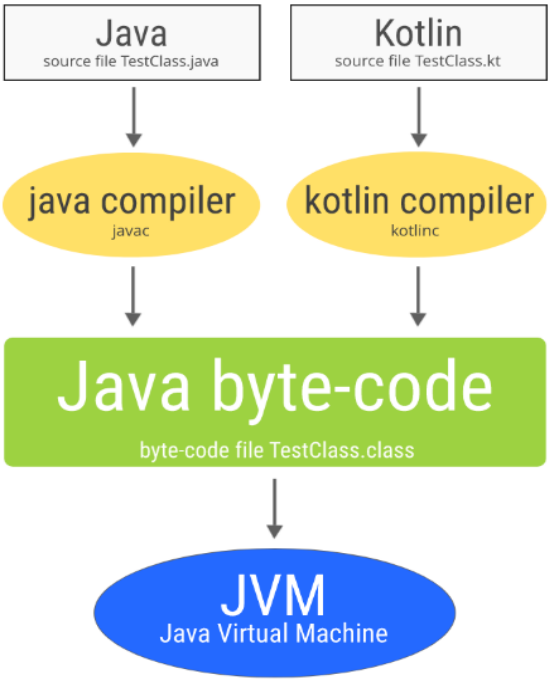

Figura 13 - Java e Kotlin processo di compilazione

Per quanto riguarda la compilazione, partendo da un file Kotlin .kt , questo viene compilato in file JVM *.class, che è un formato bytecode che può essere successivamente interpretato da una JVM o compilato nel codice macchina dalla JVM durante l'esecuzione del programma (JIT- Just In Time) o addirittura compilato in anticipo (AOT – ahead of time) fino al codice macchina[10]. Di conseguenza il compilatore Kotlin non ha bisogno di sapere come verrà utilizzato esattamente il bytecode. Bisogna tener conto che il .class compilato a partire dal codice Kotlin non può essere decompilato per generare un file .java .

È inoltre presente la possibilità di compilare il linguaggio Kotlin direttamente in linguaggio macchina tramite il compilatore Kotlin per l'ambiente di riferimento.

Kotlin si adatta perfettamente allo sviluppo di applicazioni Android, offrendo tutti i vantaggi di un linguaggio moderno per la piattaforma Android senza introdurre nuove restrizioni. Il linguaggio Kotlin è stato adottato e integrato nell'ambiente di sviluppo (IDE) Android Studio a partire dalla versione 3.0.

Ci sono diversi motivi per cui negli ultimi anni è diventato uno dei linguaggi di programmazione più usati e di rilievo per lo sviluppo di applicazioni Android:

- Compatibilità: Kotlin è completamente compatibile con la JDK 6, garantendo l'esecuzione delle applicazioni Kotlin su dispositivi Android meno recenti senza problemi. Gli strumenti di Kotlin sono completamente supportati in Android Studio e compatibili con il sistema di build Android.
- Performance: un'applicazione Kotlin è veloce come una equivalente Java, grazie alla struttura del bytecode molto simile. In alcuni contesti permette un'esecuzione anche più veloce dello stesso codice scritto in Java.
- Interoperabilità: Kotlin è interoperabile al 100% con Java, consentendo di utilizzare tutte le esistenti librerie Android in un'applicazione Kotlin.

[10] https://medium.com/@banmarkovic/process-of-compiling-android-app-with-java-kotlin-code-27edcfcce616

- Tempo di compilazione: Kotlin supporta un'efficiente compilazione incrementale, build incrementali sono generalmente veloci o più veloci rispetto a Java.
- Learning Curve: per uno sviluppatore Java, iniziare con Kotlin è molto semplice; l'automatizzato processo di conversione da Java a Kotlin incluso nel plugin Kotlin è molto utile soprattutto all'inizio dell'utilizzo di questo linguaggio.

Kotlin è ora il linguaggio consigliato da Google per la scrittura di applicazioni ed è stato adottato con successo da molte grandi aziende come Pinterest, BaseCamp e Keepsafe's App Lock i cui programmatori sostengono che l'utilizzo di questo linguaggio porta ad una maggiore qualità del software e velocità di sviluppo.[11]

2.1.1 Confronto tra Java e Kotlin

Ciò che si nota subito mettendo a confronto codice Java e codice Kotlin scritto per fare la stessa cosa è la quantità di esso: il codice Kotlin risulta molto più sintetico di quello Java grazie a diverse semplificazioni e proprietà di cui code questo linguaggio.

A proposito della sinteticità, a livello di compilatore si può considerare la Kotlin Android Extensions che è una estensione del compilatore che permette di utilizzare le varie View(oggetti utilizzati per la visualizzazione a schermo che fanno parte delle activity) senza dover richiamare in modo esplicito il metodo findViewById(), di cui invece il codice Java necessita. Questo è possibile poiché il compilatore genera in automatico delle proprietà legate all'oggetto View in questione esonerando il programmatore dalla scrittura di queste.

[11] https://kotlinlang.org/docs/

2.1.1.1 Tipi e variabili

In Kotlin tutto è oggetto: classi, funzioni, variabili etc. Alcuni tipi possono avere una rappresentazione interna speciale - ad esempio numeri, caratteri e valori booleani possono essere rappresentati come tipi primitivi in fase di esecuzione - ma per l'utente sembrano classi normali, mentre ciò non accade in Java in cui i tipi primitivi non sono considerati come oggetti.

Mentre in Java è necessario definire il tipo di una variabile, in Kotlin questo non è necessario. È sufficiente definirla con il suffisso `val` (per indicare che non potrà essere modificata dopo essere stata inizializzata, cioè una costante) o `var` (per indicare che il suo valore può cambiare), inoltre non è necessario utilizzare l'operatore `new` in fase di istanziazione uniformando le chiamate a costruttori con quelle a funzioni.

Poiché non è necessario esplicitare il tipo di una variabile questo viene dedotto in modo implicito dal compilatore, tecnica chiamata type-inference. Ciò permette una scrittura più "pulita" e veloce del codice, in particolare nei casi dove c'è un'inutile ripetizione espressiva con nomi molto lunghi.

2.1.1.2 Null Safety

Uno dei problemi più grandi che il programmatore Java si trova davanti nella scrittura di codice è l'insorgere di "null pointer exceptions": in Java è possibile assegnare il valore `null` ad una variabile e a runtime se si prova ad accedere a tale variabile, senza avergli assegnato un valore diverso da `null`, viene generata un'eccezione. Questo implica che è presente un errore logico nel codice e molte volte ci si rende conto di tale errore solo a runtime e non a tempo di compilazione. Una soluzione è quella di inserire opportuni controlli sul valore della variabile da utilizzare per evitare l'insorgere di questa eccezione, aumentando la verbosità in modo sostanziale quando sono presenti in numero elevato variabili con possibile valore nullo.

In Kotlin di dafault non si può assegnare il valore `null` ad una variabile, se si cerca di assegnare o restituire come risultato `null` la compilazione non va a buon fine. [12]Se veramente si vuole che una

[12] https://italiancoders.it/kotlin-cose-cosa-serve-sapere-ai-programmatori-java/

variabile abbia valore `null`, si deve dichiarare in modo esplicito tramite l'utilizzo dell'operatore `?`, con una sintassi di questo tipo:

```
var a :Int? = null
```

Anche in questo caso è necessario effettuare dei controlli in presenza di variabili possibilmente nulle, ma la verbosità è molto ridotta.

2.1.1.3 Funzioni

In Kotlin, a differenza di Java, come detto in precedenza, le funzioni sono trattare esattamente come sono trattati gli oggetti. Non è necessario creare una classe per contenere una funzione ma questa può essere dichiarata semplicemente a livello "globale". Inoltre le funzioni in Kotlin possono anche essere dichiarate locali, come funzioni membro e funzioni di estensione.

Le funzioni in Kotlin sono definite di "prima classe" poiché sono trattate esattamente come sono tratti gli oggetti. In aggiunta a quanto è possibile fare in Java, cioè definirle e chiamarle, è possibile assegnarle a variabili, passarle come parametro ad altre funzioni, essere restituite da altre funzioni (funzioni di ordine superiore) e anche allocarne istanze a runtime.

È possibile quindi operare con le funzioni in qualsiasi modo possibile per altri valori non funzionali.

Per quanto riguarda le funzioni di estensione, Kotlin offre la possibilità di estendere una classe con nuove funzionalità senza dover ereditare dalla classe o utilizzare modelli di design come Decorator come invece accade in Java. Tali funzioni sono disponibili per la chiamata come se fossero metodi della classe originale. È bene notare che si può agire allo stesso modo anche per le proprietà di una classe.

Per dichiarare una funzione di estensione, è necessario aggiungere il prefisso al suo nome con un tipo di ricevitore, ovvero il tipo che viene esteso. Quanto segue aggiunge una funzione di scambio a `MutableList <Int> :`

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

Kotlin a differenza di Java supporta le funzionali locali, ossia la possibilità di inserire una funzione all'interno di un'altra funzione e anche le funzioni "inline", le quali permettono di eliminare l'overhead generato dall'utilizzo delle funzioni di ordine superiore e dalle lambda expression evitando il salvataggio di esse in oggetti.

2.1.1.4 Controllo delle eccezioni

In Java è presente il controllo delle eccezioni, il quale obbliga i programmatori a dichiarare e catturare, tramite try/catch, le varie eccezioni che si possono generare durante l'esecuzione, rendendo il codice robusto e con un'ottima gestione degli errori. In Kotlin questo meccanismo non è presente, rendendo il programmatore libero di gestire oppure no l'eventuale insorgere di eccezioni durante l'esecuzione; questo porta naturalmente dei vantaggi e degli svantaggi.

2.1.1.5 Classi Data

Nel caso si renda necessario avere una classe che rappresenta un'entità, occupandosi solo di mantenere i relativi dati senza necessità di comportamenti, in Java è necessario definire il/i costruttore/i, definire le variabili per il salvataggio dei dati, scrivere i metodi getter e setter, hashCode(), equals() ,toString() e copy().

In Kotlin basta definire una classe con la keyword "data" e il compilatore si occuperà di tutto il lavoro ripetitivo per i vari campi definiti, come la generazione dei metodi getter e setter e quelli sopra

indicati. Una dichiarazione del tipo `var name: String` , se scritta nel corpo di una classe , non corrisponde semplicemente ad un attributo come in Java, ma corrisponde ad una property, cioè un attributo con getter e setter.

2.1.1.6 Smart Cast

In molti casi, a differenza di Java, non è necessario utilizzare operatori di cast espliciti in Kotlin, poiché il compilatore tiene traccia dei controlli "is" e dei cast espliciti per valori immutabili e effettua automaticamente cast (sicuri) quando necessario.

Un esempio di quanto detto è il seguente:

```
fun demo(x: Any) {
    if (x is String) {
        print(x.length) // x è automaticamente "castato" in una stringa
    }
}
```

Il compilatore è abbastanza smart per riconoscere un cast sicuro in seguito ad un controllo negativo (`!is`):

```
if (x !is String) return
print(x.length) // x è automaticamente "castato" in una stringa
```

Anche nel caso si utilizzino gli operatori && e || :

```
// x è automaticamente "castato" in una stringa alla destra di `||`
if (x !is String || x.length == 0) return

// x è automaticamente "castato" in una stringa alla destra di `||`
if (x is String && x.length > 0) {
    print(x.length) // x è automaticamente "castato" in una string
}
```

Si noti che questi cast "intelligenti" non funzionano quando il compilatore non può garantire che la variabile non cambi tra il controllo e il suo uso.

2.1.1.7 Operator Overloading

Kotlin, a differenza di Java, fornisce la possibilità effettuare l'overload degli operatori standard, in modo che lavorare con tipi definiti dal programmatore risulti più semplice. Si può effettuare l'overload di tutti gli operatori unari, binari e relazionali.

Gli operatori sono sovrascritti tramite funzioni membro o funzioni di estensione e tali funzioni sono precedute dal modificatore `operator`. Esistono funzioni standard per ogni tipo di operatore che possono essere sovrascritte in base all'utilizzo che se ne vuole fare.

La tabella sottostante indica le funzioni relative ad ogni operatore unario prefisso.

Operatore	Funzione standard
`+a`	`a.unaryPlus()`
`-a`	`a.unaryMinus()`
`!a`	`a.not()`

Ad esempio, ecco come è possibile fare l'overload dell'operatore unario prefisso "meno" in relazione ad un tipo definito:

```
data class Point(val x: Int, val y: Int)
operator fun Point.unaryMinus() = Point(-x, -y)
val point = Point(10, 20)

fun main() {
   println(-point)  // stampa "Point(x=-10, y=-20)"
}
```

2.1.1.8 Array

Un'altra differenza tra Java e Kotlin è la gestione degli Array: in Kotlin questi sono invarianti, ciò significa che per crearne uno non è necessario dichiararlo indicando il tipo di dato da contenere, come invece avviene in Java, ma, nel caso più semplice, è sufficiente utilizzare la funzione di libreria arrayOf() che crea un'array con i valori indicati al suo interno.

2.1.1.9 String Templates

Un'altra caratteristica di Kotlin che ne aumenta l'espressività sono gli string templates, non presenti in Java. [13]Essi sono delle espressioni che vengono valutate dal compilatore e poi il loro valore viene inserito nella stringa in cui sono definite evitando di utilizzare per esempio in modo ripetitivo l'operatore + come si fa con Java. Per definire un'espressione dentro una stringa di testo si deve inserire l'espressione all'interno delle parentesi graffe e precederla con il simbolo del dollaro ($). Un esempio di quanto detto è il seguente:

```
var num = 10

val testo = "${num * num} è il quadrato di $num"
```

2.1.1.10 Supporto alle Coroutines

La programmazione asincrona o non bloccante è la nuova realtà al giorno d'oggi: che si tratti di creare applicazioni lato server, desktop o mobili, è importante fornire un'esperienza non solo fluida dal punto di vista dell'utente, ma scalabile quando necessario; Kotlin utilizza le Coroutines per permettere tutto questo.

[13] https://www.geeksforgeeks.org/difference-between-java-and-kotlin-in-android-with-examples/

Le coroutines sono un modo semplice per implementare codice asincrono e non bloccante. Si basano su funzioni che hanno la capacità di sospendere la loro esecuzione ad un certo punto e riprendere in seguito.

Si alternano in modo cooperativo nell'esecuzione, consentendo così uno stile di programmazione non bloccante.

Le coroutines sono leggere e non richiedono molte risorse di sistema: un thread può eseguire molte coroutines, inoltre risultano più sicure e meno soggette a errori rispetto ad utilizzare in modo concorrente più thread per fornire lo stesso tipo di operazioni.

Questo supporto non è presente in Java dove invece l'utilizzo di più thread porta maggiore complessità nella loro gestione per il programmatore.

2.2 LocationManager

Il LocationManager rappresenta il componente centrale del pacchetto *android.location* per l'accesso ai servizi di localizzazione del sistema tramite l'utilizzo di determinate API. Questi servizi consentono alle applicazioni di ottenere aggiornamenti periodici della posizione geografica del dispositivo o di essere avvisati quando il dispositivo entra in prossimità di una determinata posizione geografica.[14]

La posizione si determina grazie all'utilizzo di due tipi di provider:

- LocationManager.GPS_PROVIDER
- LocationManager.NETWORK_PROVIDER

Il primo si basa sull'utilizzo del modulo GPS, che deve ovviamente essere presente nel dispositivo; mentre il secondo si basa sull'utilizzo del WiFi o dei dati mobili del dispositivo con modalità spiegate nel capitolo di localizzazione.

Una posizione geografica è rappresentata dalla data class Location.
Una posizione può essere costituita da latitudine, longitudine, timestamp e altre informazioni come direzione di spostamento, altitudine e velocità.[15]

È garantito che tutte le posizioni generate dal LocationManager abbiano latitudine, longitudine e timestamp validi (sia l'ora UTC che il tempo reale trascorso dall'avvio), mentre tutti gli altri parametri sono opzionali.

Un metodo importante di tale classe, oltre a basici metodi getter e setter relativi alle informazioni sopra indicate è quello che permette di calcolare l'accuratezza della posizione:

```
public float getAccuracy ()
```

[14] https://developer.android.com/reference/android/location/LocationManager
[15] https://developer.android.com/reference/android/location/Location

Tramite tale metodo si ottiene la precisione orizzontale stimata della posizione in questione, radiale, in metri. Minore è il numero di metri restituito maggiore è la precisione della posizione.

Si definisce la precisione orizzontale come il raggio del 68% di confidenza. In altre parole, se si disegna un cerchio centrato alla latitudine e longitudine di questa posizione e con un raggio uguale alla precisione, c'è una probabilità del 68% che la posizione reale sia all'interno del cerchio.

Questa stima dell'accuratezza riguarda solo l'accuratezza orizzontale e non indica l'accuratezza della direzione, velocità o altitudine se questi sono inclusi nella posizione.

Se la posizione non ha una precisione orizzontale, viene restituito 0,0. Tutte le posizioni generate dal LocationManager includono la precisione orizzontale.

2.2.1 Permessi per la localizzazione

Se un'app deve accedere alla posizione dell'utente è necessario richiedere in modo esplicito ad esso l'autorizzazione alla posizione del dispositivo aggiungendola alle autorizzazioni dell'app. Android offre due autorizzazioni di localizzazione: ACCESS_COARSE_LOCATION e ACCESS_FINE_LOCATION. L'autorizzazione scelta determina la precisione della posizione restituita dall'API utilizzata. È sufficiente richiedere solo una delle autorizzazioni di localizzazione Android, a seconda del livello di precisione di cui si ha bisogno:

- android.permission.ACCESS_COARSE_LOCATION – Consente all'API di utilizzare Wi-Fi o dati mobili (o entrambi) per determinare la posizione del dispositivo. L'API restituisce la posizione con un'accuratezza approssimativamente equivalente ad un isolato.

- android.permission.ACCESS_FINE_LOCATION – Consente all'API di determinare una posizione il più precisa possibile dai fornitori di posizione disponibili, incluso il sistema di posizionamento globale (GPS), nonché dati Wi-Fi e cellulari.

2.2.2 Metodi principali

La classe LocationManager contiene molti metodi e costanti utilizzati per la rilevazione della posizione in diversi modi e con l'utilizzo di diversi elementi. In seguito vengono mostrati i metodi più indicati ed utilizzati per la localizzazione e anche metodi accessori necessari per la corretta gestione della localizzazione.

I principali metodi, spiegati con precisione in seguito, sono:

- getLastKnownLocation: utilizzato per ottenere istantaneamente una posizione già calcolata.
- getCurrentLocation: permette di ottenere in modo asincrono una posizione.
- addProximityAlert: notifica l'ingresso o l'uscita da una determinata area.
- requestLocationUpdates: servizio di aggiornamento costante della posizione.
- getBestProvider : permette di ottenere il provider che soddisfa determinati criteri.

