ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE
CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

NeverAlone
Applicazione Mobile per la sicurezza personale

Relazione finale in

Mobile Web Design

Relatore

Dott. Mirko Ravaioli

Presentata da

Loris Sbrocca

Sessione III
Anno Accademico 2014/2015

Alla mia Famiglia.

Indice

Introduzione

Fare sport è un'attività divertente e salutare, ancora di più se lo si pratica in buona compagnia. Un amico in carne ed ossa ci tiene compagnia, può supportarci moralmente ma soprattutto può aiutarci in caso di bisogno. Purtroppo infatti non sempre tutto va come previsto e gli inconvenienti sono sempre dietro l'angolo. Cadute, incidenti, malori sono solo alcuni dei pericoli a cui si va incontro durante la normale vita quotidiana, figuriamoci quando si fa sport.

Quante volte però vi siete ritrovati a fare sport in solitudine? Che sia lo sfogo dallo stress e la frenesia della vita quotidiana, oppure l'impossibilità di organizzare un uscita con i propri compagni di sport molto spesso decidiamo o non possiamo fare a meno di uscire ad allenarci soli.

É qui che interviene *NeverAlone*, l'applicazione sviluppata in questa tesi, che consente di non essere mai davvero soli anche quando lo siamo per davvero!

Gioco di parole a parte, *NeverAlone* è un applicazione pensata per dispositivi mobili (smartphone, tablet) che consente di *fare sport in sicurezza* e di avere un occhio puntato sulla nostra attività atletica monitorandola con il solo scopo di proteggerci, supplendo così alla mancanza di un compagno in carne ed ossa. A *NeverAlone* non interessa se tu sei lento o veloce, magro o grasso. A lui interessa solamente che tu non sia in pericolo.

Sfruttando le capacità dei nuovi dispositivi mobili, presenti ormai da anni sul mercato, è possibile, attraverso determinati algoritmi, distinguere una situazione di attività sportiva normale da una anomala. In caso di bisogno è in grado di chiamare aiuto avvertendo un amico fidato che potrà prontamente intervenire.

Nell'ultimo decennio la diffusione delle tecnologie mobili ha raggiunto il suo apice ed il futuro sembra fiorente dal punto di vista delle vendite. Inoltre lo smartphone è diventato ormai un simbolo dell'epoca in cui viviamo, lo usiamo per lavoro, per studiare, per socializzare ed informaci. Lo viviamo da ogni punto di vista ed esso vive con noi in quasi ogni momento della nostra vita. Lo teniamo sempre appresso e

per questo è il mezzo più adatto per fornirci tutte le utility di cui abbiamo bisogno nella vita quotidiana. Inoltre anche se gli smartphone sono sempre più degli oggetti "tuttofare" rimangono comunque il mezzo più efficace per mettersi in contatto con qualcuno.

Questa tesi di progetto nasce inizialmente come "*porting"* su Android di un idea sviluppata per iOS dal mio amico e collega di lavoro Marco Casadio[1] e si è evoluta infine un'interpretazione personale di quest'ultima. Questa versione condivide con l'originale *NeverAlone*[2] l'idea che sta alla base dell'App, il nome, i loghi e le grafiche.

La tesi è stata suddivisa in capitoli cercando di organizzare e raccogliere in maniera ordinata tutte le fasi del progetto.

Capitolo 1 - Stato dell'arte delle tecnologie mobili

In questo capitolo viene fatta una panoramica generale dei dispositivi mobili. Si va dall'analisi dell'attuale stato della loro diffusione, vengono elencate e velocemente descritte le principali piattaforme leader di mercato delle applicazioni mobili: *Google Android*, *Apple iOS*, *Microsoft Windows Phone*. Infine vengono illustrati i diversi *Marketplace*.

Capitolo 2 - Analisi preliminare

In questo capitolo viene descritta tutta la fase preliminare alla realizzazione del progetto, la definizione degli obiettivi da raggiungere e le riflessioni che hanno condizionato le fasi successive. Infine vengono analizzate le applicazioni della concorrenza e le rispettive funzionalità.

Capitolo 3 - Progettazione

In questo capitolo viene illustrata tutta la fase di progettazione che precede l'implementazione. Si parte con la descrizione dell'ambiente di sviluppo fino ad arrivare alla stesura delle linee guida da seguire nella fase successiva.

Capitolo 4 - Implementazione

In questo capitolo viene trattata la vera e propria fase implementativa. Sono state illustrate le fasi salienti dello sviluppo ed eviden-

ziate le scelte tecniche adottate per la risoluzione dei problemi e per l'ottimizzazione delle prestazioni.

Capitolo 5 - Conclusioni e sviluppi futuri

In questo ultimo capitolo vengono illustrati possibili implementazioni future che renderebbero più appetibile l'applicazione nel mercato e la strategia che verrà adottata per favorire la sua diffusione.

CAPITOLO 1

Stato Dell'arte Delle Tecnologie Mobili

In questo capitolo verrà fatta una panoramica sul mondo delle tecnologie mobili. In particolare verrà analizzata la diffusione dei dispositivi e delle infrastrutture a loro supporto, inoltre verranno descritti i *trend* di mercato riguardanti la diffusione dei diversi sistemi operativi e le quote di mercato dei diversi "*Marketplace"* mettendo in evidenzia le tendenze che hanno influenzato le scelte di questo progetto.