Ad esclusione del metodo addProximityAlert, gli altri necessitano dell'autorizzazione Manifest.permission.ACCESS_COARSE_LOCATION o Manifest.permission.ACCESS_FINE_LOCATION

2.2.2.1 getLastKnownLocation

```
public Location getLastKnownLocation (String provider)
```

Tramite questo metodo si ottiene l'ultima posizione nota dal provider specificato o null se non si conosce nessuna posizione. La posizione restituita potrebbe essere piuttosto vecchia in alcuni casi, quindi è necessario controllare sempre quanto questa risulta aggiornata.
Questo metodo non attiva mai i sensori per calcolare una nuova posizione e restituisce solo una posizione memorizzata nella cache del sistema.

2.2.2.2 getCurrentLocation

```
public void getCurrentLocation (String provider,
                CancellationSignal cancellationSignal,
                Executor executor,
                Consumer<Location> consumer)
```

Questo metodo restituisce in modo asincrono la posizione corrente. Ciò potrebbe attivare i sensori per calcolare una nuova posizione, utilizzando di conseguenza risorse.

Quando risulterà presente una risposta il Consumer(metodo di callback) dato verrà richiamato una e una sola volta, con una posizione valida o con una posizione nulla se il provider non è stato in grado di generare una posizione valida.
Un cliente può fornire un segnale di cancellazione opzionale, CancellationSignal, per fermare la ricerca della posizione.

Questo metodo può restituire posizioni da un passato molto recente (nell'ordine di alcuni secondi), ma non restituisce mai posizioni più vecchie (ad esempio, diversi minuti o più).

Offre la garanzia che se restituisce una posizione, questa rappresenta la migliore stima della posizione del dispositivo nel momento presente.
E’ possibile notare che se tale metodo viene richiamato in background non sempre riesce a determinare una posizione valida, almeno non quanto riesce a farlo nel caso l’operazione venga effettuata in foreground(primo piano), poiché le applicazioni in background possono essere limitate in una certa misura negli accessi alla posizione del dispositivo.

2.2.2.3 addProximityAlert

```
open fun addProximityAlert(
    latitude: Double,
    longitude: Double,
    radius: Float,
    expiration: Long,
    intent: PendingIntent
): Unit
```

Imposta un avviso di prossimità per l'area calcolata a partire dalla posizione (latitudine, longitudine) e il raggio inseriti.
Quando il dispositivo rileva che è entrato o è uscito dall'area circostante la posizione, il PendingIntent specificato verrà utilizzato per creare un Intent da lanciare.
L'Intent lanciato conterrà un booleano impostato all'atto dell'evento: se il valore è true, il dispositivo sta entrando nella regione calcolata, se è false, sta uscendo.

A causa della natura approssimativa della stima della posizione, se il dispositivo passa brevemente attraverso l'area indicata, è possibile che non venga attivato alcun Intent; allo stesso modo, un Intent potrebbe essere lanciato se il dispositivo passa molto vicino a una determinata area ma in realtà non vi entra.
Dopo il numero di millisecondi fornito dal parametro di scadenza, il gestore della posizione eliminerà questo avviso di prossimità e non lo monitorerà più. Inserendo un valore di -1 si indica che non è presente tempo di scadenza e controlla sempre l'ingresso o l'uscita dall'area indicata. Internamente per quanto riguarda i provider utilizzati, questo metodo utilizza sia NETWORK_PROVIDER che GPS_PROVIDER.

Prima della versione 17 dell'API, questo metodo poteva essere utilizzato con android.Manifest.permission.ACCESS_FINE_LOCATION o android.Manifest.permission.ACCESS_COARSE_LOCATION. Dalla versione 17 dell'API in poi, questo metodo richiede l'autorizzazione android.Manifest.permission.ACCESS_FINE_LOCATION.

2.2.2.4 requestLocationUpdates

```
public void requestLocationUpdates (String provider,
                long minTimeMs,
                float minDistanceM,
                LocationListener listener)
```

Questo è il metodo principale e più utilizzato per la localizzazione in modo continuo del dispositivo e presenta diverse "signature" oltre a quella indicata sopra. Tramite questo metodo si indica la volontà di voler ottenere aggiornamenti della posizione dal provider fornito come argomento. Il funzionamento di questo tipo di localizzazione si basa su un modello ad "eventi", in cui il cambiamento della posizione, in relazione a determinati parametri, notificato al LocationListener(elemento che si occupa di stare in ascolto per eventuali aggiornamenti) scatena l'invocazione di determinati metodi di callback.

Le chiamate di callback del LocationListener avranno luogo sul Looper, elemento spiegato meglio in seguito, o sull'Executor(oggetto che si occupa di eseguire le attività indicategli). Se viene fornito un Looper nullo, verrà utilizzato il Looper del thread chiamante. È possibile registrare una sola richiesta per ogni listener, quindi qualsiasi richiesta successiva con lo stesso listener sovrascriverà tutti gli argomenti associati.
Potrebbe essere necessario diverso tempo per ricevere il primo aggiornamento della posizione. Se è richiesta un'ubicazione immediata, le applicazioni possono utilizzare il metodo getLastKnownLocation (java.lang.String).

Si può controllare l'intervallo di aggiornamento della posizione utilizzando il parametro del tempo minimo da indicare come argomento del metodo. Il tempo trascorso tra gli aggiornamenti della posizione non sarà mai inferiore a questo parametro, anche se potrebbe essere maggiore a seconda della disponibilità della posizione e di altri fattori. La scelta di un valore ragionevole per il parametro del tempo minimo è importante per preservare la durata della batteria, poiché ogni aggiornamento della posizione richiede l'alimentazione di una varietà di sensori.

Anche il parametro della distanza minima tra l'ultima posizione calcolata e quella successiva può essere utilizzato per controllare la frequenza degli aggiornamenti della posizione. Se è maggiore di 0,

il provider di posizione invierà un aggiornamento all'applicazione solo quando la posizione è cambiata di almeno minDistance metri e quando è trascorso il tempo minimo. Tuttavia è più difficile per i provider di localizzazione risparmiare energia utilizzando il parametro della distanza minima, quindi il parametro del tempo minimo risulta essere lo strumento principale per preservare la durata della batteria.

Se l'applicazione desidera osservare passivamente gli aggiornamenti della posizione attivati da altre applicazioni, ma senza consumare ulteriore energia, si utilizza il PASSIVE_PROVIDER. Questa modalità non attiva nuovi provider né modifica i provider di localizzazione attivi, quindi in questo caso non è necessario prestare eccessiva attenzione ai parametri di tempo minimo e distanza minima.

Nel caso in cui il provider selezionato sia disabilitato, gli aggiornamenti di posizione cesseranno e verrà inviato un aggiornamento di disponibilità del provider. Non appena il provider viene nuovamente abilitato, verrà inviato un altro aggiornamento sulla disponibilità del provider e riprenderanno immediatamente gli aggiornamenti della posizione.

Prima di Jellybean(Android versione 4.3), il parametro minTime era solo un "suggerimento" e alcune implementazioni del provider di posizione lo ignoravano. Da Jellybean e successivi, tuttavia, è obbligatorio per i dispositivi compatibili con Android osservare entrambi i parametri minTime e minDistance.

2.2.2.5 getBestProvider

```
public String getBestProvider (Criteria criteria,
                boolean enabledOnly)
```

Restituisce il nome del provider che soddisfa meglio i criteri specificati. Verranno considerati solo i provider a cui è consentito l'accesso da parte del chiamante. Se più provider soddisfano i criteri, viene restituito quello con la migliore accuratezza.

Criteria è la classe che indica i criteri dell'applicazione per la selezione di un provider di posizione. I fornitori possono essere ordinati in base a precisione, consumo energetico, capacità di segnalare altitudine, velocità, direzione e costo monetario.

Oltre ai metodi per impostare i criteri sopra indicati sono presenti anche costanti per rappresentare diversi requisiti:

- `ACCURACY_COARSE` : requisito di accuratezza approssimativa
- `ACCURACY_FINE` : requisito di accuratezza più precisa
- `ACCURACY_HIGH` : requisito di elevata precisione - può essere utilizzato per la precisione orizzontale, altitudine, velocità o direzione
- `ACCURACY_LOW` : requisito di precisione della posizione bassa - può essere utilizzato per la precisione orizzontale, altitudine, velocità o direzione
- `ACCURACY_MEDIUM` : requisito di precisione medio, attualmente utilizzato solo per la precisione orizzontale
- `NO_REQUIREMENT` : non si pongono requisiti su una particolare funzionalità
- `POWER_HIGH` : requisito di elevato consumo energetico
- `POWER_LOW` : requisito di basso consumo energetico
- `POWER_MEDIUM` : requisito di medio consumo energetico

Per quanto riguarda il metodo getBestProvider, se nessun fornitore soddisfa i criteri indicati, i criteri vengono “allentati” nel seguente ordine:

- Requisiti energetici
- precisione
- direzione
- velocità
- altitudine

Notare che il requisito sul costo monetario non viene rimosso in questo processo

2.2.3 LocationListener

Questa interfaccia è utilizzata, come detto in precedenza, per ricevere notifiche dal LocationManager quando la posizione cambia. Comprende una serie di metodi che devono risultare tutti implementati per il corretto funzionamento della localizzazione. Questi metodi vengono chiamati se un LocationListener è stato registrato utilizzando il metodo requestLocationUpdates (String, long, float, LocationListener).

I metodi da implementare sono i seguenti:

-onLocationChanged

```
abstract fun onLocationChanged(location: Location): Unit
```

Chiamato quando la posizione cambia. Al suo interno generalmente si inseriscono logiche di funzionamento dell'applicazione e si effettuano cambiamenti a livello di visualizzazione.

-onProviderDisabled

```
open fun onProviderDisabled(provider: String): Unit
```

Chiamato quando il provider viene disabilitato dall'utente. Se requestLocationUpdates viene chiamato su un provider già disabilitato, questo metodo viene chiamato immediatamente.

-onProviderEnabled

```
open fun onProviderEnabled(provider: String): Unit
```

Chiamato quando il provider viene abilitato dall'utente.

-onStatusChanged

```
open fun onStatusChanged(
    provider: String!,
    status: Int,
    extras: Bundle!
): Unit
```

Questo metodo non verrà mai chiamato su Android Q e versioni successive (API livello 29) poiché vengono utilizzati i metodi descritti sopra. Si basa sul concetto di stato del provider rappresentato da diverse costanti.

2.2.4 Looper

Il Looper è un altro elemento cardine per il funzionamento del metodo “requestLocationUpdates” descritto sopra.
Questa classe è utilizzata per gestire una coda di messaggi, MessageQueue, per un determinato thread. Per impostazione predefinita ai thread non è associata una coda di messaggi.
Per permettere ciò si chiama il metodo “prepare()” nel thread che deve eseguire il ciclo, quindi eseguire il ciclo, tramite il metodo “loop()”, il quale elabora i messaggi nella coda fintanto che non si indica in modo esplicito la terminazione di tale operazione.
La maggior parte delle interazioni con un loop di messaggi avviene tramite un handler, il gestore che si occupa di effettuare la corretta operazione in base al messaggio ricevuto.

Questo è un tipico esempio di implementazione di un thread Looper, utilizzando la separazione di "prepare()" e "loop()" per creare un Handler per la gestione dei messaggi:

```
class LooperThread extends Thread {
       public Handler mHandler;

       public void run() {
           Looper.prepare();

           mHandler = new Handler() {
               public void handleMessage(Message msg) {
                   // process incoming messages here
               }
           };

           Looper.loop();
       }
   }
```

2.2.5 Metodi accessori

Vengono ora elencati una serie di metodi accessori utilizzati per effettuare controlli e modificare l'esecuzione dei metodi spiegati sopra.

2.2.5.1 removeUpdates

```
open fun removeUpdates(listener: LocationListener): Unit
```

Tramite questo metodo si rimuovono gli aggiornamenti di posizione per il LocationListener specificato, quindi a seguito di questa chiamata, il Listener non riceverà più gli aggiornamenti sulla posizione.

Richiede android.Manifest.permission.ACCESS_COARSE_LOCATION o android.Manifest.permission.ACCESS_FINE_LOCATION

2.2.5.2 removeProximityAlert

```
open fun removeProximityAlert(intent: PendingIntent): Unit
```

Rimuove l'avviso di prossimità legato all'oggetto PendingIntent specificato.

Come per addProximityAlert() Prima della versione 17 dell'API, questo metodo poteva essere utilizzato con android.Manifest.permission.ACCESS_FINE_LOCATION o android.Manifest.permission.ACCESS_COARSE_LOCATION. Dalla versione 17 dell'API in poi, questo metodo richiede l'autorizzazione android.Manifest.permission.ACCESS_FINE_LOCATION.

2.2.5.3 getAllProviders

```
open fun getAllProviders(): MutableList<String!>
```

Restituisce un elenco dei nomi di tutti i provider di posizione noti. Vengono restituiti tutti i provider, inclusi quelli a cui non è consentito l'accesso dall'activity chiamante o che sono attualmente disabilitati.

2.2.5.4 isLocationEnabled

```
open fun isLocationEnabled(): Boolean
```

Restituisce “true” se la localizzazione è abilitata, “false” altrimenti.

2.2.5.5 isProviderEnabled

```
open fun isProviderEnabled(provider: String): Boolean
```

Restituisce “true” se il provider indicato è attivo, “false” altrimenti.

2.2.6 Esempio di Uso

Per concludere è presente un esempio di utilizzo a grandi linee del LocationManager e LocationListener per ottenere la posizione in modo continuo del dispositivo.

```
//impostazione del Location Manager
private var locationManager : LocationManager? = null
locationManager = getSystemService(LOCATION_SERVICE) as LocationManager?

//definizione del listener
private val locationListener: LocationListener = object :
LocationListener {
    override fun onLocationChanged(location: Location) {
        //logica da eseguire quando la posizione cambia
    }
    override fun onStatusChanged(provider: String, status: Int, extras:
Bundle) {}
    override fun onProviderEnabled(provider: String) {}
    override fun onProviderDisabled(provider: String) {}

//abilitazione dell'aggiornamento della posizione con la registrazione
del listener definito sopra
locationManager?.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
0L, 0f, locationListener)
```

Capitolo 3 : Progettazione e OpenStreetMap

In questo capitolo verrà descritto e analizzato il progetto dell'applicazione includendo descrizione, requisiti, macro-moduli e descrizione grafica. Inoltre verrà presentato il progetto OpenStreetMap.

3.1 Progettazione dell'applicazione

3.1.1 Descrizione del contesto

Come già detto nel capitolo relativo alla localizzazione dei dispositivi mobili, questo ambito, della localizzazione, risulta al giorno d'oggi di grande importanza, poiché sempre più servizi si basano sulla localizzazione e quindi sulla conoscenza del luogo in cui l'utente, che usufruisce dei servizi, si trova.

Nel progetto e nello sviluppo di applicazioni che possano rendere disponibili tali servizi è necessario e di primaria importanza tener conto dei dispositivi sui quali tale software andrà ad essere installato ed eseguito.

Ogni giorno che passa porta alla luce nuovi smartphone con nuove tecnologie che aumentano la diversità di questo mondo, così in forte sviluppo da ormai dieci anni a questa parte, ognuno con diverse caratteristiche tecniche. Questo fa sì che lo sviluppo di nuove applicazioni debba tener conto della necessità di progettare e realizzare software compatibile con un'elevata molteplicità di dispositivi.

Concentrandosi sui dispositivi, un altro aspetto molto importante è l'utilizzo delle risorse di cui il dispositivo dispone. Per questo risulta necessario focalizzare l'attenzione anche su quelli che sono i consumi che la nuova applicazione avrà sullo smartphone in questione, principalmente per quanto riguarda i consumi in termini di CPU, memoria RAM e batteria del dispositivo. Di questi tre elementi quello che risulta ancora oggi più problematico e il vero "tallone di Achille" degli smartphone è la

batteria, poiché non garantisce una lunga autonomia di funzionamento attivo (si intende utilizzo a schermo acceso).

Si è deciso così di progettare un'applicazione con funzionalità di localizzazione per giri a piedi e successivamente studiarne i consumi per poter capire in modo approfondito quanto la localizzazione influisca sulle risorse sopra citate e possibilmente indicare una eventuale metodologia di implementazione che possa ridurre i consumi.

3.1.2 Descrizione del progetto

Il progetto si occupa dello sviluppo di un'applicazione per la piattaforma Android che permetta la localizzazione del dispositivo e la registrazione di informazioni relative a tragitti effettuati a piedi. In particolare l'utente potrà visualizzare la mappa del mondo in tempo reale con possibilità di spostarsi su di essa e di "zoom in" e "zoom out"; richiedere il ritrovamento della propria posizione con la mappa che si sposta su quest'ultima; registrare e visualizzare il tragitto a piedi che si sta svolgendo; infine visualizzare lo storico delle informazioni relative ai tragitti salvati.

3.1.3 Requisiti funzionali e non funzionali

Vengono ora indicati i requisiti del sistema, i quali rappresentano la descrizione dei servizi forniti e dei vincoli operativi. Più precisamente vengono indicati i requisiti funzionali, che descrivono quello che il sistema dovrebbe fare con specifiche complete e coerenti, e i requisiti non funzionali, i quali non riguardano direttamente le specifiche funzionali ma altre caratteristiche del progetto.

3.1.3.1 Requisiti funzionali

- Calcolo posizione corrente

Possibilità di visualizzare sulla mappa tramite un marker la propria posizione corrente, premendo un determinato bottone.

- Visualizzare una mappa aggiornata del mondo

Visualizzare sulla schermata principale la mappa aggiornata del mondo.

- Bussola

Presenza di una bussola sulla schermata principale per le indicazioni dei punti cardinali.

- Mini-mappa

Presenza di una mini-mappa nell'angolo in basso a destra dello schermo, la quale permette di avere una prospettiva meno "zoomata" rispetto alla mappa principale.

- Barra con indicazione di lunghezza

Presenza di una barra nella parte bassa che indica la scala lineare, ossia quanto la lunghezza visibile della barra rappresenta nella realtà la lunghezza in metri/chilomentri

- Spostamento nella mappa

Possibilità di muoversi all'interno della mappa nella schermata principale.

- “Zoom in” e “Zoom out” sulla mappa

Possibilità di avvicinarsi (“Zoom in”) e di allontanarsi (“Zoom out”) da un determinato punto della mappa con l’utilizzo di pulsanti oppure effettuando determinati movimenti con le dita sullo schermo.

- Registrare un nuovo percorso

Tramite un determinato pulsante è possibile iniziare a registrare un nuovo percorso e, eventualmente, inserire un titolo.

- Salvare i dati relativi al percorso appena terminato

Al termine del percorso registrato è possibile decidere se salvare le informazioni relative al tragitto fatto. Tali informazioni includono: titolo (se inserito all’inizio della registrazione), data, ora di inizio, ora di fine, chilometri percorsi e dislivello positivo affrontato.

- Aggiornamento costante della mappa durante il percorso

Come comportamento di default la mappa, durante la registrazione del percorso, rimane costantemente centrata sulla posizione corrente.

- Fermare/ripristinare l’aggiornamento della mappa

Durante la registrazione del tragitto è possibile bloccare il comportamento per il quale la mappa rimane centrata sulla posizione corrente, semplicemente muovendosi sulla mappa. È possibile, nel caso si sia precedentemente fermato, ripristinare il comportamento del costante aggiornamento della mappa cliccando sul bottone per il calcolo della posizione corrente.