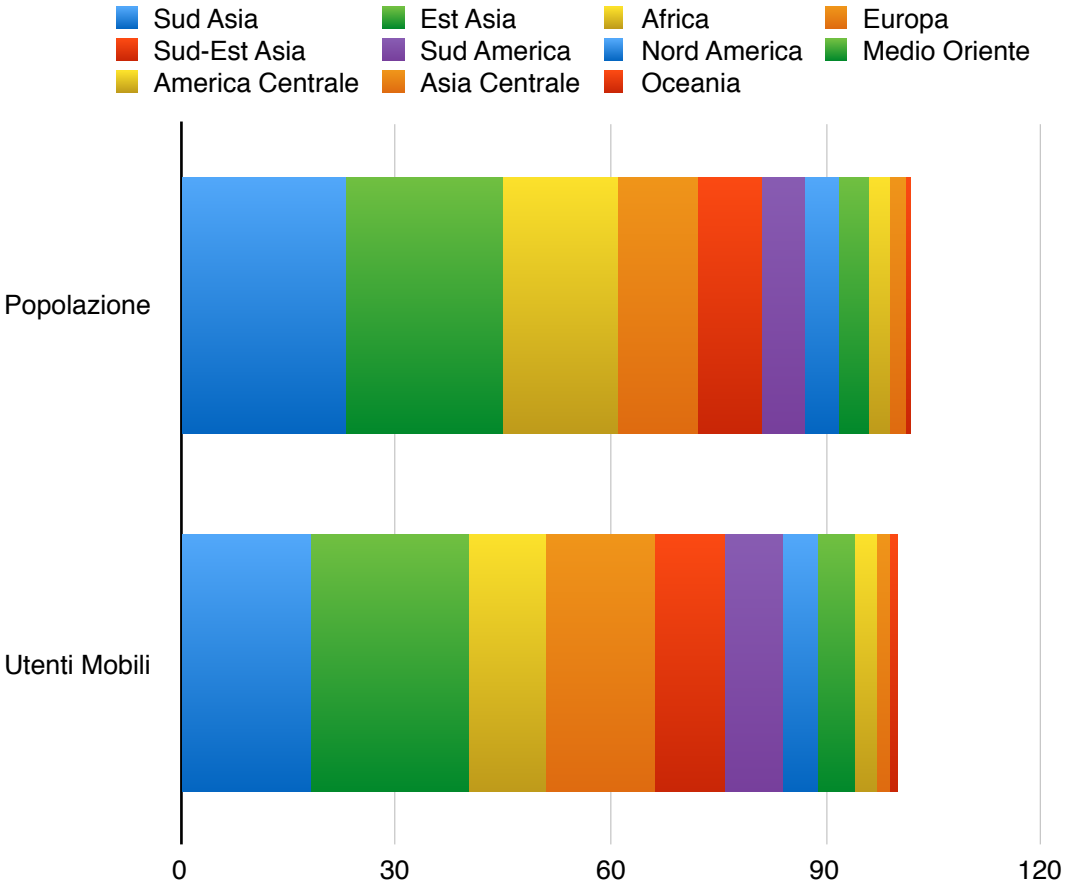
1.1 Diffusione dispositivi mobili

Si stima che i dispositivi mobili (smartphone, laptop, netbook, tablet) attivi nel 2016 saranno circa 10 miliardi superando di 3 miliardi il numero degli abitanti di questo pianeta.[3]
Ciò significa che in media ogni abitante della terra possiederà più di un dispositivo mobile.

Un rapporto di 279 pagine intitolato *Social, Digital & Mobile in Europa 2014*[4] di *We Are Social* datato 2015 ha raggruppato i dati provenienti da diverse fonti (*US Census Bureau, InternetWorldStats, CNNIC, IAMAL Tencent, Facebook, VKantakte, ITU, CIA*) e ha stilato un report dettagliato sulla diffusione dei dispositivi mobili nel Mondo con un approfondimento particolare per l'Europa. Secondo il report la popolazione mondiale conta poco più di 7 miliardi di persone e attualmente sono attivi nel mondo 6,5 miliardi di abbonamenti mobili, 1 miliardo nella sola Europa. La percentuale di penetrazione è del 93%. Questo dato è ovviamente non significativo se si vuole conoscere esattamente il numero di persone raggiunte da questa tecnologia, ma può essere rilevante per intuire che l'infrastruttura in termini di carico è in grado, almeno nella teoria, di sopportare una copertura effettiva totale.
Un'analisi geografica più dettagliata denota come le percentuali di penetrazione degli abbonamenti mobili all'interno delle macro-aree geografiche abbiano raggiunto saturazione fatta eccezione per Africa e

Sud Asiatico. Questi dati evidenziano come la diffusione dei dispositivi mobili è destinata a raggiungere la saturazione nell'arco di pochi anni nelle aree del primo mondo.

In Italia in particolare la percentuale di penetrazione degli abbonamenti mobili attivi è particolarmente alta e si attesta attorno al 158% su una media europea del 139%.



Una ricerca a cura dell'*Università di Firenze* chiamata *Teenagers su Internet*[5] afferma che il 31% dei ragazzi di 10-11 anni possiede uno smartphone che diventano il 58% l'anno immediatamente successivo e il 72% nella fascia d'età 12-13.

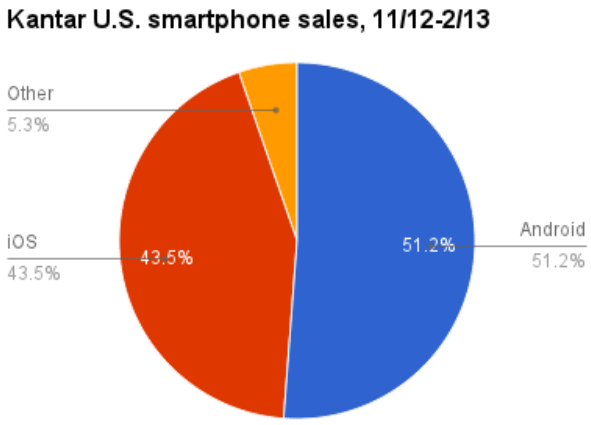
Questo dato è molto rilevante in quanto significa che l'eta minima di accesso a queste tecnologie è bassa e con tutta probabilità tenderà ad abbassarsi sempre di più. Parte del bacino di utenza che potrebbe interessare l'App di questo progetto possono essere proprio i giovanissimi che incalzati dai genitori installeranno un'applicazione per la sicurezza personale come può essere appunto *NeverAlone*.

In generale comunque la diffusione globale dei dispositivi mobili evidenziata in questo paragrafo è la prova che investire sul mercato delle App è oggi come non mai un'opportunità di guadagno.

1.2 Sistemi operativi mobile

Nel corso degli anni 2000 sono stati immessi sul mercato diversi sistemi operativi pensati per essere eseguiti appositamente sui dispositivi mobili. I più famosi tra questi sono *Symbian OS*, *Android OS*, *iOS*, *Windows Phone*, *BlackBerry OS*.