- Visualizzazione di uno storico relativo ai percorsi salvati

Possibilità di visualizzare, in una pagina dedicata, la lista dei percorsi salvati con le relative informazioni.

- Visualizzazione di una pagina con informazioni sulla applicazione

Presenza di una pagina dedicata con informazioni sull'applicazione.

- Eliminazione dei percorsi salvati

Possibilità, nella pagina di storico, di eliminare i percorsi con le relative informazioni.

3.1.3.2 Requisiti non funzionali

- Semplicità di navigazione tra le varie schermate

La fruizione dell'applicazione deve risultare intuitiva e semplice per renderla utilizzabile dalla totalità degli utilizzatori di smartphone.

- Reattività nella ricerca della posizione

La ricerca e il calcolo della propria posizione devono risultare più reattivi possibili, tenendo conto dei tempi intrinsechi e ineliminabili per effettuare tali operazioni. Questo requisito deve essere rispettato poiché è necessario che l'applicazione agli occhi dell'utente non risulti "bloccata" e quindi mal funzionante.

- Memorizzazione su file in memoria interna

Il sistema memorizzerà le informazioni relative ai percorsi effettuati in un file di testo in memoria interna, garantendo così un'elevata velocità di scrittura e lettura da tale file e l'impossibilità di agire su tale file da percorsi esterni.

- Utilizzo dell’applicazione previa autorizzazione dei permessi

Sarà possibile utilizzare l’applicazione solo dopo che l’utente avrà esplicitamente consentito l’autorizzazione ai permessi necessari per il corretto funzionamento dell’applicazione. Tale argomento è di fondamentale importanza per le politiche Android poiché sempre più informazioni personali sono presenti nei nostri smartphone ed è necessario garantire la giusta protezione a tali informazioni.

3.1.4 Sistemi esterni

Per lo sviluppo dell’applicazione è risultato necessario utilizzare un sistema esterno da cui reperire i dati per la generazione della mappa da mostrare all’utente: OpenStreetMap.

Successivamente, in tale capitolo, verrà spiegato in cosa consiste il progetto OpenStreetMap, per poi indicare con dettaglio il suo utilizzo nel capitolo successivo.

3.1.5 Diagramma dei casi d’suo

Per formalizzare i requisiti funzionali e comprendere meglio il funzionamento del sistema, sotto è presente il diagramma dei casi d’suo.

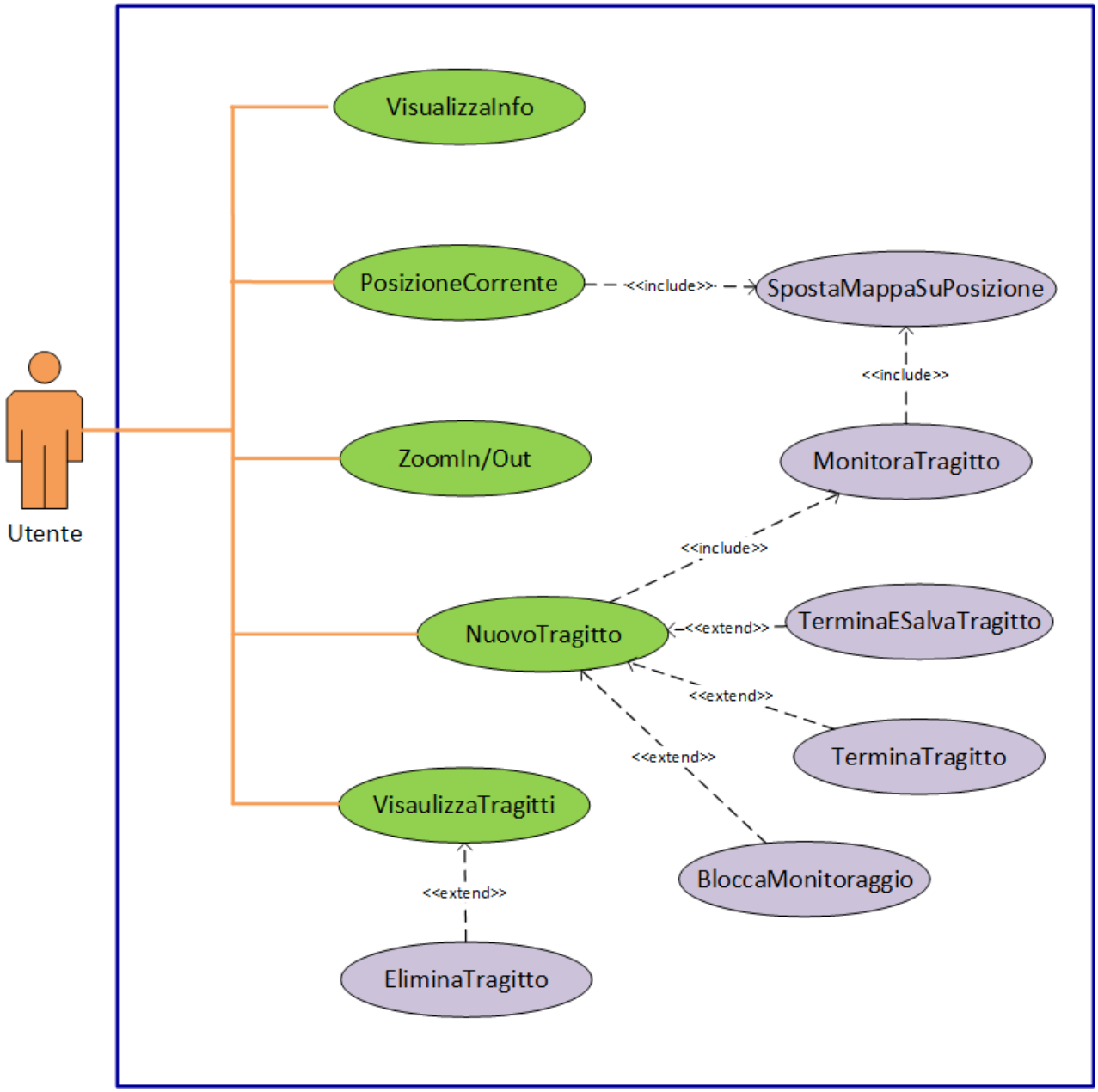

Figura 14 - Diagramma dei casi d’uso

3.1.6 Struttura dell’applicazione

Per quanto riguarda la struttura della applicazione si è deciso di adottare il pattern architetturale Layer (strati) e in particolare di dividere l’architettura in tre livelli:

- Presentazione
- Logica di business
- Persistenza

Il primo si occupa dell’interfaccia grafica, anche detta UI (User Interface) mostrata all’utente; il secondo è adibito alla logica di funzionamento dell’applicazione, quindi all’implementazione di tutte le funzionalità descritte nei requisiti; il terzo si occupa invece di gestire la persistenza dei dati, quindi in questo caso, il salvataggio e recupero delle informazioni relative ai percorsi effettuati.

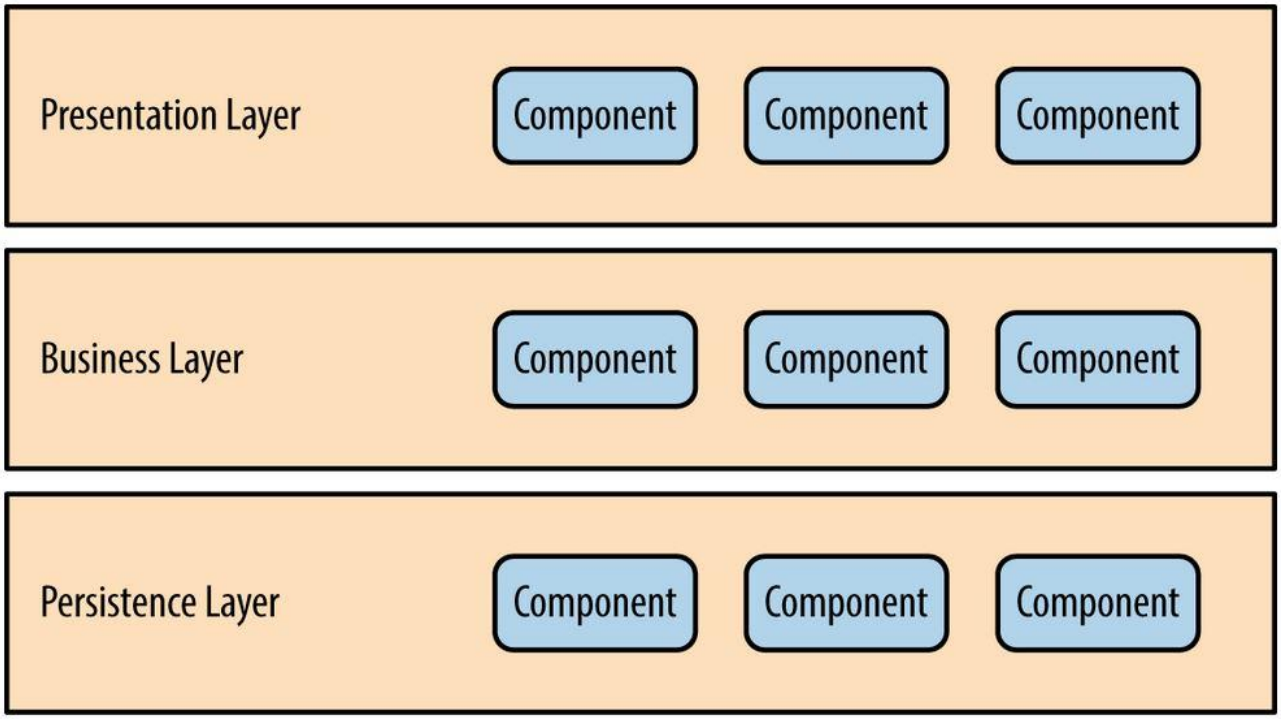

Figura 15 - Architettura a layer

Uno dei vantaggi dell’adozione del pattern di architettura a Layer è la separazione delle responsabilità tra i componenti. I componenti all'interno di un livello specifico si occupano solo della logica che

appartiene a quel livello. Ad esempio, i componenti nel livello di presentazione si occupano solo della logica di presentazione, mentre i componenti che risiedono nel livello di logica di business si occupano solo della logica di business e non di altro. [16]

Questo tipo di separazione dei componenti semplifica la creazione di ruoli e modelli di responsabilità efficaci nell'architettura e semplifica anche lo sviluppo, il test, la gestione e la manutenzione delle applicazioni. Per permettere questa potente separazione si utilizzano interfacce per ogni componente.

Tale modello di sviluppo rispetta anche il "Dependency inversion Principle", principio di design in cui i moduli di alto livello non dipendono dai moduli di basso livello in modo diretto ma tramite astrazioni, in questo caso interfacce.

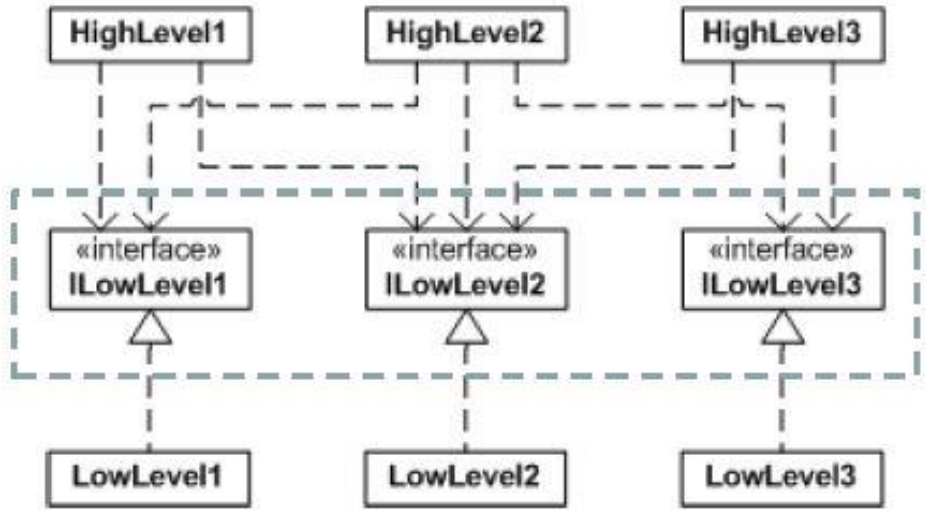

Figura 16 - Applicazione del "Dependency Inversion Principle"

I moduli di basso livello contengono la maggior parte del codice e della logica implementativa e quindi sono i più soggetti a cambiamenti.

Se i moduli di alto livello dipendessero dai dettagli dei moduli di basso livello (risultano quindi accoppiati in modo troppo stretto), i cambiamenti si propagherebbero e le conseguenze sarebbero:

- Rigidità: bisogna intervenire su un numero elevato di moduli

[16] https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

- Fragilità: si introducono errori in altre parti del sistema
- Immobilità: i moduli di alto livello non si possono riutilizzare perché non si riescono a separare da quelli di basso livello

Questo principio funziona poiché le astrazioni contengono pochissimo codice (in teoria nulla) e quindi sono poco soggette a cambiamenti; i moduli non astratti sono soggetti a cambiamenti ma questi cambiamenti sono sicuri perché nessuno dipende da tali moduli.
In questo modo i dettagli del sistema sono stati isolati, separati da un muro di astrazioni stabili, e questo impedisce ai cambiamenti di propagarsi ("design for change"). Nel contempo i singoli moduli sono maggiormente riusabili perché sono disaccoppiati fra di loro ("design for reuse").

3.1.7 Diagramma architetturale

Nella figura sottostante è riportato il diagramma architetturale dell'applicazione con i vari componenti.

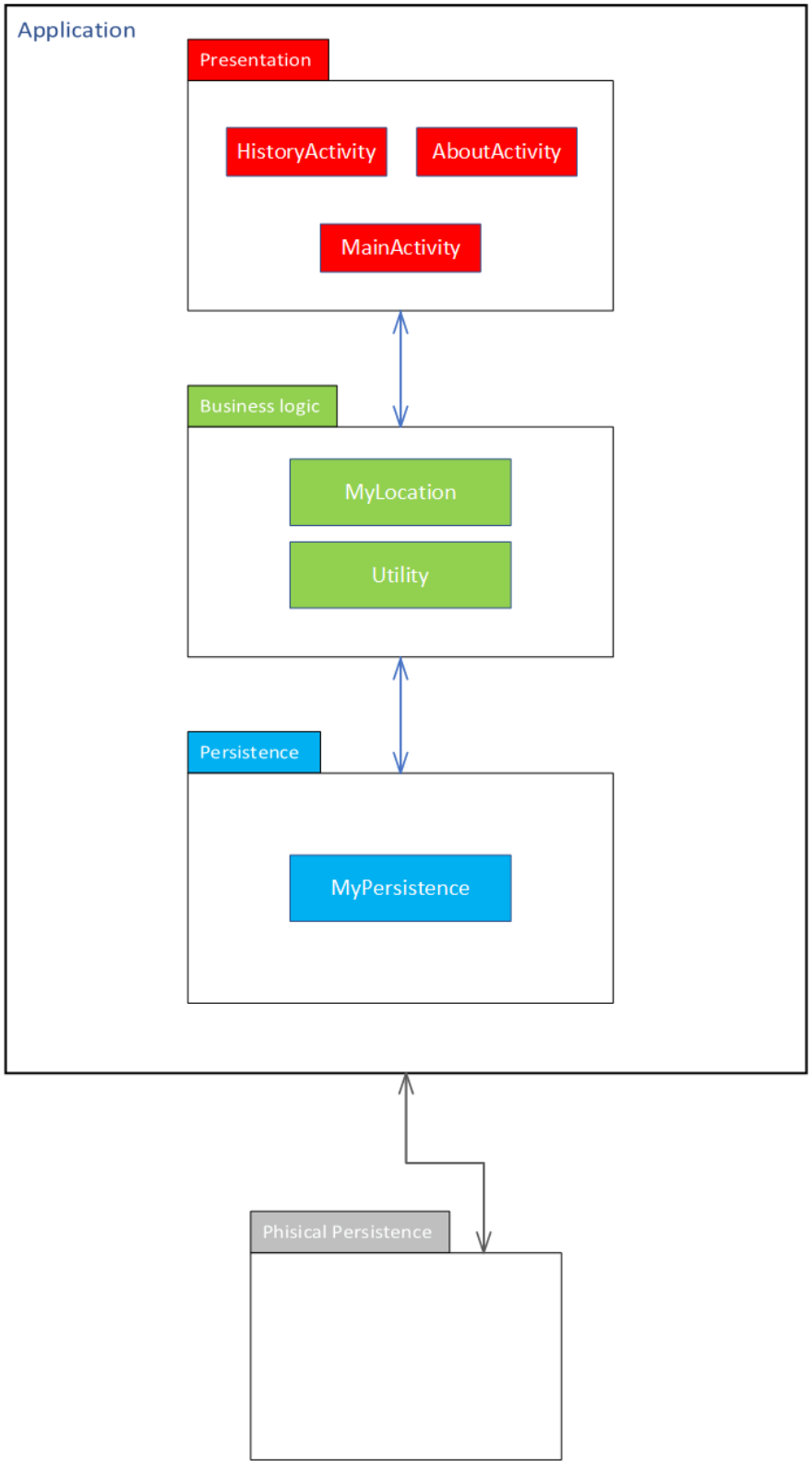

Figura 17 - Diagramma dell'architettura

Come si può notare dal diagramma sono presenti vari componenti relativi ai vari Layer dell'applicazione:

- Layer Presentazione: in questo layer sono presenti tre componenti, i quali rappresentano le tre activity visibili all'utente. La MainActivity rappresenta la schermata principale che si visualizza in ingresso all'applicazione e durante l'utilizzo di alcune delle sue funzionalità; La HistoryActivity rappresenta la schermata di storico con la lista dei percorsi salvati e le relative informazioni; infine la AboutActivity rappresenta la activity con al suo interno varie informazioni di utilizzo e di carattere generale dell'applicazione.

- Layer Logica di Business: in questo layer sono presenti due componenti, i quali vengono utilizzati e richiamati dopo una determinata interazione dell'utente con le schermate. In particolare abbiamo un componente che si occupa della localizzazione (MyLocation) e un componente che offre svariate funzionalità "laterali".

- Layer Persistenza: in questo layer è presente un unico componente, MyPersistence, che si occupa di effettuare le operazioni di salvataggio e ritrovamento delle informazioni riguardanti i percorsi effettuati.

Nel diagramma è inoltre indicato l'elemento fisico, Phisical Persistence, esterno all'applicazione su cui avverrà il salvataggio delle informazioni da parte del Layer di Persistenza. In questa versione dell'applicazione questo elemento è un file di testo noto a priori.

3.1.8 Descrizione dell'interfaccia grafica

Come detto in precedenza sono presenti tre Activity nell'applicazione, le quali offrono all'utente diverse interfacce grafiche. In base all'operazione che si sta eseguendo tali interfacce presenteranno all'utente diversi elementi visivi. Sotto vengono riportate immagini relative alle diverse interfacce.

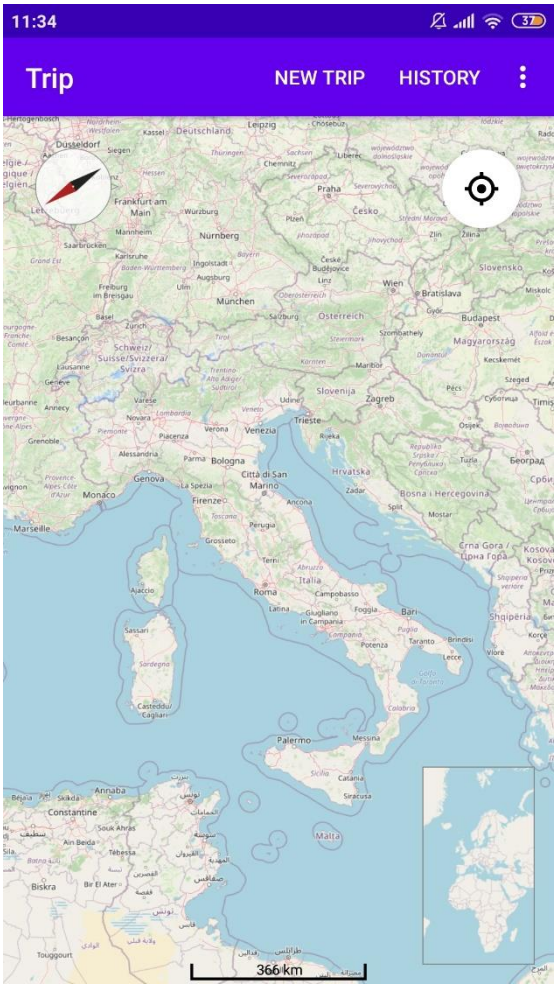

Figura 18 - Schermata iniziale

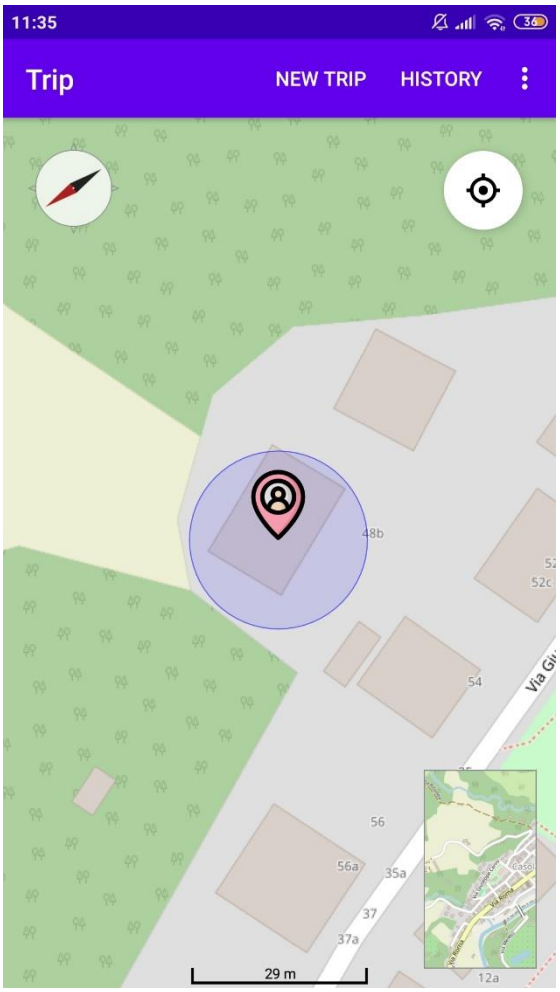

Figura 5 - Visualizzazione della propria posizione

La prima immagine a sinistra mostra cosa viene visualizzato dall'utente all'ingresso dell'applicazione, ossia la mappa con visione sull'Italia, una action bar (barra superiore) con vari elementi, bussola in alto a sinistra e pulsante "LOCATE ME" per ottenere la propria posizione.

La seconda immagine, quella a destra, mostra invece cosa si visualizza dopo che è stata richiesta e trovata la propria posizione corrente, ossia un marker sul punto calcolato (e un raggio che indica la

zona in cui la posizione reale dell'utente può effettivamente essere poiché la posizione centrale indicata non sempre è precisa).

Sotto sono invece presenti le immagini relative alla registrazione di un percorso:

- la prima mostra la schermata di dialogo che viene presentata all'utente quando questo clicca su "NEW TRIP", con la possibilità di inserire un eventuale titolo
- la seconda mostra la vista della mappa durante la registrazione effettiva del percorso che si sta effettuando
- la terza mostra la schermata di dialogo in cui viene richiesto se salvare oppure no il percorso appena terminato, dopo che si ha cliccato sul pulsante "STOP TRIP"
- la quarta mostra invece la finestra di dialogo in cui si avvisa l'utente che non è possibile iniziare a registrare un nuovo percorso poiché non è abilitata la localizzazione tramite GPS; nel caso si prema "OK" l'utente verrà ridirezionato nel menù delle impostazioni di localizzazione del proprio smartphone.

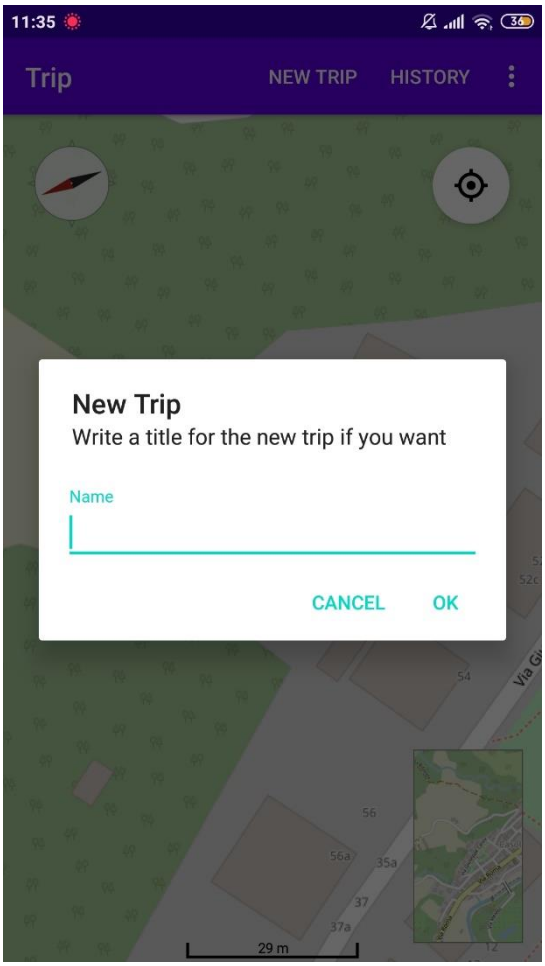

Figura 20 - Nuovo percorso

Figura 6 - Registrazione percorso

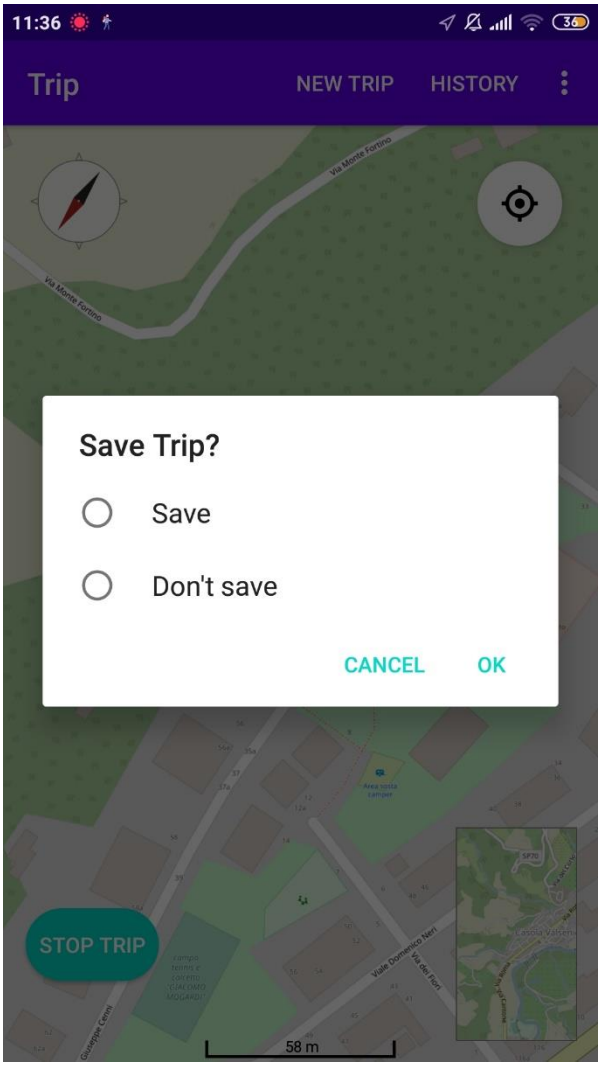

Figura 22 - Terminte percorso

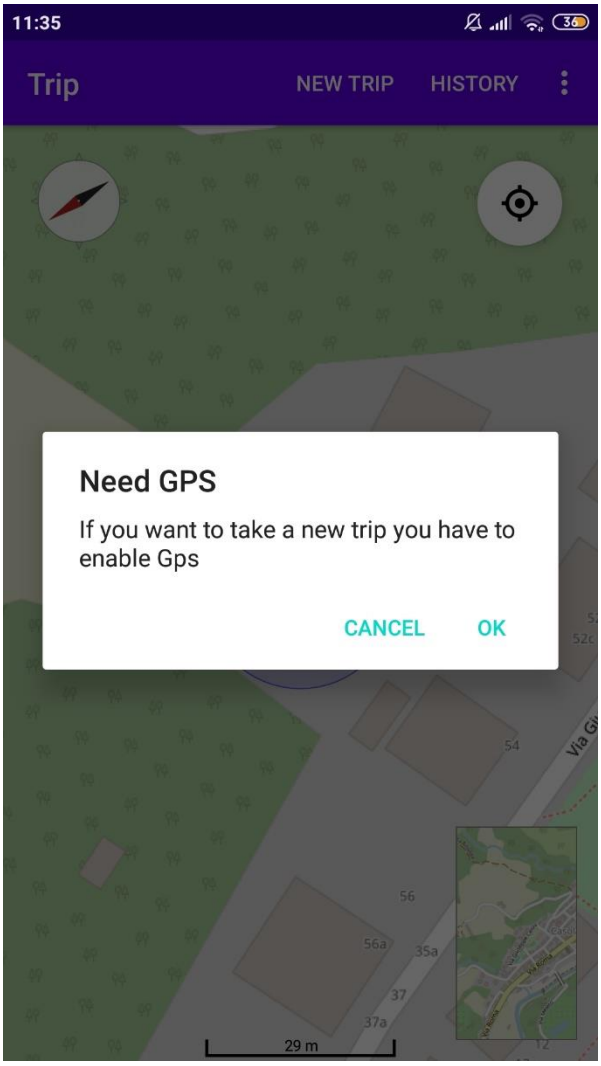

Figura 7 - Richiesta abilitazione GPS

Notare come in ogni finestra di dialogo è possibile premere sull'indicazione "CANCEL", tramite cui si torna alla schermata precedente con l'effetto di non aver effettuato nessuna operazione.

Sono presenti infine altre due schermate all'interno dell'applicazione:

- In basso a sinistra si può vedere la schermata relativa allo storico, ossia la lista di tutti i percorsi che sono stati salvati con le relative informazioni
- In basso a destra la schermata relativa alla pagina di informazioni sull'applicazione

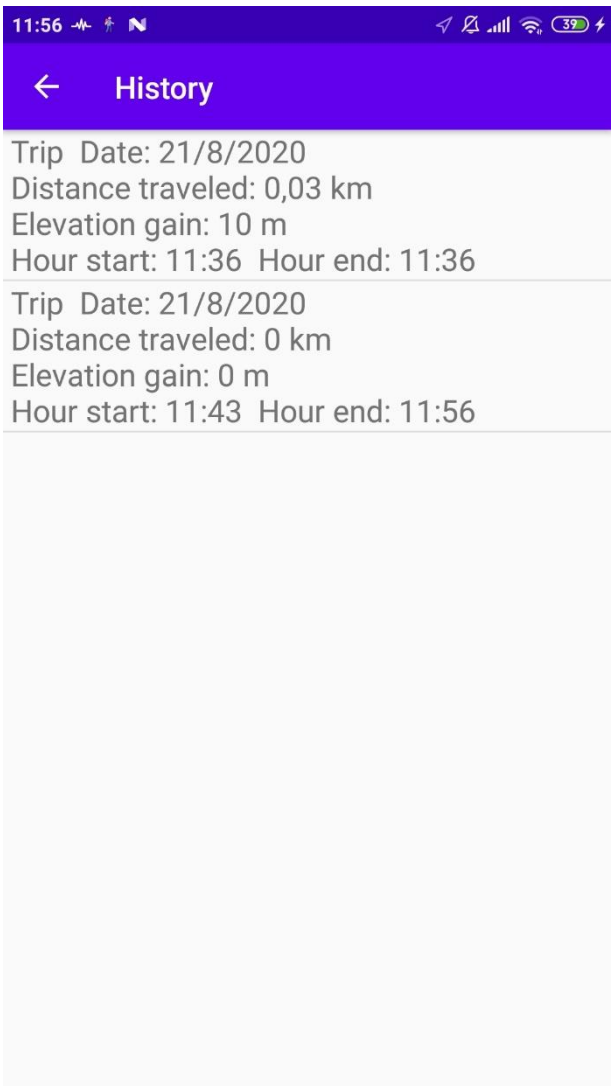

Figura 24 - Schermata storico

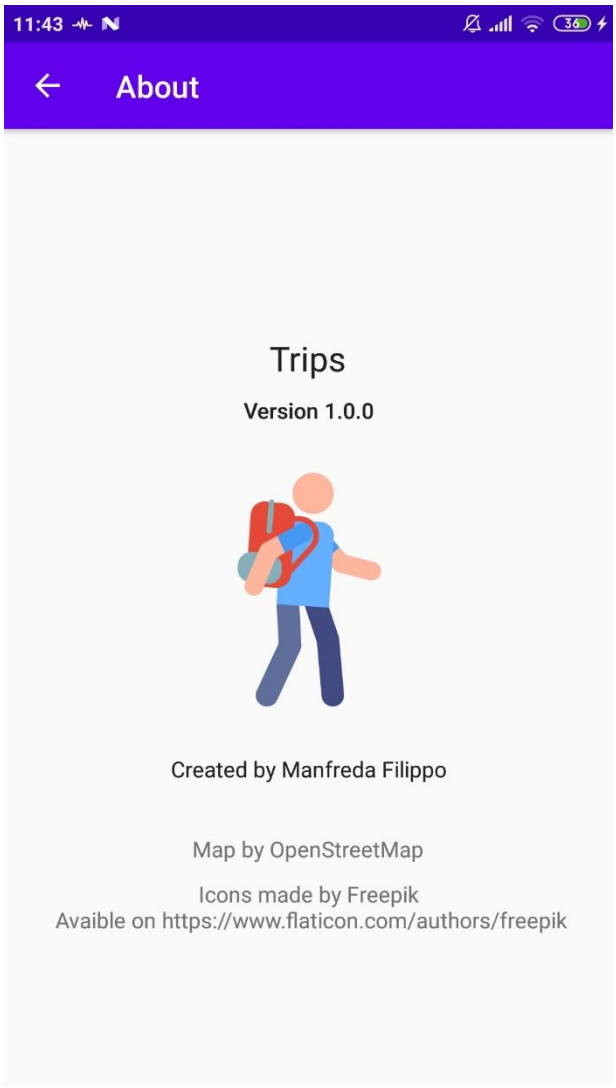

Figura 8 - Schermata di informazioni sulla applicazione

3.2 OpenStreetMap

OpenStreetMap (OSM) è un progetto collaborativo finalizzato a creare mappe del mondo a contenuto libero, ossia visualizzabile ed utilizzabile da chiunque. Il progetto punta ad una raccolta mondiale di dati geografici, con scopo principale quello della creazione di mappe e cartografie[17].

Figura 26 - Logo OpenStreetMap

La caratteristica fondamentale dei dati geografici presenti in OSM è che vengono distribuiti con una licenza libera, la Open Database License: è cioè possibile utilizzarli liberamente per qualsiasi scopo, anche commerciale, con il solo vincolo di citare la fonte e usare la stessa licenza per eventuali lavori derivati dai dati di OSM. Tutti possono contribuire arricchendo o correggendo i dati. Una alternativa proprietaria che riprendeva le pratiche di crowdsourcing introdotte dalla comunità di OpenStreetMap era Google Map Maker, nel quale i contributi dell'utente potevano essere inseriti su Google Maps a fronte di validazione, impedendone però il riutilizzo da parte di terzi.

Le mappe sono create usando come riferimento i dati registrati da dispositivi GPS portatili, fotografie aeree ed altre fonti libere. Sia le immagini renderizzate che i dati vettoriali, oltre che lo stesso database di geodati sono diffusi sotto licenza Open Database License. Il progetto OpenStreetMap è stato ispirato da altri come Wikipedia: la pagina in cui la mappa è consultabile (www.openstreetmap.org) espone in evidenza un'etichetta "Modifica" per procedere con la modifica dei dati ed il progetto è accompagnato da un archivio storico delle modifiche (cronologia e log). Gli utenti registrati possono caricare nei database del progetto tracce GPS e modificare i dati vettoriali usando gli editor forniti.