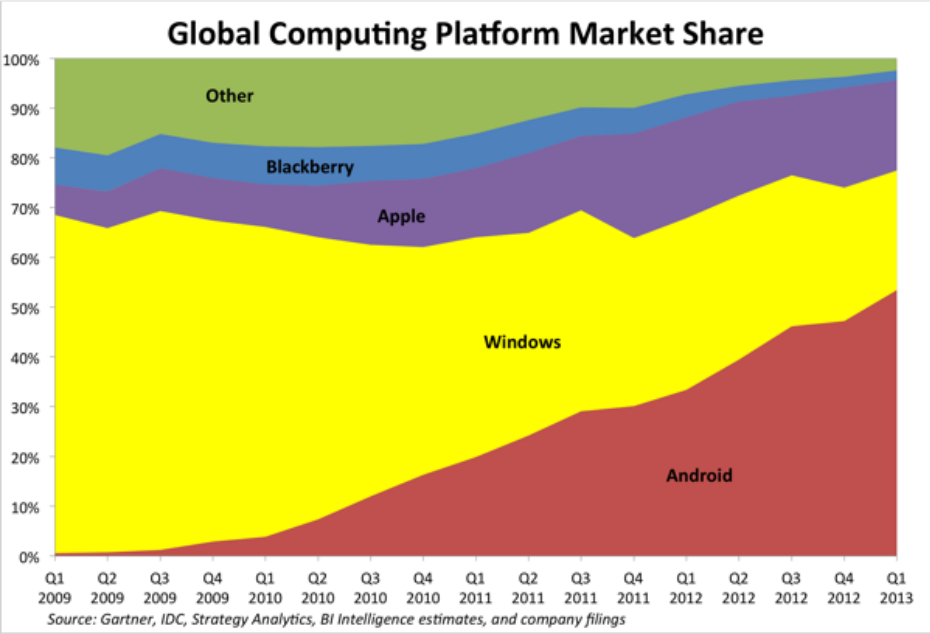
Analizzando le tendenze di mercato possiamo vedere come attualmente il sistema operativo con più quote di mercato è Android OS con cifre che si attestano a livello europeo intorno al 60% (10 punti percentuali in più rispetto al mercato statunitense) e iOS in europa mediamente al 25% (10 punti percentuali in meno rispetto al mercato statunitense). Windows Phone cresce un po' in tutto il mondo ma in misura nettamente minore rispetto ad Android e iOS e le sue quote si attestano intorno al 7%. Le quotazioni dei restanti sistemi operativi quali Symbian, RIM, Bada appaiono decisamente in declino e sono addirittura azzerate in mercati come quello statunitense[6] rendendo lo sviluppo di applicazioni su queste piattaforme decisamente poco appetibile.



Un altro importante indicatore qualitativo che descrive il mercato sono il numero di applicazioni rilasciate all'interno dei diversi *Marketplace*.

I *Marketplace* sono i luoghi digitali all'interno del quale avviene la scelta e l'acquisto delle applicazioni. Sono in pratica anch'esse delle applicazioni di tipo eCommerce completamente integrate all'interno del sistema operativo in uso.

Dal grafico si evince come il marketplace di Google sia quello più grande e con un fattore di crescita maggiore.



In questo paragrafo emerge come per avere un bacino di utenza elevato, senza il quale sarebbe impossibile immaginare margini di guadagno, è necessario essere presenti su almeno uno dei marketplace di punta. Per questo motivo la scelta è prevedibilmente ricaduta su iOS ed Android.

NeverAlone come già detto è stato sviluppato per due piattaforme, ed anche se non condividono la progettazione e l'implementazione comunque sono accomunate dai requisiti.

1.2.1 Android

Android è un sistema operativo per dispositivi mobili sviluppato da *Google Inc*.

Android è un progetto *Open Source* guidato da Google che ha come obiettivo quello di creare un vero e proprio successo, in modo da migliorare l'esperienza mobile per gli utenti.

La presentazione ufficiale di Android avvenne il 5 novembre 2007 dalla neonata OHA (*Open Handset Alliance*), un consorzio di aziende del settore Hi Tech che include Google, produttori di smartphone come HTC e Samsung, operatori di telefonia mobile come Sprint Nextel e T-Mobile, e produttori di microprocessori come Qualcomm e Texas Instruments Incorporated.[7]

L'appartenenza al consorzio ha rapidamente permesso di equipaggiare uno svariato numero di dispositivi con il sistema operativo di Google, questo ha favorito la sua diffusione a livello globale. La nota dolente riguarda però la frammentazione di versioni che si è andata a creare nel corso degli anni. Infatti il rilascio delle versioni compatibili del sistema di Google è demandata ai singoli produttori del dispositivi che seguendo le regole di mercato o per motivi tecnologici non sempre sono in grado di rilasciare aggiornamenti tempestivi per i propri dispositivi. Questo si traduce in uno svariato numero di versioni della stessa piattaforma ancora in circolazione che complica lo sviluppo di applicazioni compatibili con l'intera gamma di versioni.[8]

Nella tabella seguente sono elencate le versioni di Android attualmente in circolazione con percentuali di distribuzione significative.

Versione	Nome in codice	API	Distribuzione
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	3.4%
4.1.x	Jelly Bean	16	11.4%
4.2.x	Jelly Bean	17	14.5%
4.3	Jelly Bean	18	4.3%
4.4	KitKat	19	38.9%
5.0	Lollipop	21	15.6%
5.1	Lollipop	22	7.9%
6.0	Marshmallow	23	Non disponibile

Di recente Google ha presentato **Android Studio,** il suo ambiente di sviluppo integrato per sviluppare applicazioni per la piattaforma Android.

Google Play raccoglie tutte le applicazioni verificate da Google installabili su dispositivi Android. Nonostante il primato per numero di applicazioni presenti nello store, l'introito delle vendite delle applicazioni vale all'incirca la metà rispetto il mercato Apple.

Essendo la piattaforma su cui si è sviluppata l'applicazione di questa tesi è importante notare come la frammentazione delle versioni porterà ad una difficolta maggiore nella finale fase di *fine tuning* e durante la fase di *debugging e testing.*

1.2.2 iOS

iOS (precedentemente iPhone OS) è un sistema operativo sviluppato da Apple per iPhone, iPod touch e iPad.

Il sistema operativo è stato presentato il 9 gennaio 2007 al Macworld Conference & Expo di San Francisco, e la versione 1.0, ancora priva di nome, è entrata in commercio con il primo iPhone il 29 giugno dello stesso anno.

Apple progetta sia il software che l'hardware, questo ha permesso di mantenere sotto controllo la frammentazione delle versioni della piattaforma.