[17] https://it.wikipedia.org/wiki/OpenStreetMap

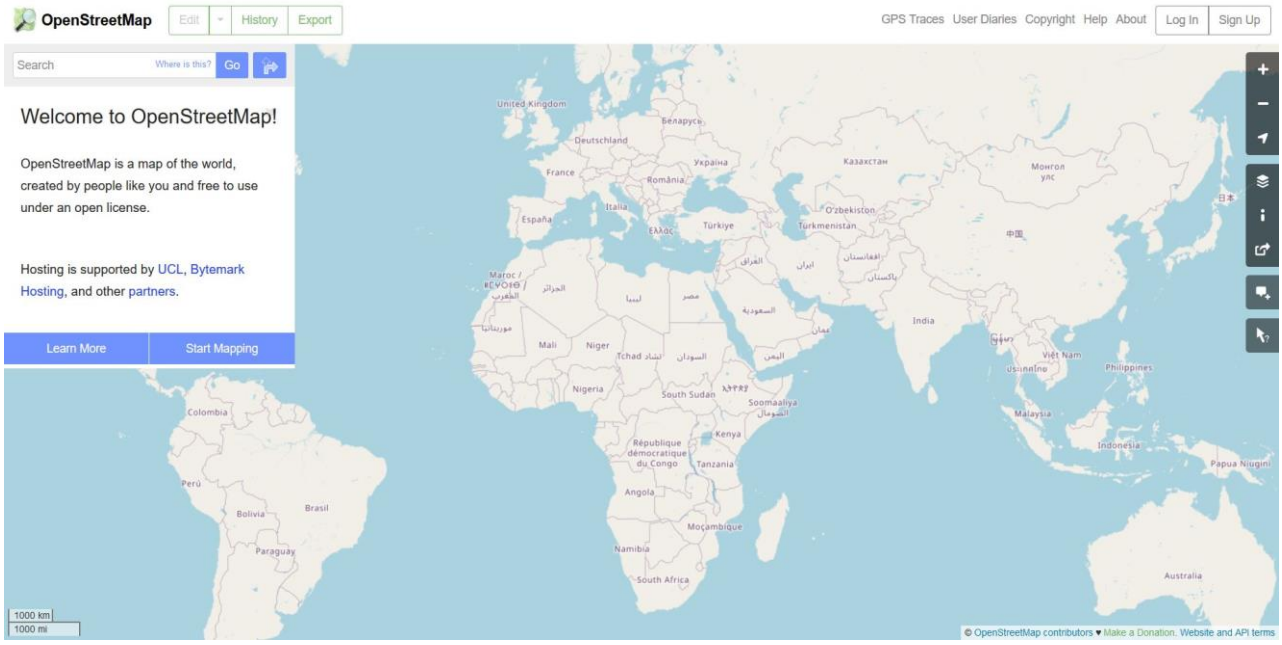

Figura 27 - Home page di OpenStreetMap

3.2.1 Tecnica

Generalmente, i dati per la realizzazione delle mappe sono ottenuti da volontari che effettuano rilievi sul territorio muniti di unità GPS capaci di registrare il percorso, a volte con l'ausilio di smartphone, fotocamera, registratori vocali, computer portatili; le informazioni raccolte vengono inviate via computer, o tramite alcune applicazioni su smartphone, nel database in formato vettoriale.

Molti collaboratori usano programmi come GPSBabel per convertire i dati GPS dal formato grezzo (NMEA) o da formati proprietari al formato GPX (un'applicazione dell'XML). I dati, come latitudine/longitudine, sono raccolti nel formato WGS84 e sono normalmente visualizzati sulla proiezione di Mercatore.

Più recentemente, la disponibilità di ortofoto aeree e di altri dati da enti commerciali e fonti governative ha incrementato la velocità e l'efficacia di questo lavoro, lasciando ai volontari il tempo per l'acquisizione di ulteriori dati legati al territorio. La conversione e l'importazione dei dati da grandi basi di dati, quando queste vengono rese disponibili con licenza libera, è affidata ad un team tecnico.

3.2.2 Rilievi sul territorio

I rilievi sul territorio vengono effettuati , come già detto, da volontari (autodefinitisi "mappatori", dall'inglese *mappers*) a piedi, in bici, auto, treno o qualsiasi altro mezzo di trasporto (benché la bicicletta rimanga il mezzo di trasporto preferito dai *mapper* nelle aree urbane), usando un'unità GPS, unitamente in alcuni casi a computer portatili, registratori vocali digitali e fotocamere digitali per la raccolta di dati (in alcuni casi si è rivelata utile anche la raccolta di informazioni mediante domande ai passanti). Alcuni contributori concentrano la propria attività nella propria città di residenza, raccogliendo dati in maniera sistematica. Inoltre, periodicamente vengono organizzati dai volontari dei *mapping party*, campagne di mappatura concentrate in un luogo preciso a cui i volontari partecipano divisi in squadre; queste attività possono durare anche più di un giorno nel caso di "mappature intensive".

Oltre alla raccolta di dati strutturata, un gran numero di modifiche al database viene effettuato da contributori che controllano i dati, aggiungono caratteristiche o correggono errori; per farlo, è sufficiente essere registrati al sito del progetto OSM.

3.2.3 La banca dati di OpenStreetMap

Il database di OpenStreetMap è pubblicato secondo la licenza ODbL - Open Database License. La licenza ODbL prevede che le modifiche ai dati devono essere rilasciate aperte, mentre, i prodotti derivati possono usare qualsiasi altra tipologia di licenza con il solo vincolo di citare la fonte. La cartografia ufficiale sviluppata sul sito del progetto, infatti fa uso della licenza Creative Commons Attribution. Esistono innumerevoli rappresentazioni dei dati di OpenStreetMap ciascuna con una propria licenza ma con sempre il riferimento alla sorgente dati.

3.2.4 Elementi della struttura del database OpenStreetMap

Gli elementi che possono essere inseriti (strade, negozi, ecc.), tramite diversi software, sono di quattro tipologie:

- punti(node): singoli punti
- linee(way): un insieme di punti non chiuso
- aree(polygon): un insieme di punti chiuso, solitamente con il tag area=yes
- relazioni(relation): un insieme degli elementi precedenti, per esempio una linea degli autobus che `e composta da più strade e dalle sue fermate

3.2.4.1 Tag

Le etichette (tag) servono per descrivere le caratteristiche dei vari elementi. I tag sono sempre composti da una coppia di nomi. Il primo è detto "key", il secondo "value". [18]Solitamente la "key" descrive una famiglia di caratteristiche, mentre "value" va più nello specifico. Ad esempio la "key" "highway" indica la famiglia delle strade di qualsiasi tipo, dalle autostrade ai sentieri. Di lato ne sono presenti alcuni.

key	values	descrizione
highway	motorway	autostrada
	trunk	superstrada
	primary	strada di importanza nazionale
	secondary	strada di importanza regionale
	tertiary	strada di importanza locale
	unclassified	strada del reticolo di base
	residential	strada per abitazioni
	service	strada di servizio
	track	strada agricola o forestale
	pedestrian	via pedonale cittadina
	footway	sentiero
	cycleway	pista ciclabile
	steps	scala
	bus_stop	fermata dell'autobus
	stop	segnale stop
	traffic_signals	semaforo

Figura 28 - Tabella relativa al tag "highway"

I tag non rappresentano solo un elemento, ma possono essere usati per più elementi: per esempio "highway" è prevalentemente

[18] https://wiki.openstreetmap.org

associato alle linee ma, come si può vedere a lato, vi sono alcuni casi in cui è utilizzato con i nodi "highway=bus_stop" o "highway=traffic_signals". I tag usati sono tantissimi e continuano ad aumentare e migliorare, permettendo di mappare qualsiasi elemento possa essere rappresentato da una coppia di coordinate geografiche.

Oltre ai tag per le strade, esistono molti tag per elementi puntuali, lineari e areali. Eccone alcuni:

elemento	key	value	descrizione
puntuale	**amenity**	pub	pub
		bank	banca
	shop	supermarket	supermercato
		bakery	panificio
	tourism	hotel	albergo o hotel
		information	punto informazioni turistiche
	railway	station	stazione ferroviaria
		level_crossing	passaggio a livello
lineare	**aerialway**	cable_car	funivia
		chair_lift	seggiovia
	waterway	river	fiume
		canal	canale
	railway	rail	ferrovia
		tram	linea tram
areale	**natural**	water	fiume molto largo o lago
		wood	foresta
	leisure	playground	parco giochi
		sport_center	stadio
	landuse	residential	zona residenziale
		vineyard	vigneto

Figura 29 - Altri tipi di Tag

Inoltre è bene essere a conoscenza che per ciascun elemento è possibile assegnare più di un tag in modo da descriverlo nella maniera più completa, ad esempio:

key	value
highway	unclassified
name	Via Roma
foot	yes
bicycle	no
oneway	yes

Figura 30 - Tag associati ad uno stesso elemento

3.2.4.3 Relation

Le relation servono per mettere in relazione più elementi singoli, diverse vie oppure vie e punti. Esistono svariate tipologie di relation, di seguito ne vengono mostrate alcune e poi sarà approfondita la route, che è una tra le più importanti e usate.

tipo	descrizione
associatedStreet	serve per mettere in relazione le vie con i numeri civici
multipolygon	serve per creare poligoni complessi che si compongono di più di una way (per esempio un'area con un buco come un lago con un'isola in mezzo)
restriction	serve per vietare le svolte
boundary	serve per raggruppare aree e creare enclavi ed exclavi
route	serve per creare dei percorsi, possono essere pedonali (per esempio sentieri montani), ciclabili, linee di trasporti pubblici ecc ecc
enforcement	serve per inserire elementi per misurare e documentare le violazioni veicolari
destination sign	serve per dare informazioni sulle destinazioni agli incroci e alle svolte (molto utile per i programmi di routing)

Figura 31 - Esempi di Relation

La relation route serve per indicare che più way diverse fanno parte di una stessa rotta. Gli unici due campi obbligatori sono “type” e “route”; gli altri, pur non essendo richiesti, sono ugualmente importanti e, quando presenti, è bene utilizzarli.

key	value
type	route
route	road - bicycle - foot - hiking - bus - ferry - canal -pilgrimage - detour - railway - tram - trolleybus - mtb (mountainbike) - roller_skate - running - horse - parade - protest_march (recurring)
ref	codice identificativo se presente
operator	nome dell’operatore se presente
name	nome se presente
symbol	simbolo se presente

Figura 32 - Relation route

Per creare una route bisogna quindi aggiungere i tag e poi selezionare quali elementi già presenti sul database ne fanno parte.

Capitolo 4 : Implementazione e studio dei consumi

In questo ultimo capitolo verranno trattati nello specifico l'implementazione, quindi il codice e la sua strutturazione con richiami sull'IDE utilizzato per la scrittura dell'applicazione, ossia Android Studio, e osmdroid, una libreria sviluppata per l'utilizzo di OpenStreetMap. Infine verrà presentato uno studio sui consumi relativi all'utilizzo di tale applicazione.

4.1 Android Studio

Android Studio è un ambiente di sviluppo integrato (IDE) per la piattaforma Android. Basato sul software IntelliJ IDEA dell'azienda JetBrains[19]. Android Studio è stato progettato specificamente per lo sviluppo di applicazioni Android.

Android Studio presenta diverse funzionalità specifiche proprio per la progettazione e lo sviluppo di app:

Figura 33 - Logo Android Studio

- Un sistema di compilazione flessibile basato su Gradle (sistema open source per l'automazione dello sviluppo)
- Un ambiente unificato in cui è possibile sviluppare applicazioni per tutti i dispositivi Android
- Strumenti e framework di test estesi
- Supporto per C ++ e NDK
- Templates di codice e integrazione con GitHub per velocizzare lo sviluppo di parti comuni e per importare esempi di codice
- Integrazione con un sistema VCS (Version Control System) per semplificare la gestione delle varie versioni dell'applicazione e poter tornare a implementazioni precedenti con semplicità

[19] https://it.wikipedia.org/wiki/Android_Studio

- Un emulatore di smartphone veloce e ricco di funzionalità in cui è possibile testare l'applicazione in sviluppo
- Uno strumento di profilazione relativo all'esecuzione dell'applicazione per monitorarne i consumi, utilizzato per la parte sperimentale

4.1.1 Struttura del progetto

Ogni progetto in Android Studio contiene uno o più moduli con file di codice sorgente e file di risorse[20]. I tipi di moduli includono:

- Moduli di app Android
- Moduli di libreria
- Moduli di Google App Engine

Per impostazione predefinita, Android Studio visualizza i file di progetto nella vista del progetto Android, come mostrato nella figura a lato. Questa vista è organizzata in moduli per fornire un accesso rapido ai file principali del progetto.

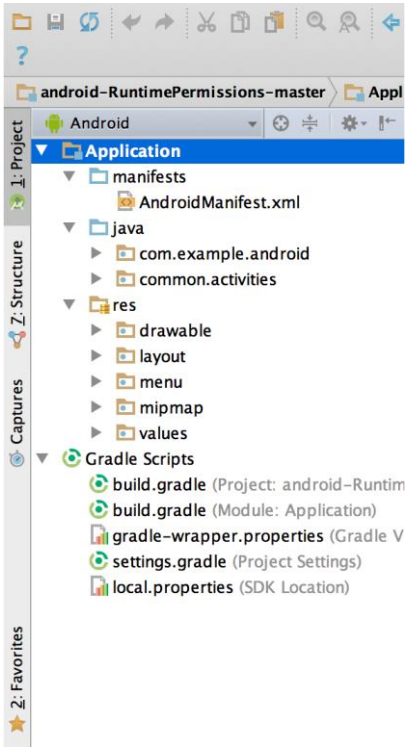

Figura 34 - File del progetto nella vista "Android"

Tutti i file di "build" sono visibili al livello superiore in Gradle Scripts e ogni modulo dell'app contiene le seguenti cartelle:

- Manisfests: contiene il file AndroidManifest.xml.
- java: contiene i file del codice sorgente Java, incluso il codice di prova JUnit.
- res: contiene tutte le risorse non di codice, come layout XML, stringhe dell'interfaccia utente e immagini bitmap.

[20] https://developer.android.com/studio/intro

La struttura del progetto Android su disco è diversa da questa rappresentazione. Per vedere l'effettiva struttura dei file del progetto è necessario selezionare “Project” dal menu a tendina in alto a sinistra.

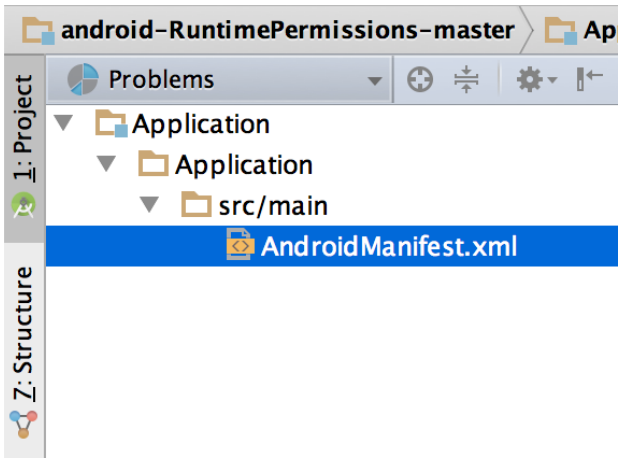

Figura 35 - vista "Problems"

E’ possibile anche personalizzare la visualizzazione dei file di progetto per focalizzarsi su aspetti specifici dello sviluppo dell’applicazione. Ad esempio, selezionando la vista “Problems” del progetto vengono visualizzati i collegamenti ai file di origine contenenti eventuali errori di codifica e sintassi riconosciuti, come per esempio un tag di chiusura dell'elemento XML mancante in un file di layout.

4.2 Osmdroid

Osmdroid è una libreria di codice open source sviluppata per sostituire la classe MapView di Android, la quale a differenza di osmdroid, per essere utilizzata su progetti necessita la creazione di un account con determinate caratteristiche presso Google.

Tale libreria viene utilizzata principalmente per rappresentare e visualizzare una mappa con dati ottenuti dai server di OpenStreetMap, progetto spiegato nel capitolo precedente.

Include anche un sistema di provider di "tile" diversi (insieme di elementi utilizzati per rappresentare graficamente la mappa), con supporto per numerose origini dei dati online e offline e inoltre supporto di elementi in "overlay"(in sovrapposizione), con determinati "overlay" integrati per la rappresentazione di icone, la posizione, il tracciamento e il disegno di forme.[21]

Per poter visualizzare in una schermata una mappa è necessario definire nel layout relativo all'activity corrente l'elemento MapView, successivamente nell'activity indicare la fonte del tile tramite `map.setTileSource(TileSourceFactory.MAPNIK)`. L'indicazione "MAPNIK" identifica una determinata sorgente del tile poiché sono presenti diverse sorgenti ed in fase di inizializzazione è necessario indicare quale si vuole utilizzare.
Come detto sono presenti e disponibili all'utilizzo diversi overlay da collegare alla mappa, i più importanti sono:

- MyLocation overlay: utilizzato per disporre del servizio di localizzazione costante del proprio dispositivo
- CompassOverlay: sviluppato per inserire in sovrapposizione alla mappa una bussola che indica i punti cardinali tramite l'utilizzo del giroscopio del dispositivo
- Grid line overlay: si utilizza per inserire una griglia di linee relative alla latitudine e longitudine
- Map Scale bar overlay: tramite questo overlay si inserisce una linea che indica in scala quanto la lunghezza di essa misuri nella realtà
- Rotation Gestures overlay: offre la possibilità di ruotare la mappa a piacimento durante la visualizzazione
- Minimap overlay: inserisce una mini-mappa in sovrapposizione a quella già presente

[21] https://github.com/osmdroid/osmdroid/wiki

4.3 Implementazione

Viene ora presentata la parte implementativa del progetto dell'applicazione, indicando quali classi sono state realizzate e descrivendo le parti di codice più significative.

4.3.1 Pattern Factory

Durante l'implementazione è risultato necessario adottare un altro pattern oltre a quello già indicato in fase di progettazione, il patter Factory. Tale scelta è risultata necessaria per poter rispettare il "Dependency Inversion Principle", ossia non far dipendere i moduli di alto livello da quelli di basso livello in modo diretto ma tramite astrazioni. Tale pattern si è implementato con una classe "Factory" che conosce la reale implementazione dei vari moduli, ed in base alle richieste ricevute fornisce un'istanza del ralativo oggetto.

Quindi per poter soddisfare il principio indicato sopra sono state create diverse interfacce: un'interfaccia per il modulo relativo alla localizzazione, "MyLocation", una per il modulo relativo alla persistenza, "MyPersistence" ed infine una per il modulo relativo a metodi di utilità secondari, "MyUtility".

Grazie a queste nei moduli, relativamente al contesto, di più alto livello è stato sufficiente dichiarare variabili del tipo dell'interfaccia senza conoscerne la reale implementazione e richiederne successivamente l'istanziazione ad un oggetto di classe Factory.

Come si può vedere dall'immagine sottostante, l'istanziazione dell'oggetto avviene nella classe Factory al momento della richiesta da parte del modulo di più alto livello. In particolare sono presenti tre metodi, relativi alla localizzazione, persistenza e utility.

```
class Factory {

    @RequiresApi(Build.VERSION_CODES.N)
    fun getLocation(context : Context, map : MapView, locationManager : LocationManager) : MyLocation{
        var retVal: MyLocation?

        retVal = LocationImp(context, map, locationManager)

        return retVal

    }

    fun getPersistence(context : Context): MyPersistence{
        var retVal : MyPersistence? = null

        retVal = PersistenceImp(context)

        return retVal
    }

    fun getUtility() : MyUtility{

        var retVal : MyUtility? = null

        retVal = UtilityImp()

        return retVal

    }
}
```

Figura 36 - Classe Factory

Il fulcro dell'applicazione risiede però in altre classi, suddivise nei vari package indicati nel diagramma strutturale presente nel capitolo di progettazione:

- Per quanto riguarda il package "Business logic" sono presenti le classi LocationImp che si occupa di tutte le funzionalità relative alla localizzazione e UtilityImp che si occupa dei metodi di utilità secondari
- Per quanto riguarda il package "Persistence" è presente la classe PersistenceImp la quale si occupa del salvataggio e ritrovamento dei dati
- Per quanto riguarda il package "Presentation" sono presenti le tre activity dell'applicazione: MainActivity, AboutActivity e HistoryActivity.

4.3.2 Package Business Logic

Nel package “Business Logic”, come detto sopra, sono presenti le classi LocationImp e UtilityImp.

La classe UtilityImp presenta metodi per calcolare quanto, in termini di tempo, è aggiornata una determinata posizione ed essendo “secondaria” non risulta necessario approfondirne il contenuto.

Più interessante è il codice della classe LocationImp, che si occupa delle funzionalità principali dell’applicazione.

Prima tra queste è il ritrovamento della posizione corrente, che avviene tramite vari step, come si può vedere dall’immagine sottostante. Prima viene effettuato un controllo su quali provider sono disponibili, si preferisce il provider GPS se disponibile poiché è il più preciso nella localizzazione; dopo tale operazione viene controllato se è presente in memoria una posizione e nel caso se questa risulta non più vecchia di trenta secondi: in caso positivo si imposta la mappa su tale posizione tramite la chiamata al metodo “setLocation”, altrimenti viene effettuata una chiamata al metodo “requestSingleUpdate” che notifica la nuova posizione appena disponibile al location listener fornito al metodo; appena è presente l’aggiornamento il location listener chiamerà il metodo “setLocation” che imposterà la mappa sulla nuova posizione.

```
hasGps = locationManager!!.isProviderEnabled(LocationManager.GPS_PROVIDER)
hasNetwork = locationManager!!.isProviderEnabled(LocationManager.NETWORK_PROVIDER)

if (hasGps || hasNetwork) {
    provider = if (hasGps) {
        LocationManager.GPS_PROVIDER
    } else {
        LocationManager.NETWORK_PROVIDER
    }

    return if (loc != null &&  util.ageMs(loc!!) < 30000) {
        setLocation(loc!!)
        true
    } else {
        locationManager!!.requestSingleUpdate(provider!!, locationSingleListener,  looper: null)
        return true
    }
}else {return false }
```

Figura 37 - Vista sul metodo che si occupa del calcolo della posizione corrente

L’altro metodo principale presente in tale classe è quello che riguarda l’inizio della registrazione di un nuovo percorso, il cui nucleo è visibile nell frammento di codice sottostante.

```
    startTime = Calendar.getInstance().get(Calendar.HOUR_OF_DAY)
        .toString() + ":" + Calendar.getInstance().get(Calendar.MINUTE).toString()
    follow = true
    locationManager!!.requestLocationUpdates(
        LocationManager.GPS_PROVIDER,
        minTime: 10000,
        minDistance: 10F,
        locationListener
    )
}
```

Figura 38 - Frammento di codice riguardante l'inizializzazione della registrazione di un nuovo percorso

Successivamente alla richiesta da parte dell’utente di inizare a registrare i dati di un nuovo percorso, viene chiamato il metodo “startTrip”, le cui operazioni principali sono quelle mostrate nella immagine soprastante: viene salvato il momento di inizio registrazione, abilitato “l’inseguimento” della mappa su ogni nuova posizione e poi viene effettuata la chiamata al metodo “requestLocationUpdates”, il quale richiede al provider continui aggiornamenti della posizione in base ai parametri indicati (in questo caso dieci secondi come tempo minimo e dieci metri come distanza minima). Gli aggiornamenti verranno notificati al location listener indicato, che si occuperà di aggiornare la mappa e i dati della registrazione.

4.3.3 Package Persistence

Per quanto riguarda la persistenza, in particolare la classe PersistenceImp, non risulta necessario inserire descrizioni del codice poiché si occupa esclusivamente di salvare i dati raccolti, recuperare quelli presenti in un determinato file ed eliminare quelli di cui l'utente non necessità più, tutte funzionalità non interessanti per quanto riguarda la trattazione della tesi.

4.3.4 Package Presentation

In questo package, come detto in precedenza, sono presenti tre Activiy: MainActivity, HistoryActivity e AboutActivity.

Queste tre rappresentano le schermate che l'utente visualizzerà durante l'utilizzo dell'applicazione, descritte graficamente nel capitolo precedente.

La AboutActivity non presenta comportamenti degni di nota, per cui la trattazione viene fatta sulle altre due Activity.

Partendo dalla MainActivity, ossia l'activity principale dell'applicazione e quella che viene mostrata in ingresso, nelle immagini sottostanti è possibile visualizzare il comportamento iniziale in cui viene impostata la mappa e gli elementi ad essa collegati.

```
//set the map
mapView.setTileSource(TileSourceFactory.DEFAULT_TILE_SOURCE)
mapView.setMultiTouchControls(true)

//scale bar
val dm : DisplayMetrics!  = ctx.resources.displayMetrics
val mScaleBarOverlay = MyScaleBarOverlay(mapView)
mScaleBarOverlay.setAlignBottom(true)
mScaleBarOverlay.setScaleBarOffset( x: dm.widthPixels / 2,  y: 10)
mapView.overlays.add(mScaleBarOverlay)
```

Figura 39 - Impostazione iniziale della mappa e della barra

In questa prima immagine si può notare l'impostazione del "tile" e della possibilità di utilizzare più tocchi sullo schermo per interagire con la mappa, successivamente si imposta la barra con le indicazioni di lunghezza aggiungendola agli "overlay" della mappa.

Sotto invece è possibile visualizzare l'impostazione degli altri elementi visibili graficamente, ossia la minimappa e la bussola, elementi che si aggiungono anche in questo caso agli "overlay" della mappa; mentre a lato è mostrato come vengono impostati i metodi di callback relativi alla pressione dei due bottoni, il primo per la richiesta della posizione corrente, il secondo per la richiesta di terminare la registrazione corrente.

```
//set the callback for the buttons
floatingActionButton.setOnClickListener{ it: View!
    if(!locMe.findMe()){requestOption( type: "")}
}

btn_stop_trip.setOnClickListener{ it: View!
    requestConfirm()
}
```

Figura 40 - Impostazione metodi di callback per i bottoni

```
//minimap
val mMinimapOverlay = MyMinimapOverlay(ctx, mapView.tileRequestCompleteHandler)
mMinimapOverlay.width = dm.widthPixels / 5
mMinimapOverlay.height = dm.heightPixels / 5
mMinimapOverlay.zoomDifference = 5
mapView.overlays.add(mMinimapOverlay)

mScaleBarOverlay.setScaleBarOffset( x: dm.widthPixels / 2,  y: 10)
mapView.overlays.add(mScaleBarOverlay)

//set the compass
val compassOverlay = MyCompassOverlay(
    ctx,
    InternalCompassOrientationProvider(ctx),
    mapView
)
compassOverlay.enableCompass()
mapView.overlays.add(compassOverlay)
```

Figura 41 - Impostazione degli ulteriori elementi visibili sulla mappa

Un'ultima parte di tale activity che risulta interessante per quanto riguarda il funzionamento del programma è quella mostrata nell'immagine successiva.

Tale immagine riguarda le azioni che vengono effettuate in base alla pressione di uno degli elementi del menu, presente in quella che è chiamata “action bar” visibile in alto. Gli elementi due e tre del menu portano alla richiesta di attivazione di una nuova activiy, HistoryActivity e AboutActivity; mentre il primo elemento riguarda la registrazione di un nuovo percorso: si controlla se è abilitata la localizzazione tramite GPS, in caso negativo viene chiamato il metodo “requestOption” che si occuperà di informare l’utente della necessità di abilitare il GPS, in caso positivo verrà chiamato il metodo “requestTitle” che si occuperà di avviare la registrazione del nuovo percorso partendo dalla richiesta di un eventuale titolo.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    //method used when clicking on option of the menu/action bar
    when (item.itemId) {
        R.id.MENU_1 -> {
            hasGps = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)

            if (!hasGps) {
                requestOption( type: "GPS")
            } else {
                requestTitle()
            }
        }
        R.id.MENU_2 -> {
            val i = Intent( packageContext: this, HistoryActivity::class.java)
            startActivity(i)
        }
        R.id.MENU_3 -> {
            val ii = Intent( packageContext: this, AboutActivity::class.java)
            startActivity(ii)
        }
    }
    return false
}
```

Figura 42 - Azioni del menu

Si conclude la descrizione dell'implementazione con la HistoryAcitivity, la quale si occupa di mostrare all'utente i dati dei percorsi salvati e offre la possibilità di eliminare quelli che l'utente non vuole più visionare.

```
val adapter : ArrayAdapter<String!>  = ArrayAdapter( context: this, R.layout.row, list)
val listView: ListView = listView
listView.adapter = adapter
listView.onItemClickListener = AdapterView.OnItemClickListener {
        parent, view, position, id ->
    // remove clicked item from list view
    val mBuilder = AlertDialog.Builder( context: this)
    mBuilder.setTitle("Delete Trip?")

    mBuilder.setPositiveButton(
        text: "Yes"
    ) { _, _ ->
            Toast.makeText(this.applicationContext,  text: "Deleted", Toast.LENGTH_SHORT).show()
            pers.removeLine(position)
            list.removeAt(position)
            if(list.isEmpty()) {empty.visibility = View.VISIBLE}
            adapter.notifyDataSetChanged()
    }
    mBuilder.setNegativeButton(
        text: "No"
    ) { dialog, _ -> dialog.cancel() }