Versione	Distribuzione
5.x	1.3%
6.x	2.7%
7.x	15.0%
8.x	39.8%
9.x	41.0%

Apple fornisce ai propri clienti sviluppatori un ambiente di sviluppo integrato chiamato **xCode.**

AppStore raccoglie tutte le applicazioni verificate da Apple installabili su dispositivi iOS. Come precedentemente detto, nonostante il primato per numero di applicazioni presenti nello store Android, l'introito delle vendite delle applicazioni vendute sull'Apple Store vale all'incirca il doppio rispetto il mercato Google.

1.2.3 Windows Phone

Windows Phone (spesso abbreviato in WP) è una famiglia di sistemi operativi per smartphone di Microsoft, presentata per la prima volta al Mobile World Congress il 15 febbraio 2010. Si rivolge al mercato consumer a differenza del suo predecessore (Windows Mobile) che era rivolto al mercato Enterprise.

Windows Phone 8.1 è partner OEM assieme a 17 aziende partner come Samsung, HTC, Sony, LG.

Nonostante ciò l'ingresso sul mercato ritardato rispetto i suoi principali concorrenti ha condizionato moltissimo la diffusione della piattaforma che procede molto lentamente. Questa scarsa diffusione ha condizionato anche gli sviluppatori che hanno preferito specializzarsi sulle piattaforme concorrenti.

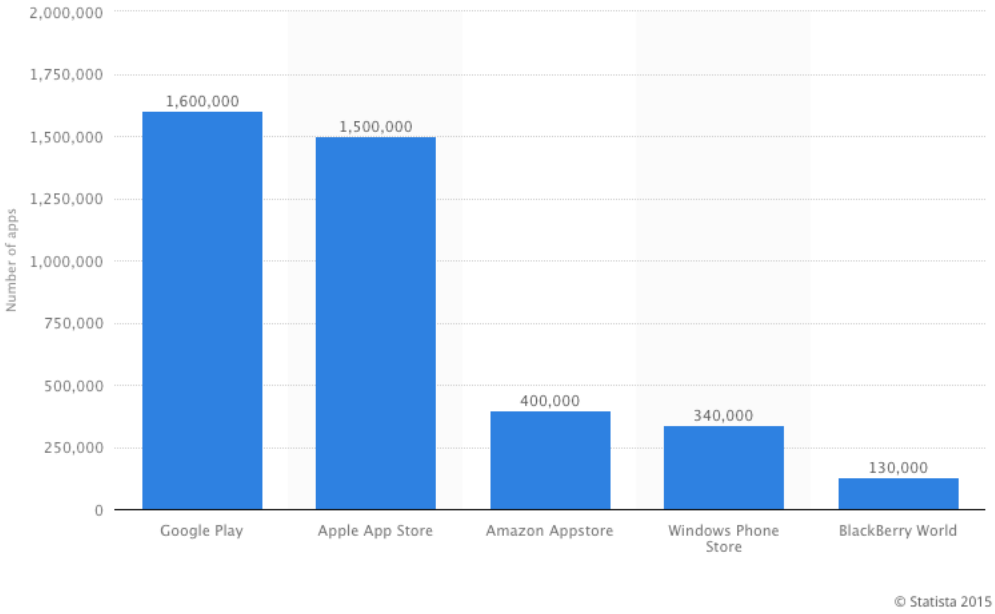
Il vantaggio principale che vanta Windows Phone è la compatibilità massima con i terminali Desktop Windows e soprattutto la possibilità di usare un ambiente di sviluppo avanzato e con anni di esperienza come Visual Studio.

Windows Phone Store raccoglie tutte le applicazioni verificate da Microsoft ed instancabili sui dispositivi dotati di Windows Phone. Questo store essendo arrivato in ritardo rispetto agli altri concorrenti contiene un numero notevolmente minore di applicazioni. Questo è anche uno dei maggiori problemi rilevati dagli utenti e che spesso condiziona le vendite di questa piattaforma.

1.3 Il mercato delle Applicazioni

Come abbiamo visto per ogni piattaforma esiste un negozio digitale dove poter acquistare applicazioni in maniera semplice e sicura. Infatti questi negozi garantiscono la bontà dei prodotti proteggendo l'utente finale da applicativi esplicitamente fraudolente.

Attualmente gli store di riferimento per numero sono l'*Apple Store* e *Google Play* senza però dimenticare l'esistenza di store con un mercato più di nicchia come *Amazon Appstore*, *Windows Phone Store* e *BlackBerry World*.[9]



Il mercato delle Applicazioni è, come prevedibile, direttamente collegato alle vendite di dispositivi mobili. Non sempre però ad un successo nelle vendite di una specifica piattaforma corrisponde un ugual successo in termini di quote di mercato nel mondo delle applicazioni digitali. Lo stesso si può dire del rapporto tra App presenti nel marketplace e introito delle App vendute. Il caso più eclatante è quello che mette a confronto i dati dei rispettivi store di Apple e Google. Nel 2015 le applicazioni pubblicate su Google Play hanno superato in numero quelle presenti sull'Apple Store, nonostante ciò però il mercato Apple vale circa il doppio di quello Google.

Le motivazioni sono da ricercare nelle disponibilità economiche di chi sceglie l'una o l'altra piattaforma:

- Le quote di mercato di Google sono più forti nei paesi emergenti che non sono ricchi.
- Google è stata lenta nell'offrire a questi paesi un processo di fatturazione alternativo alle carte di credito, perdendo così quote di mercato.
- Mediamente un dispositivo Android costa dai 250$ ai 300$ contro una spesa media di 600$ per un iPhone. Per cui un utente che sceglie iPhone è anche più propenso a spendere ulteriore denaro per acquistare applicazioni.

- Questo atteggiamento da parte degli utenti Android condiziona il mercato stesso all'interno del Marketplace danneggiandolo. Infatti la scarsa propensione degli utenti ad acquistare costringe gli sviluppatori ad immettere sul mercato applicazioni gratuite o comunque molto economiche che spesso non soddisfano requisiti qualitativi e funzionali che l'utente medio si aspetta, facendoli così dirottare su altre piattaforme, in primis Apple. Spostando così le quote di mercato della fascia di popolazione più ricca (quella di maggior rilevanza) su iOS.

Da questo punto di vista quindi Apple vince su Google, però non bisogna dimenticare che il mercato delle App, in particolare quello di Android, è in particolare fermento tanto che sono nati marketplace diversi da quello di Google (come per esempio *Amazon Appstore*) che hanno quote di mercato non più trascurabili e che rendono i sistemi Android ancora più appetibili all'utente finale. Questo ulteriore potenziale al lungo e medio termine si prevede possa favorire ancora di più la diffusione del sistema operativo di Mountain View.

In questo paragrafo si consolida l'idea che la presenza su Google Play ed Apple Store è attualmente la scelta vincente per avere margini di guadagno ed opportunità elevate.

CAPITOLO 2
Analisi Preliminare

In questo capitolo viene illustrata la fase preliminare del progetto e le idee che hanno portato alla sua realizzazione. Si è tentato di raccogliere tutte le riflessioni che hanno condizionato le fasi successive. Viene fatta una analisi dei requisiti, un analisi di mercato, un benchmarking ed un brainstorming.

2.1 Analisi dei Requisiti

Con l'Analisi di Requisiti si vogliono prima di tutto stabilire quali saranno le funzionalità che il prodotto dovrà offrire, ovvero i requisiti che devono essere soddisfatti una volta sviluppato il prodotto.

L'idea di base che ha portato alla realizzazione di questo progetto è stata la necessita di garantire una maggior sicurezza nelle uscite in bicicletta del mio collega di lavoro Marco. Lui pratica *Downhill*, una disciplina di tipo *gravity,* in cui le cadute possono essere molto pericolose. Lo stesso meccanismo di sicurezza può essere applicato a svariate discipline come corsa, escursionismo, bicicletta, automobilismo, volo libero ecc…

L'applicazione che è stata realizzata avrà il compito principale di monitorare l'attività del suo utilizzatore con lo scopo di captare situazioni di pericolo come malori o incidenti e, in maniera automatica, di lanciare un messaggio di aiuto ad una persona di fiducia che potrà allertare i soccorsi o intervenire.

Inoltre dovrà essere di supporto al suo utilizzatore quando, trovandosi in una situazione di pericolo, avrà la necessità con pochi tap di spedire un messaggio di aiuto e inviare la propria posizione ad una persona di fiducia.

Il tutto deve essere raccolto in un'interfaccia usabile essenziale e facilmente configurabile.

Requisito necessario è che i consumi energetici siano tenuti bassi.

2.2 Analisi di Mercato

Con l'Analisi di Mercato si vogliono determinare le caratteristiche del bacino di utenza dell'applicazione.

Lo scenario in cui andrà ad inserirsi questa applicazione è strettamente legata alla categoria *Salute e Fitness* (categoria a cui fanno parte applicazioni come *Runtastic* ed *Endomondo*) e la categoria *Strumenti* dove trovano spazio moltissime applicazioni di utilità varia.

Per determinare il **target** di utilizzatori finali ho identificato 3 *Personas*.

Un primo esempio è rappresentato da colui che avrà probabilmente già scaricato un'applicazione che monitori il proprio allenamento ma avrà un occhio di riguardo alla salute ed alla sicurezza personale, integrando così una funzionalità mancante nelle loro applicazioni di supporto all'allenamento. La *personas* di questa situazione è un tipo atletico, salutista che a causa di una fitta agenda organizza i suoi allenamenti all'ultimo momento.

Un altro target di utilizzatori è identificabile da chi si ritrova spesso solo a percorrere tragitti giornalieri in solitudine e vuole un meccanismo o semplicemente una scorciatoia per richiedere aiuto al bisogno. Una *personas* in questa situazione è identificato dalla ragazza che deve raggiungere casa al ritorno da scuola o lavoro.

Un importante fenomeno socio-culturale che potrebbe favorire l'interesse verso questo tipologia di applicazioni è la sempre più bassa età minima degli utenti finali. Come già detto nel primo capitolo il primo cellulare arriva sempre di più in età precoce. Già a 11-12 anni oltre la metà dei ragazzi possiede il suo primo smartphone. Una delle tipologie di applicazioni che i genitori vorranno installare sui dispositivi dei figli sarà proprio un applicazione legata alla sicurezza come questa. In questo caso la tipologia di *personas* è un adolescente che esce di casa con i propri amici e che ha dei genitori apprensivi.

L'obiettivo che si pone questo progetto è di offrire al mercato uno strumento economico ma che sia in grado di soddisfare un grande numero di utenti. Non sarà adatto quindi a chi si aspetta una sicurezza totale e un team di supporto dedicato (come vedremo nel successivo paragrafo alcune delle applicazioni di questa categoria prevedono

canoni di abbonamento a Team di professionisti). Difficilmente chi è disposto a spendere centinaia di euro all'anno per questi servizi si accontenterà di *NeverAlone*. Al contrario il target di riferimento saranno quelle persone che sono disposte a spendere pochi euro per un'applicazione e tutti quelli che non hanno trovato nelle applicazioni gratuite la qualità desiderata.

2.3 Benchmarking

In questo paragrafo si procede sistematicamente con il confronto tra l'applicazione sviluppata in questo progetto e le soluzioni simili disponibili attualmente sul mercato. Bisogna innanzitutto dire che l'idea di base di questo progetto sembra valida in primis perché non sembra esserci in commercio nulla che faccia esattamente quello che promette di fare questa applicazione. Alcune delle applicazioni prese in considerazione sono manuali e meno sofisticate, altre invece si appoggiano a servizi esterni, formati in genere da Team di supporto, e prevedono la sottoscrizione di un canone mensile o annuale che di solito è piuttosto costoso.

2.3.1 Competitor su Google Play Store e App Store

Su **Google Play Store** e su **Apple App Store** sono attualmente pubblicate alcune applicazioni simili. Qui di seguito vengono elencate le più significative in termini di funzionalità e numero di download[10]:

RunSafe GPS Fitness Safety[11] è un applicazione presente su Play Store e su App Store, dedicata a chi pratica sport. Permette di monitorare l'attività fisica, tenere traccia del percorso fatto e ha delle funzionalità sociali di condivisione attività. Dispone di un bottone antipanico che al bisogno può essere utilizzato per condividere la propria posizione alla comunità (formata da coloro che hanno installata l'applicazione. E' presente una versione gratuita ed una a pagamento con delle funzionalità aggiuntive al costo di 4,99 $ al mese che comprende la possibilità di configurare alert automatici e configurare più di un contatto di fiducia.
Pro: funzionalità sociali per la condivisione dell'allenamento, statis-

tiche avanzate allenamento, design semplice ed accattivante
Contro: Costoso ($ 4,99 al mese)

bSafe[12] è un applicazione presente su Play Store e su App Store, che si pone come obiettivo la creazione di una rete di sicurezza personale. Permette quindi di aggiungere alla propria rete amici e familiari per condividere con loro la posizione e per permettere loro di monitorare il proprio itinerario se necessario. Inoltre può monitorare i tuoi movimenti durante l'attività sportiva così da allertare la tua rete nel caso in cui non vengano rilevati movimenti per un tempo troppo lungo. L'applicazione è completamente gratuita, ma è possibile sottoscrivere un abbonamento premium che garantisce il supporto 24/7 di professionisti in caso di bisogno.
Pro: funzionalità sociali per la condivisione della posizione e degli itinerari, design semplice ed accattivante, possibilità di sottoscrivere un abbonamento per il collegamento con il centro operativo.
Contro: nessuno

Siamo Sicure[13] è un applicazione dedicata alle donne, è presente su Play Store e su App Store, permette in caso di necessità di attirare l'attenzione dei passanti attraverso avvisi acustici e attraverso l'utilizzo del flash. Permette inoltre di effettuare rapidamente una chiamata di emergenza e di inviare un SMS con la posizione.
Pro: gratuita, design intuitivo e ricercato
Contro: per l'invio della notifica via SMS necessita di una conferma manuale (su iOS)

Panic Button Red è un'applicazione presente sul Play Store e su App Store, formata essenzialmente da una schermata principale con il bottone "panic" e da qualche schermata di configurazione. Mette a disposizione dell'utente un widget di tipo bottone da posizionare in una delle schermate del telefono che può essere cliccato al bisogno. Nella versione gratuita il messaggio di aiuto può essere inviato via SMS.
Esiste poi una versione a pagamento del costo di 6,64€ che aggiunge diverse caratteristiche avanzate: rimozione banner pubblicitari, aggiun-

ta indirizzo all'SMS, possibilità di specificare più indirizzi SMS, possibilità di registrare una chiamata di emergenza, possibilità di inviare il messaggio via e-mail, accesso da remoto alla posizione del dispositivo, condivisione sui social del messaggio di aiuto.
Ha ottenuto fino ad ora oltre 50.000 download ed una valutazione media di 3,7/5 su un campione di 665 opinioni.
Pro: grande varietà di canali per l'invio delle emergenze, presenza widget.
Contro: Pubblicità, design e User Experience obsoleti.

SOS - Stay Safe! è formata essenzialmente dalla schermata principale con il bottone "alert!" e da alcune schermate di configurazione. Può inviare messaggi di aiuto via SMS o e-mail. E' disponibile in versione gratuita e in versione a pagamento. Nella versione gratuita è possibile aggiungere al massimo 2 numeri di telefono e 2 indirizzi e-mail. La versione a pagamento costa 0,99€ e questa limitazione non c'è, inoltre vengono abilitati i messaggi d'aiuto vocali (via e-mail) e non è presente la pubblicità. La caratteristica che rende interessante questa applicazione è la possibilità di inviare il messaggio di aiuto tramite lo shake del dispositivo.
Ha ottenuto fino ad ora oltre 10.000 download ed una valutazione media di 2,8/5 su un campione di 280 opinioni.
Pro: possibilità di usare lo shake per inviare l'alert, costo versione completa basso.
Contro: Pubblicità, design e User Experience migliorabili.

ProTechTor[14]*,* è un applicazione presente su tutti i maggiori marketplace (Google, Apple, Windows) con caratteristiche avanzate di supporto in caso di necessità. L'applicazione per funzionare necessita della sottoscrizione di un abbonamento annuale del costo di 182,50€ e permette di comunicare direttamentee con un centro operativo dedicato a cui è possibile inoltrare, attraverso l'App, diverse tipologie di richieste di aiuto: SOS, HELP, ALERT, TRACKING
Pro: applicazione di design, Centro di Controllo dedicato con professionisti.
Contro: Costoso (abbonamento annuale 182,50€)

Panik[15], è una semplice applicazione che permette in caso di bisogno di attirare l'attenzione dei passanti attraverso l'utilizzo del flash e di un rumoroso allarme. Può mandare richieste di aiuto sui social network o mediante SMS. Infine ha la particolarità di poter essere impostata in modo che si attivi dopo tramite lo shake del dispositivo
Pro: gratuito, design piacevole ed accattivante, intuitivo
Contro: per l'invio della notifica via SMS necessita di una conferma manuale

2.3.2 Altre soluzioni al di fuori dei Marketplace

Di recente *Garmin* ha presentato una serie di dispositivi dedicati ai cicloturisti. Viene fatto riferimento ai device *Garmin Edge*[16]. Questi dispositivi nascono essenzialmente come navigatori GPS che possono essere installati sul manubrio di una bicicletta. Hanno caratteristiche di impermeabilità e resistenza agli urti. Hanno pre-caricate mappe, sentieri e punti di interesse. Permettono di tenere traccia del percorso fatto e come ulteriore caratteristica permettono di rilevare urti e cadute per mezzo di un accelerometro interno e associandolo ad uno smartphone possono inviare notifiche con messaggi di aiuto.
Pro: Hardware completamente dedicato, funzionalità avanzate di traking e navigazione
Contro: Costoso (prezzi a partire da 449€), necessità di usare uno smartphone per inviare i messaggi di aiuto.

2.4 Brainstorming

Per competere con gli altri prodotti presenti sul mercato bisogna offrire un prodotto con determinate caratteristiche. Per mezzo del seguente *word cloud* si è tentato di dare una visione di massima, attraverso delle parole chiave, sulle caratteristiche che dovrà avere il progetto.

2.5 Card Sorting

Il *card sorting* per un applicazione di questo tipo è generalmente molto piccola vista la scarsità di contenuti da presentare, però comunque sia vale la pena di sottoporre il progetto a questo tipo di analisi.

I contenuti che dovranno poter essere visualizzati dall'applicazione si possono suddividere in 4 categorie velocemente consultabili:

- *MONITOR*: contiene la *call-to-action* principale per avviare il monitoraggio dell'attività. Mostrerà inoltre altre informazioni utili come per esempio il timer di avvio del servizio, eventuali situazioni di pericolo rilevate;
- *MAPPA*: contiene una mappa visuale con la posizione corrente, sarà utile in caso di smarrimento. Tramite questa mappa l'utente potrà capire esattamente dove si trova.