    val mDialog : AlertDialog!  = mBuilder.create()
    mDialog.show()
```

Figura 43 - Creazione della schermata con la lista di elementi e possibilità di eliminazione

Dall'immagine è possibile vedere come, dopo aver recuperato tramite un'operazione non mostrata in questo frammento di codice la lista di percorsi salvati grazie alla classe di persistenza, si crea un ArrayAdapter (componente che si occupa della rappresentazione grafica dei dati e dell'interazione con essi) popolato con la lista e successivamente associato alla ListView (elemento grafico che mostra le informazioni sotto forma di lista) per la vera visualizzazione a schermo.

La parte successiva del codice mostra come viene impostata l'interazione degli elementi della ListView, ossia quali operazioni vengono effettuate dopo la pressione di uno degli elementi. In questo caso viene richiesto all'utente se vuole eliminare l'elemento oppure no: in caso positivo si procede all'eliminazione di tale elemento dal file che contiene tutti i dati chiamando il metodo "removeLine" della classe di persistenza, si aggiorna la lista corrente ed infine si aggiorna la schermata mostrata all'utente tramite il metodo "notifyDataSetChanged".

4.4 Analisi sperimentale e studio dei consumi

In questa ultima parte della tesi vengono presentati ed illustrate le analisi sperimentali e gli studi sui consumi eseguiti durante l'esecuzione dell'applicazione sviluppata.

Gli strumenti utilizzati per tali operazioni sono:

- Due smartphone: uno Xiaomi Mi Max 2 e un Samsung Galaxy A51
- L'applicazione sviluppata
- Un computer portatile per la rilevazione di determinati dati

Sono stati raccolti su ogni smartphone utilizzato, dati e informazioni relativi a diversi indicatori:

- La latenza dell'applicazione per quanto riguarda il calcolo della posizione corrente
- L'utilizzo della CPU
- L'utilizzo di memoria
- Il consumo di batteria

I due smartphone utilizzati presentano caratteristiche tecniche diverse, le principali e utili per la sperimentazione sono indicate nella tabella sottostante. Inoltre è bene specificare che lo Xiaomi, per quanto riguarda il GPS, supporta la tecnologia A-GPS, GLONASS e BeiDou mentre il Samsung oltre a queste anche Galileo.

	CPU	RAM	Capacità Batteria
Xiaomi Mi Max 2	Qualcomm Snapdragon 625 Octa-Core 2.0 GHz clock speed	4 GB	5300 mAh
Samsung Galaxy A51	SAMSUNG Exynos 9611 Quad-Core 2.3 GHz clock speed Quad-Core 1.7 GHz clock speed	4 GB	4000 mAh

Tabella 1 - Dati tecnici degli smartphone

4.4.1 Latenza

La latenza in informatica e telecomunicazioni, indica in un sistema di elaborazione dati e/o di telecomunicazioni, l'intervallo di tempo che intercorre fra il momento in cui viene inviato l'input/segnale al sistema e il momento in cui è disponibile il suo output. In altre parole, la latenza non è altro che una misura della velocità di risposta di un sistema[22].

Nel caso di studio di tale tesi è stata registrata la latenza relativa alla localizzazione del dispositivo, in particolare l'intervallo di tempo che intercorre tra la pressione del pulsante per richiedere la posizione corrente e la risposta visibile dell'aggiornamento della posizione.

E' stato valutato questo tempo in diverse modalità di calcolo della posizione:

- Tramite l'utilizzo del WiFi, con potenza del segnale massima
- Tramite l'utilizzo della rete mobile, utilizzando lo stesso operatore (Iliad)
- Tramite l'utilizzo del modulo GPS

Si è proceduto effettuando nei diversi casi serie di 15 misurazioni effettuate nello stesso momento e nelle stesse condizioni esterne con i due dispositivi, ogni misurazione distante 15 secondi dalla precedente. Nelle tabelle sottostanti sono visibili le misurazioni effettuate in tutti i casi presi in considerazione. Partendo da sinistra sono presenti i dati relativi al WiFi, poi quelli relativi alla rete mobile ed infine quelli relativi al GPS(non considerando la prima taratura tra il GPS ed i satelliti).

Xiaomi	Samsung
1.90	3.05
1.80	3.07
1.78	2.97
1.83	3.01
1.81	3.06
1.70	2.95
1.83	2.98
1.77	2.94
0.65	2.98
1.77	2.96
1.83	2.90
1.77	2.98
1.64	2.98
1.70	2.91
1.77	3.000

Tabella 4 - Dati Wifi

Xiaomi	Samsung
1,57	2,62
1,39	2,65
1,29	2,60
1,32	2,51
1,26	2,37
1,29	2,45
1,15	2.45
1,29	2,62
1,18	2,58
1,20	2,52
1,31	2,53
1,39	2,45
1,29	2,48
1,13	2,41
1,15	2,84

Tabella 3 - Dati Rete mobile

Xiaomi	Samsung
2.82	3.57
2.62	4.75
1.58	3.36
2.43	2.95
2.30	3.24
2.75	3.49
1.57	4.93
2.23	4.38
1.63	4.77
1.51	3.21
1.57	3.45
2.88	3.06
3.34	4.54
3.27	2.51
3.20	1.63

Tabella 2 - Dati GPS

[22] https://it.wikipedia.org/wiki/Latenza

I valori medi per ogni serie, scartando il valore massimo e minimo, risultano:

- WiFi: Xiaomi 1,77 s, Samsung 2,98 s
- Rete mobile: Xiaomi 1,27 s, Samsung 2,53 s
- GPS: Xiaomi 2,37 s, Samsung 3,64 s

Dai valori medi calcolati è possibile notare come la latenza risulti sempre minore nello Xiaomi, in particolare risultando più reattivo all'incirca di 1,2/1,3 s.

In entrambi i dispositivi la minore latenza si è registrata durante l'utilizzo della rete mobile, mentre quella maggiore durante il calcolo della posizione tramite il GPS.

Inoltre è importante sottolineare come le misurazioni con accuratezza maggiore della posizione sono risultate essere quelle con il GPS, mentre quelle con minore accuratezza sono risultate essere quelle con la rete mobile.

Concludendo è possibile affermare che il tipo di localizzazione "preferibile" in ambienti interni,come miglior compromesso tra latenza e precisione, risulta essere quello WiFi, ricordando che il GPS non funzioniona in tali ambienti; mentre per quando riguarda l'esterno la tipologia di calcolo dipende dalle necessità, considerando però che la localizzazione tramite rete mobile potrebbe essere molto poco accurata.

4.4.2 Utilizzo di CPU

In uno smartphone la CPU, “central processing unit”, rappresenta il componente principale per l’esecuzione di qualsiasi operazione necessaria per il funzionamento del dispositivo. Per tale motivo quando si sviluppano applicazioni è necessario considerare quanto l’esecuzione della nuova applicazione impatterà sulla CPU, per garantire che l’app non si trasformi in un collo di bottiglia per il funzionamento generale del dispositivo. I due dispositivi utilizzati per le misurazioni presentano CPU diverse, come indicato nella tabella di specifiche precedentemente mostrata.

Per monitorare l’utilizzo di CPU da parte dall’applicazione sono state effettuate tre diverse misurazioni per ogni dispositivo utilizzando il profiler presente nell’IDE Android Studio, incentrate sulla funzionalità principale dall’applicazione, ossia quella di registrazione di un percorso. Le tre misurazioni differiscono l’una dall’altra per la frequenza di aggiornamento del calcolo della nuova posizione: una misurazione con frequenza di aggiornamento pari a cinque secondi, una con frequenza pari a dieci secondi ed una infine pare a quindici secondi.

Ogni misurazione ha avuto una durata di tre minuti con l’utilizzatore costantemente in movimento.

Sotto è possibile visualizzare i grafici relativi alle varie misurazioni partendo da quelle effettuate sullo Xiaomi con iniziale frequenza di aggiornamento di cinque secondi. In particolare i grafici mostrano l’utilizzo in percentuale di CPU da parte dell’applicazione, mostrato dalle aree verdi, durante lo scorrere del tempo dei primi momenti di utilizzo poiché sono risultati i più importanti e significativi dato che quelli successivi si ripetevano allo stesso modo.

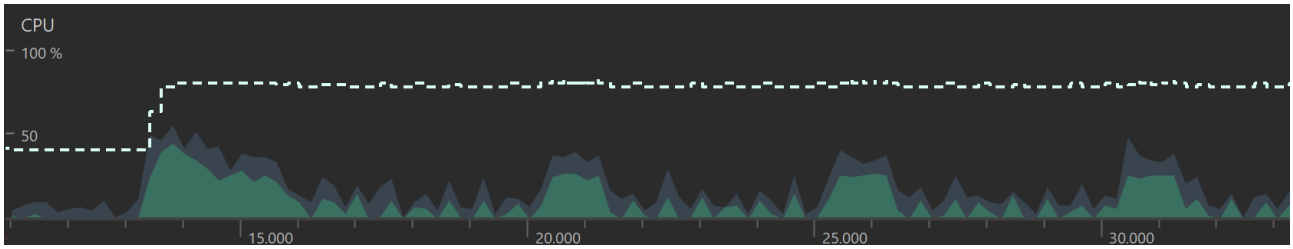

Figura 44 – Xiaomi utilizzo CPU con frequenza di aggiornamento della posizione di 5 secondi

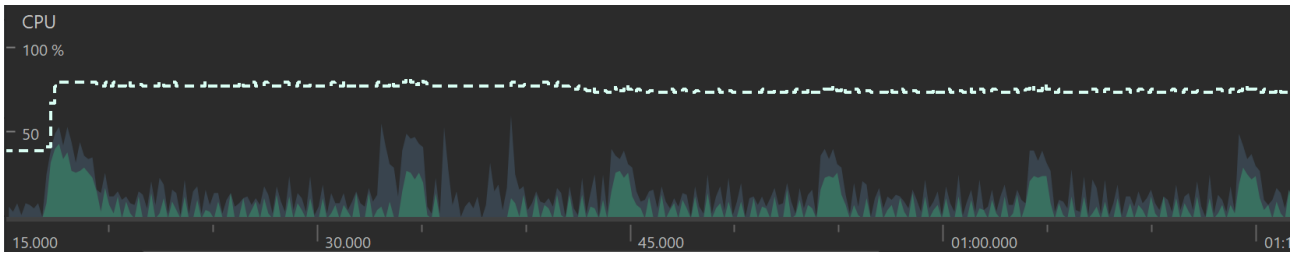

Figura 45 – Xiaomi utilizzo CPU con frequenza di aggiornamento della posizione di 10 secondi

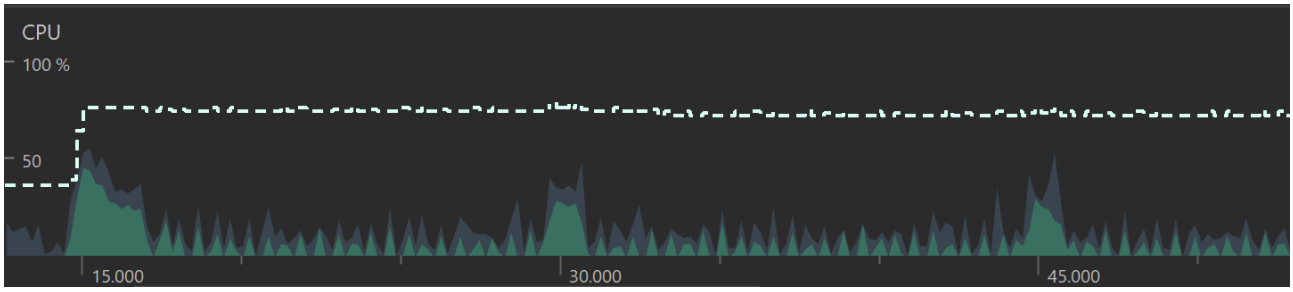

Figura 46 – Xiaomi utilizzo CPU con frequenza di aggiornamento della posizione di 15 secondi

Dalle immagini mostrate è possibile notare come il comportamento in linea di massima risulti simile in tutti e tre i casi : all'inizio della registrazione del percorso è presente un picco di utilizzo di CPU con valore intorno al 45% (prima area verde del grafico), per poi scendere rimanendo fluttuante intorno a valori base e periodicamente avere picchi intorno al 25/30 % di utilizzo di CPU in corrispondenza dei momenti di aggiornamento della posizione, ossia ogni cinque, dieci o quindici secondi in base al caso.

Vengono ora mostrate le misurazioni effettuate per il Samsung.

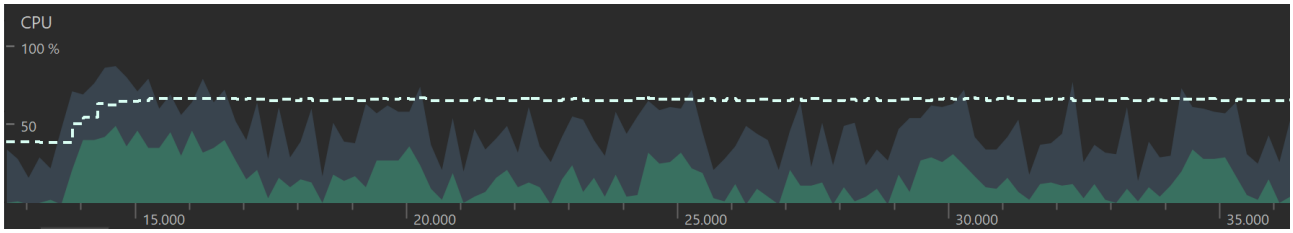

Figura 47 - Samsung utilizzo CPU con frequenza di aggiornamento della posizione di 5 secondi

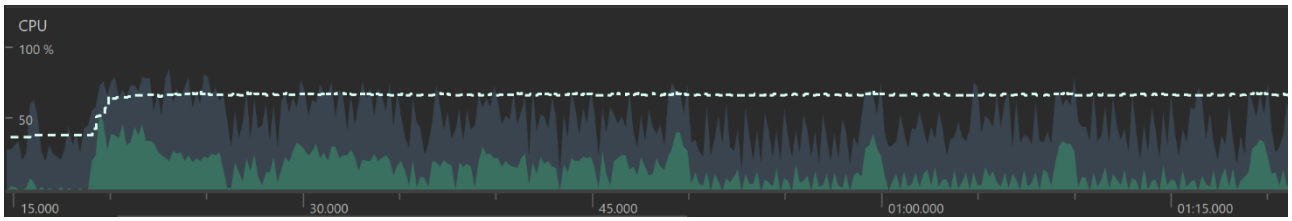

Figura 48 - Samsung utilizzo CPU con frequenza di aggiornamento della posizione di 10 secondi

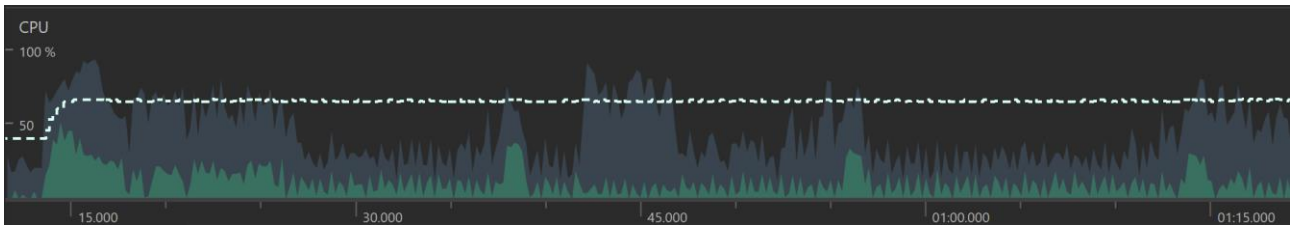

Figura 49 - Samsung utilizzo CPU con frequenza di aggiornamento della posizione di 15 secondi

Ugualmente alle misurazioni effettuate sullo Xiaomi, anche sul Samsung il momento in cui è risultato maggiore l'uso di CPU è stato durante il primo aggiornamento arrivando a picchi, in questo caso, compresi tra il 50% e il 55% nelle diverse misurazioni. Successivamente sono risultati picchi periodici, come si era visto nello Xiaomi, relativamente agli aggiornamenti di posizione, arrivando però a valori attorno al 40%.

In tutte le immagini sono inoltre presenti linee formate da punti bianchi, elementi che indicano il numero corrente di thread che eseguono per l'applicazione, ossia in questo caso sotto-processi in esecuzione concorrente; come è possibile notare in tutte e tre le misurazioni nello Xiaomi il numero di thread è salito fino ad ottanta per stabilizzarsi nell'avanzare del tempo intorno a valori di settanta/settantacinque; mentre per quanto riguarda il Samsung il numero è salito fino a cento e nel tempo è rimasto stabile intorno a questo valore.

Dalle misurazioni appena descritte è possibile notare due diversi aspetti:

- Nel Samsung l'uso di CPU risulta maggiore di circa 10% concludendo che quest'ultimo riserva all'applicazione più potenza di calcolo, in percentuale, rispetto allo Xiaomi. Tale potenza può essere attribuibile al numero maggiore di thread dell'applicazione nel Samsung
- La frequenza di aggiornamento impostata in fase di implementazione risulta rispettata durante l'effettiva esecuzione dell'applicazione in entrambi i dispositivi

4.4.3 Utilizzo di memoria RAM

Durante gli ultimi anni la memoria RAM disponibile nei dispositivi è aumentata considerevolmente, permettendo un'elevata quantità di applicazioni in esecuzione contemporaneamente. Questo però non toglie l'importanza di capire quanto il software sviluppato sia significativo da questo punto di vista, poiché elevate occupazioni di memoria potrebbero rallentare il dispositivo e rendere peggiore l'esperienza utente.

Per questo si è deciso di monitorare l'utilizzo di memoria RAM da parte dell'applicazione sviluppata.

Lo studio è stato effettuato utilizzando il profiler di Android Studio, come nel caso di studio della CPU, il quale permette di visionare in tempo reale l'utilizzo di RAM da parte dell'applicazione in esecuzione.

L'utilizzo di RAM è stato campionato su entrambi i dispositivi in tre diversi momenti:

- Dopo l'avvio dell'applicazione
- Durante un uso "strees" dell'applicazione
- Al termine dell'uso "stress"

Per "stress" si intende un elevato spostamento nella mappa, con operazioni di "zoom in" e "zoom out", poiché sono quelle che più incrementano l'utilizzo di memoria.

Partendo dal Samsung è possibile vedere l'uso di memoria RAM dalle immagini sottostanti.

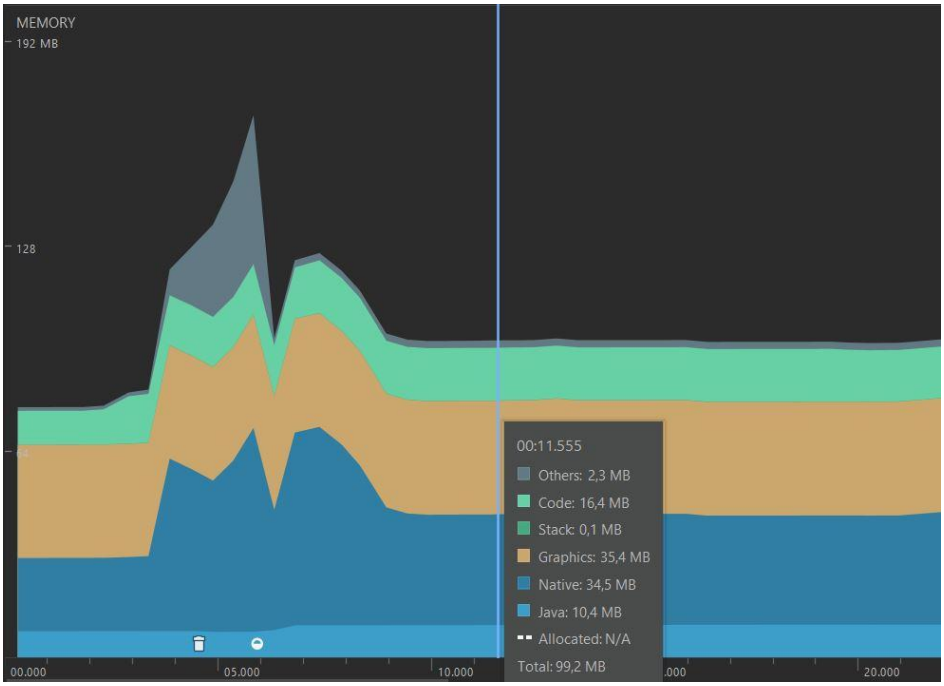

Figura 50 - Samsung utilizzo di RAM all'avvio

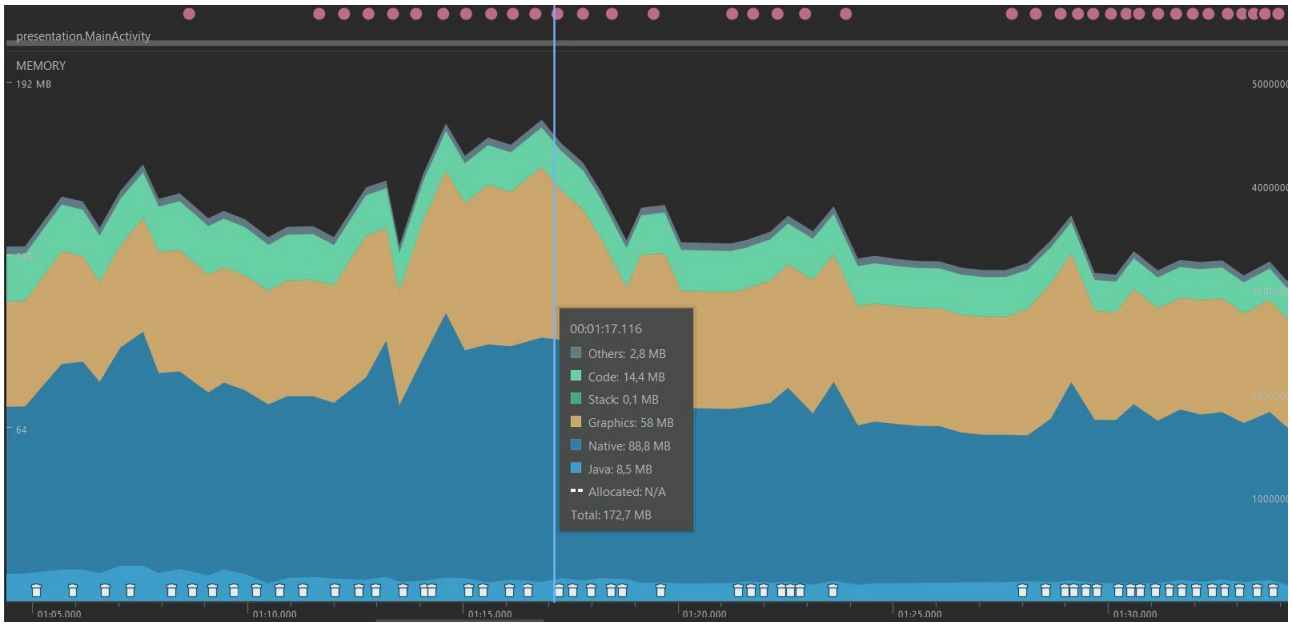

Figura 51 - Samsung utilizzo "stress"

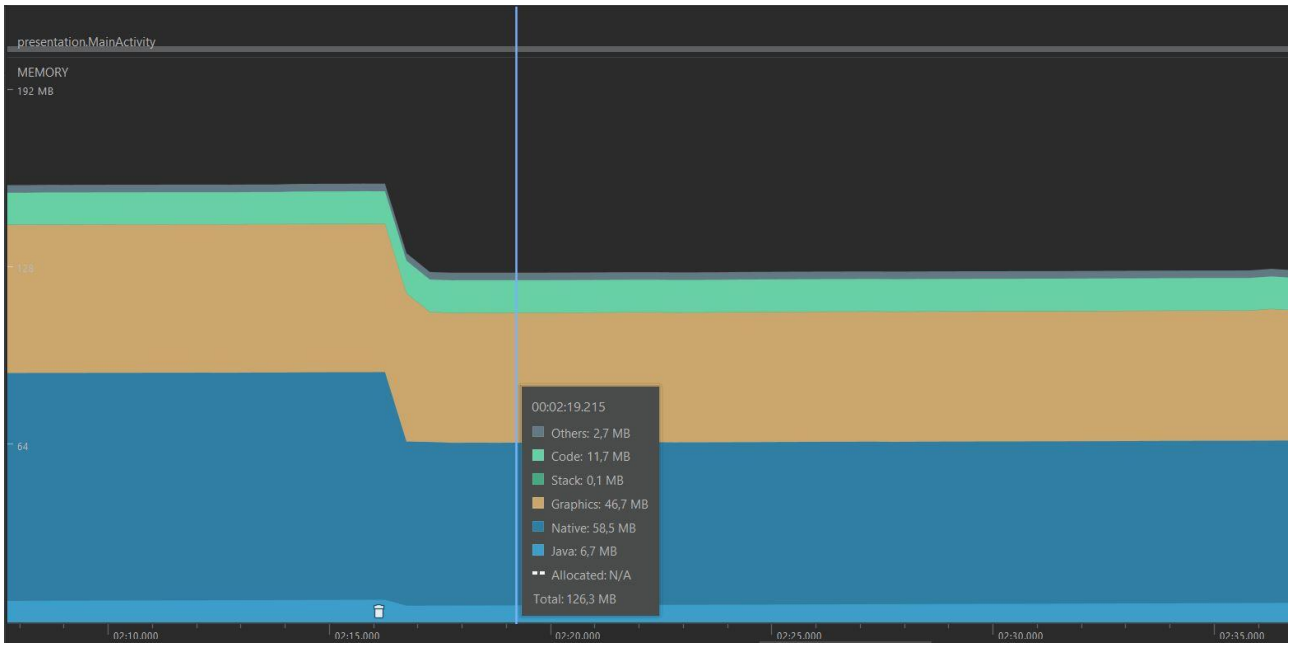

Figura 52 - Samsung termine del periodo di uso "stress"

Nelle immagini sono presenti zone del grafico con colori diversi, i quali rappresentano le diverse categorie di memoria che fanno parte della RAM:

- Java: memoria da oggetti allocati da codice Java o Kotlin.
- Native: memoria da oggetti allocati da codice C o C ++. Anche se non si è sviluppata l'applicazione in codice C/C++, è possibile vedere un po' di memoria nativa utilizzata perché il framework Android utilizza tale memoria per gestire varie attività per conto dell'applicazione.

- Graphic: memoria utilizzata per le code del buffer grafico per visualizzare i pixel sullo schermo.
- Stack: memoria utilizzata dagli stack nativi e Java nella applicazione.
- Code: memoria che l'app usa per il codice e le risorse, come bytecode dex, codice dex ottimizzato o compilato, librerie .so e caratteri.
- Other: memoria utilizzata dall'applicazione che il sistema non è in grado di classificare.
- Allocations: il numero di oggetti Java / Kotlin allocati dall'applicazione.