- *POSIZIONE*: mostra le coordinate geografiche (latitudine, longitudine e Altitudine) della posizione corrente, può essere comodo nel caso si voglia comunicare in altro modo la propria posizione, per esempio dettandole a voce.
- *IMPOSTAZIONI*: Ci sarà infine una schermata di configurazione raggiungibile per mezzo del menu. Questa schermata servirà per configurare le impostazioni di base tra cui anche la scelta dell'algoritmo di monitoraggio.

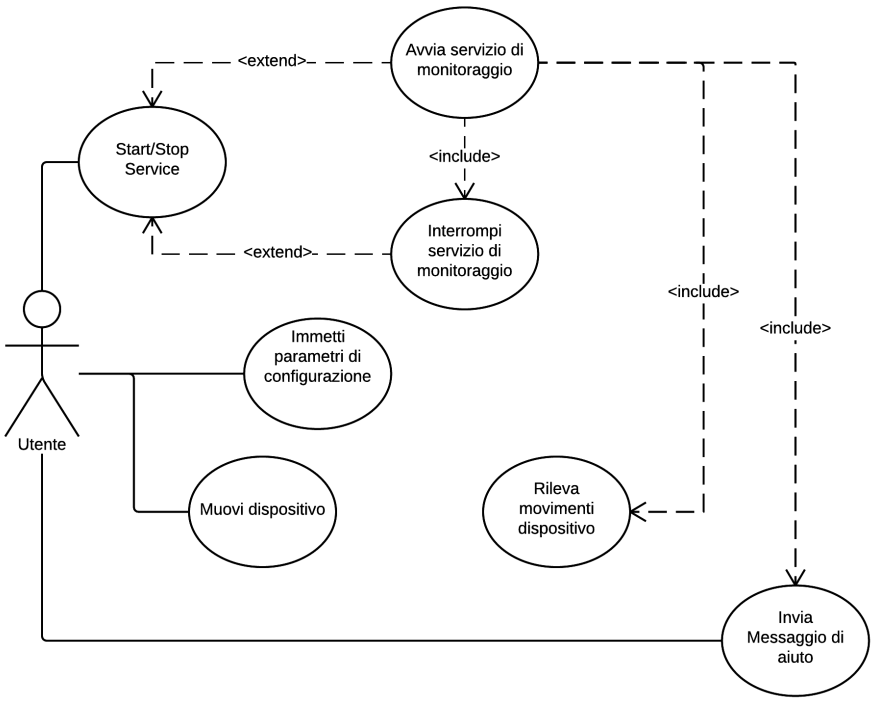
CAPITOLO 3
Progettazione

In questa fase vengono descritte tutte le varie fasi produttive che vanno a definire la struttura e il modello da seguire nella seguente fase di implementazione.

Ogni fase della progettazione è caratterizzata dall'uso di diagrammi e schemi che descrivono in modo visuale lo scenario da implementare. Per quanto riguarda gli schemi UML e i diagrammi di flusso è stato utilizzato il servizio web *Lucidchart*[17]. I mockup sono stati disegnati con il servizio web *Moqups*[18]. Inoltre per tenere traccia dei task e per organizzare in maniera ordinata il lavoro nelle sue fasi si è utilizzato il servizio web *Trello*[19].

3.1 Casi d'uso

Il diagramma dei casi d'uso è piuttosto semplice. Prevede un unico attore, l'utente utilizzatore dell'applicazione, che si limita a configurare l'applicazione, ad avviare o interrompere il monitoraggio e in caso di bisogno di inviare manualmente il messaggio di aiuto. Inoltre in maniera involontaria, durante l'attività sportiva, muove il dispositivo registrando così degli eventi (spostamenti, accelerazioni ecc…) che caratterizzano lo stato o meno di allerta.



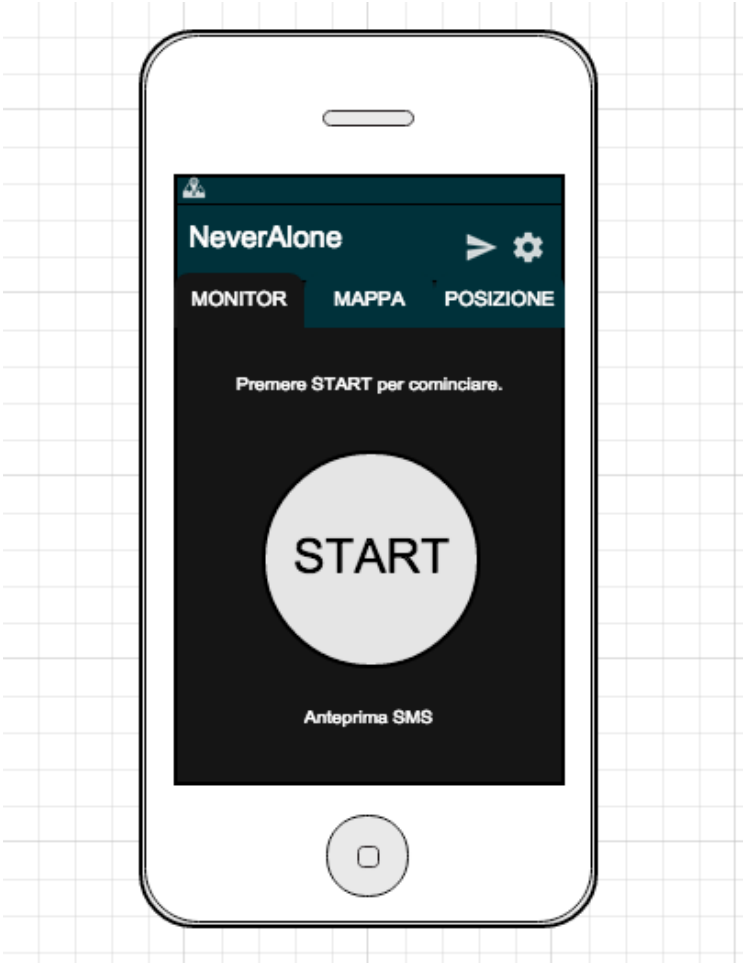
3.2 Mockup

Attraverso la modellazione per mezzo dei *Mockup* si è voluto dare una visione di massima di come dovrà essere l'applicazione finita ancor prima di cominciare lo sviluppo vero e proprio.