Analizzando i grafici è possibile notare che all'avvio l'applicazione utilizza all'incirca cento MB di RAM, durante l'utilizzo "stress" si è arrivati ad un picco di circa centottanta MB, per poi iniziare a scendere dopo tale utilizzo. La memoria viene liberata dal Garbage Collector di Android, le cui esecuzioni sono rappresentate dal simbolo di cestino visibile in basso.

Passando allo Xiaomi, i risultati delle misurazioni sono visibili nelle successive immagini.

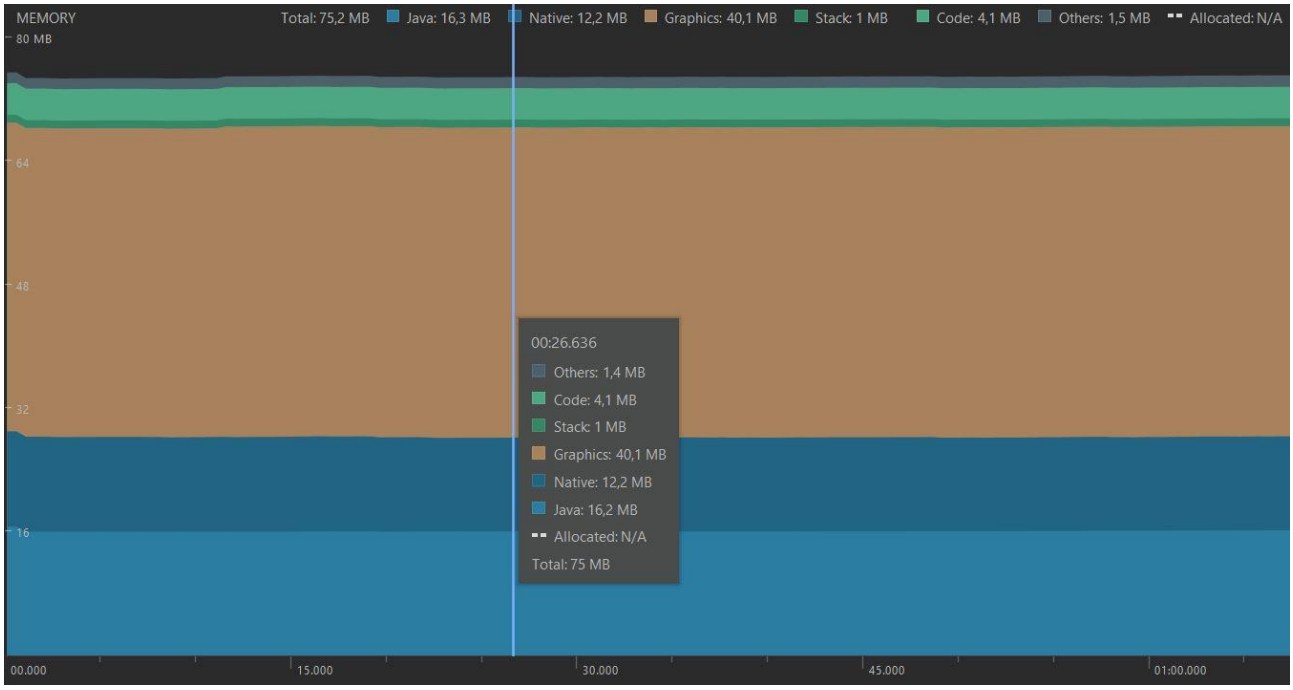

Figura 53 - Xiaomi utilizzo di RAM all'avvio

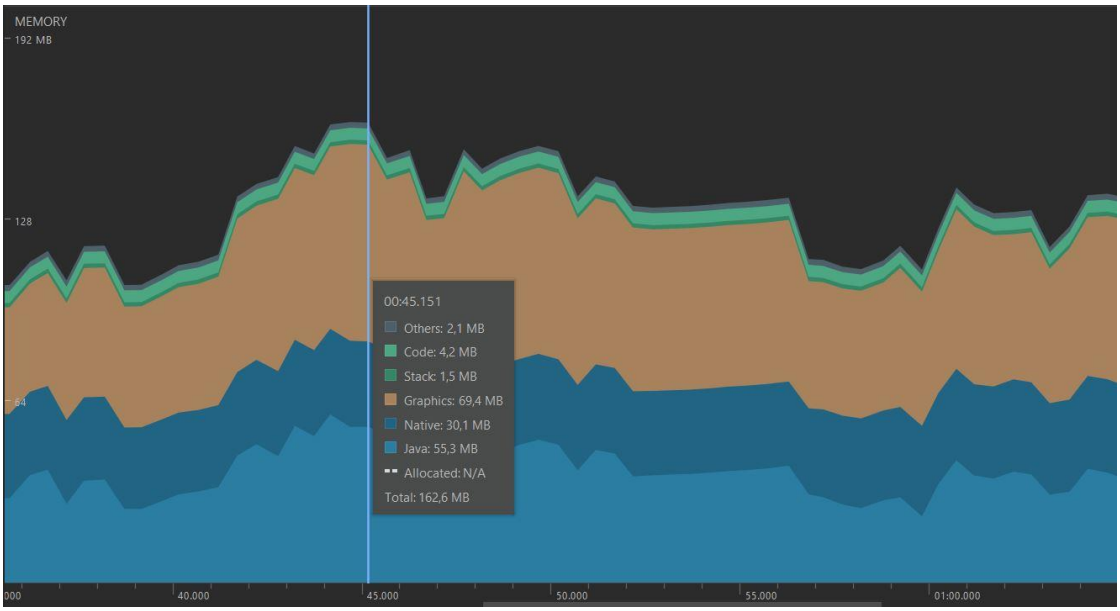

Figura 54 - Xiaomi utilizzo "stress"

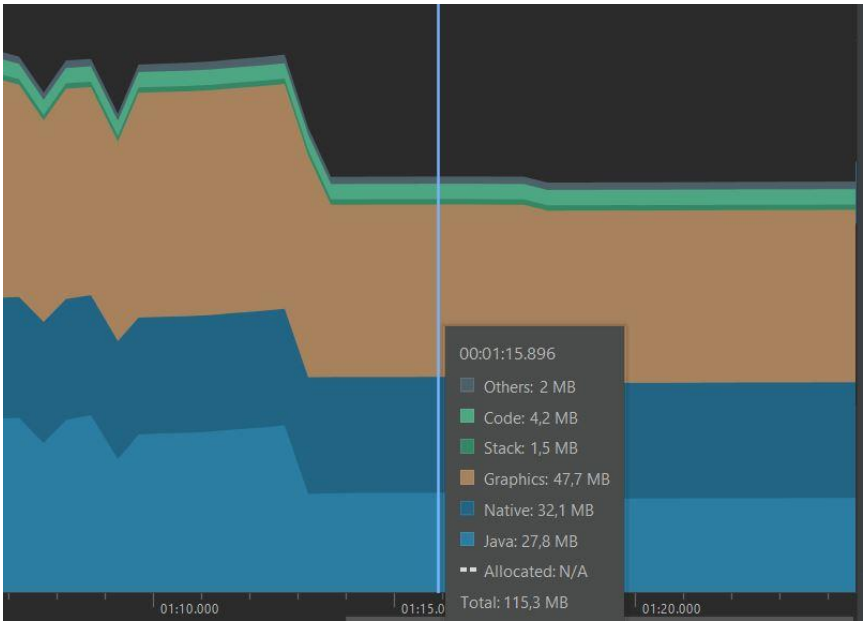

Figura 55 - Xiaomi termine del periodo di uso "stress"

In questo caso analizzando i grafici è possibile notare che all'avvio l'applicazione utilizza all'incirca settantacinque MB di RAM, durante l'utilizzo "stress" si è arrivati ad un picco di circa centosessanta MB, per poi iniziare a scendere dopo tale utilizzo. La memoria viene liberata dal Garbage Collector di Android, le cui esecuzioni non sono visibili nelle immagini perché il dispositivo utilizzato ha una versione di Android (7.1) che non permette ad Android Studio di rilevare tali informazioni.

4.4.4 Consumo di batteria

Come ultimo studio è stato effettuato quello relativo al consumo di batteria. Nello specifico si è monitorato tale consumo relativamente all'operazione di aggiornamento della posizione con una determinita frequenza utilizzando il provider GPS, modificando in modo opportuno la chiamata al metodo "requestLocationUpdate".

Si è escluso dallo studio il dispositivo Xiaomi poiché essendo relativamente datato (utilizzato per circa 3 anni) presenta un'elevato grado di deterioramento della batteria, il quale causa in modo non prevedivile drastici picchi di scaricamento. Il Samsung è risultato accettabile poiché è stato acquistato pochi mesi fa e presenta in generale una scarica lineare della batteria.

Lo studio si è svolto effettuando tre diverse misurazioni della durata di un'ora partendo con la misurazione sempre dalla stessa precentuale di batteria residua e stesse condizioni di funzionamento del dispositivo, ossia con solo la rete mobile e il GPS attivi, stesso livello di luminosità dello schermo e nessun'altra applicazione in esecuzione, ad esclusione dei processi in background non visibili. Le frequenze di aggiornamento utilizzate sono state: un secondo, dieci secondi e venti secondi.

I dati raccolti sono rappresentati nel grafico cartesiano sottostante collegati da una linea continua per evienziarne l'andamento. Nell'asse delle ordinate è rappresentata la percentuale di batteria utilizzata durante la misurazione; nell'asse delle ascisse è rappresentato il numero di aggiornamenti effettuati nell'arco della misurazione, relativi alla la frequenza di aggiornamento impostata: con un secondo si hanno tremilaseicento aggiornamenti in un'ora, con dieci secondi trecentosessanta, con venti secondi centottanta.

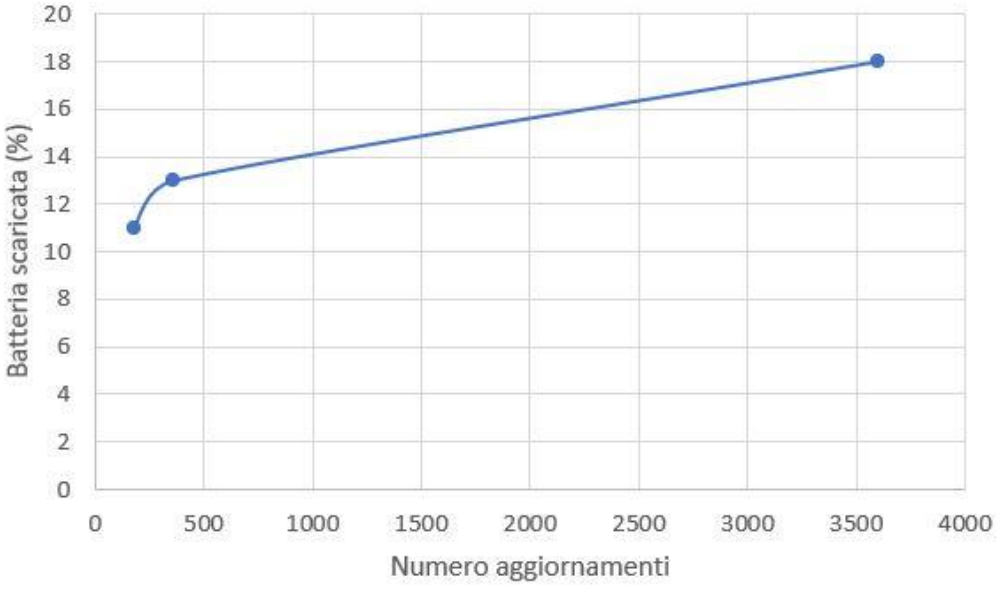

Figura 56 - Grafico relativo al consumo di batteria

Dal grafico è possibile notare come effettivamente la frequenza di aggiornamento influisce sul consumo di batteria dell'applicazione. In particolare, dalla misurazione con frequenza di venti secondi a quella di dieci secondi la percentuale di batteria scaricata è aumentata, approssimando alla prima cifra intera, del 18%; dalla misurazione con frequenza di venti secondi a quella con frequenza di un secondo è aumentata, sempre approssimando alla prima cifra intera, del 64%.

Da questi dati è possibile affermare che il consumo di batteria non aumenta in proporzione all'aumento del numero di aggiornamenti effettuati, ossia all'aumento della frequenza. Anche non aumentando in modo proporzionale, si presenta comunque un aumento non trascurabile. Per tale motivo in fase di progettazione di un'applicazione che offre funzionalità periodiche di aggiornamento della posizione, utilizzando il GPS, è necessario avere particolare riguardo alla frequenza di aggiornamento della posizione per preservare la batteria del dispositivo dell'utente.

Conclusioni

Nel mondo odierno l'utilizzo di applicazioni risulta essere parte costante nella vita quotidiana delle persone. Questo utilizzo si concentra su applicazioni pensate per la comunicazione, ma anche altri ambiti risultano essere significativi. Tra questi è presente l'ambito della localizzazione, che porta con se la possibilità di creare applicazioni per semplificare gli spostamenti e la navigazione.
Probabilmente tutti noi abbiamo usato almeno una volta applicazioni come Google Maps o Waze, che grazie alla possibilità di localizzare il dispositivo, ci hanno aiutato a dirigerci in determinati luoghi o a trovare attività interessanti nei dintorni del luogo in cui eravamo. La possibilità di localizzare un determinato dispositivo in modo sufficientemente accurato permette inoltre di calcolare distanze, tempi e spostamenti di persone o mezzi.

Per questi motivi il lavoro di tesi si è concentrato sulla progettazione e sviluppo di un'applicazione che potesse offrire un servizio di localizzazione e di registrazione di dati relativi ad un percorso effettuato a piedi. In particolare, lo scopo era quello di poter visualizzare su una mappa la propria posizione corrente, tramite l'utilizzo di GPS, reti mobili o WiFi e offrire all'utente la possibilità di avviare la registrazione di un nuovo percorso. Durante la registrazione il sistema sviluppato monitora lo spostamento, mostrando all'utente costantemente la propria posizione e calcolando lo spazio percorso e il dislivello positivo affrontato. Questi dati sono disponibili nell'applicazione in una determinata schermata, in cui sono mostrati tutti i percorsi salvati, con possiblità di eliminarli. Le informazioni salvate in modo persistente e mostrate nella realativa schermata includono: eventuale titolo del percorso, data in cui si è effettuato il giro, distanza percorsa e dislivello affrontato e per ultimi ora di inizio e ora di fine registrazione.
Per offrire un'esperienza utente più completa possibile, in sovrapposizione alla mappa sono stati inseriti diversi elementi: una bussola per potersi orientare, un barra per indicare le distanze reali visualizzabili nella mappa ed una minimappa per visualizzare in ogni momento la propria posizione da una prospettiva meno ravvicinata.

Prima di arrivare allo sviluppo vero e proprio sono stati studiati ed analizzati la piattaforma Android e il linguaggio di programmazione utilizzato per implementare il progetto, ossia Kotlin. Inoltre si è approfondito il progetto OpenStreeMap, legato allla mappa visualizzabile nell'applicazione ed infine sono state effettuate analisi e studi sperimentali sul funzionamento dell'applicazione.

Soffermandosi sulla parte sperimentale, è interessante riassumere il lavoro fatto e i risultati ottenuti. Per effettuare tali studi sono stati utilizzati due dispositivi, un Samsung A51 e uno Xiaomi Mi Max 2. In primo luogo è stata analizzata la latenza, ossia l'intervallo di tempo che intercorre fra il momento in cui viene inviato l'input/segnale al sistema e il momento in cui è disponibile il suo output; l'operazione in questione è la richiesta della propria posizione, effettuabile tramite la pressione di un determinato pulsante sulla schermata principale. Dalle misurazioni è risultato che tra i due dispositivi, utilizzando qualsiasi tipo di localizzazione (GPS, rete mobile o WiFi), il più reattivo è lo Xiaomi, in particolare di circa 1.2/1.3 s rispetto al Samsung. E' stato inoltre riscontrato che il tipo di localizzazione "preferibile" in ambienti interni,come miglior compromesso tra latenza e precisione, risulta essere quello WiFi, ricordando che il GPS non funzioniona in tali ambienti; mentre per quanto riguarda l'esterno, la tipologia di calcolo dipende dalle necessità, considerando però che la localizzazione tramite rete mobile potrebbe essere, in alcuni casi, molto poco accurata.

Passando all'utilizzo di CPU, è stato misurato durante la registrazione di un percorso. Nel Samsung si sono presentati picchi massimi di utilizzo di CPU di circa il 55%, mentre sullo Xiaomi, sempre per le stesse operazioni, picchi massimi di circa 40%; collegata a questo risultato è la misurazione del numero di thread in esecuzione per l'applicazione nei due dispositivi, ossia nell'intorno di cento per il Samsung e nell'intorno di ottanta per lo Xiaomi.

Per la misurazione dell'allocazione di memoria RAM da parte dell'app, sono stati presi campioni in momenti diversi: dopo l'avvio dell'applicazione, durante un uso "stress" dell'applicazione e al termine dell'uso "stress". Dalle analisi è emerso che all'avvio l'applicazione utilizza all'incirca cento MB di RAM nel Samsung e circa settantacinque MB nello Xiaomi, durante l'utilizzo "stress" si è arrivati ad un picco di circa centottanta MB nel Samsung e circa centosessanta MB nello Xiaomi. Dopo l'utilizzo "stress" la memoria ha inizato ad essere liberata dal Garbage Collector di Android.

Per ultimo è stato misurato il consumo di batteria. In questo caso le analisi sono stati effettuate esclusivamente sul Samsung poiché lo Xiaomi presentava un elevato grado di deterioramento della batteria, il quale causava in modo non prevedivile drastici picchi di scaricamento.
Nello specifico si è monitorato tale consumo relativamente all'operazione di aggiornamento della posizione con una determinata frequenza utilizzando il provider GPS. Il cambio di frequenza è stato

effettuato modificando in modo opportuno la chiamata al metodo “requestLocationUpdate”, la cui funzione è proprio quella di ottenere continui aggiornamenti della posizione. E’ stato osservato come effettivamente la frequenza di aggiornamento influisce sul consumo di batteria, aumentando in modo non proporzionale con l’aumento della frequenza. Per tale motivo in fase di progettazione di un’applicazione che offre funzionalità periodiche di aggiornamento della posizione, utilizzando il GPS, è necessario avere particolare riguardo nell’impostazione della frequenza, per preservare la batteria del dispositivo dell’utente.

L’applicazione sviluppata rappresenta una prima versione, i cui servizi offerti risultano perfettamente funzionanti. Grazie all’utilizzo di determinati pattern in fase di progettazione, Layer Pattern e Factory Pattern, gli sviluppi futuri sono facilitati. Le direzioni più promettenti di estensione prevedono la possibilità di scambiare dati tra due dispositivi vicini, migliorare la visualizzazione del percorso sulla mappa ed infine la possibilità di scattare fotografie durante il percorso, con l’indicazione dell’esatta posizione in cui la foto è stata scattata.

Bibliografia

[1] Statista - Mobile operating systems' market share worldwide from January 2012 to July 2020
https://www.statista.com

[2] Wikipedia - Android
https://it.wikipedia.org/wiki/Android

[3] Google - Android Platform
https://developer.android.com/guide/platform

[4] Giuseppe Maggi - Guida Android
https://www.html.it/guide/guida-android/

[5] Google - Android Components
https://developer.android.com/guide/components/fundamentals

[6] Google - Android Manifest file and component
https://developer.android.com/guide/topics/manifest/manifest-intro

[7] Wikipedia – Geolocalizzazione
https://www.wikipedia.org/wiki/Geolocalizzazione

[8] Ivano Tagliapietra – Geolocalizzazione nei dispositivi mobili
http://www.mokabyte.it/2019/03/geolocmobile-1/

[9] Wikipedia – Kotlin
https://it.wikipedia.org/wiki/Kotlin_(linguaggio_di_programmazione)

[10] Ban Markovic – Process of compiling Android app with Java/Kotlin code
https://medium.com/@banmarkovic/process-of-compiling-android-app-with-java-kotlin-code-_27edcfcce616

[11] JetBrains - Kotlin Documentation
https://kotlinlang.org/docs/

[12] Ruggiero Altini – Kotlin: che cos'è e cosa serve sapere ai programmatori Java
https://italiancoders.it/kotlin-cose-cosa-serve-sapere-ai-programmatori-java/

[13] Vipin Yadav – Difference Between Java and Kotlin in Android with examples
https://www.geeksforgeeks.org/difference-between-java-and-kotlin-in-android-with-examples/

[14] Google - LocationManager
https://developer.android.com/reference/android/location/LocationManager

[15] Google - Location
https://developer.android.com/reference/android/location/Location

[16] Mark Richards – Layered Architecture
https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

[17] Wikipedia - OpenStreetMap
https://it.wikipedia.org/wiki/OpenStreetMap

[18] Luca Delucchi, Maurizio Napolitano, Alessio Zanol – Tutorial OpenStreetMap
https://wiki.openstreetmap.org

[19] Wikipedia – Android Strudio
https://it.wikipedia.org/wiki/Android_Studio

[20] Google – Android Studio
https://developer.android.com/studio/intro

[21] OpenStreetMap Community – osmdroid
https://github.com/osmdroid/osmdroid/wiki

[22] Wikipedia - Latenza
https://it.wikipedia.org/wiki/Latenza