Le schermate identificate per permettere l'interazione con l'utente sono relativamente poche. Per prediligere un aspetto moderno ed una navigazione ottimale tra le varie schermate si è optato per l'uso di un layout denominato *swipe tab* che consente di scorrere tra le pagine per mezzo della *gesture* swipe.

3.2.1 Schermata Monitor

Schermata **Monitor**: questa schermata conterrà una call-to-action "START" che ha il compito di avviare il servizio di monitoraggio. Quando il servizio sarà in esecuzione l'etichetta del bottone si tramuterà in STOP e metterà in evidenza, attraverso un timer, i minuti e secondi dall'avvio del servizio. Sopra al bottone troveranno spazio una riga di testo che conterrà le informazioni relative allo stato di allerta (per esempio ultimo movimento rilevato oppure se è stato rilevato un incidente o una situazione di pericolo) mentre al di sotto del bottone un link per visualizzare rapidamente il messaggio che verrà inviato.

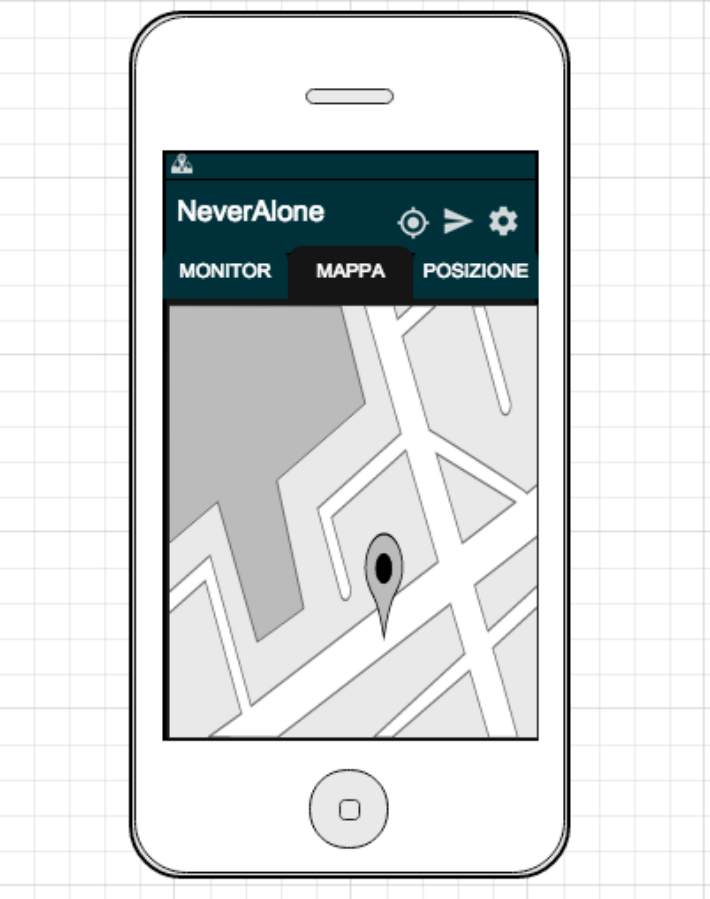


3.2.2 Schermata Mappa

Schermata **Mappa** consente all'utente di visualizzare la mappa del territorio circostante centrata nella posizione attuale. Sarà possibile interagire con la mappa per mezzo delle classiche gesture degli schermi multi-touch, in particolare quelle predefinite di Google Map.

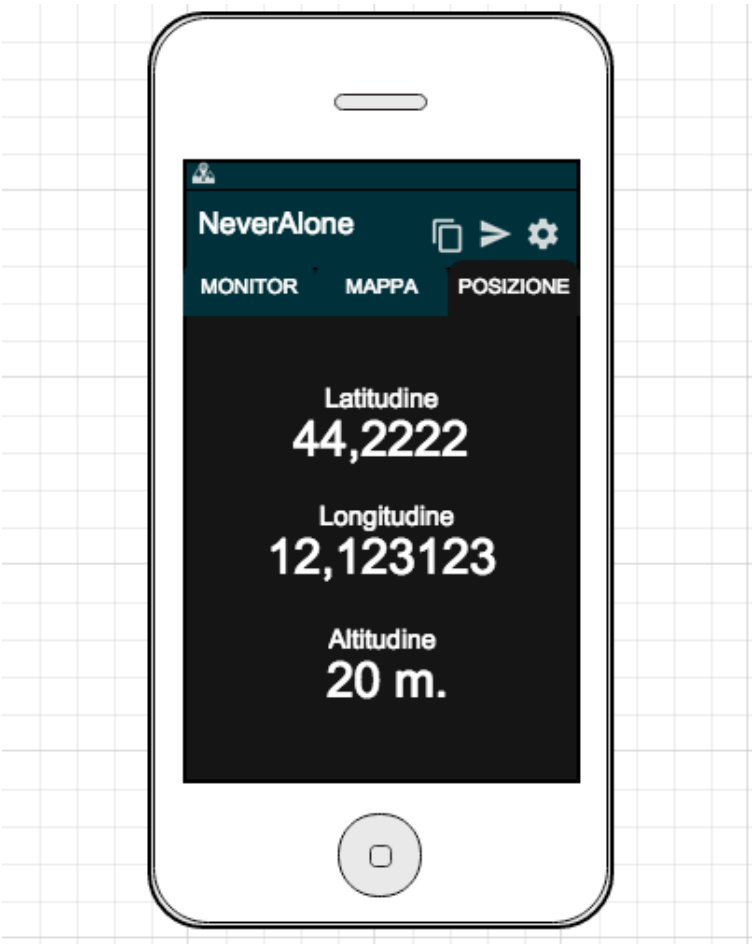
- *pan* per scorrere la mappa
- *zoom* per ingrandire la mappa
- *pinch* per rimpicciolire la mappa
- *rotate* per ruotare la mappa

Oltre alle azioni ella barra delle azioni è presente l'azione centra mappa.



3.2.3 Schermata Posizione

Schermata **Posizione** consente all'utente di avere una visione immediata della sua posizione espressa in coordinate geografiche (Latitudine, Longitudine) e altitudine. Nella barra delle azioni è presente l'azione "copia" che permette di copiare queste coordinate negli appunti del dispositivo.



3.2.4 Schermata Impostazioni

Schermata **Impostazioni** permette all'utente di configurare l'applicazione con i parametri relativi al destinatario dei messaggi di soccorso e alle preferenze dell'algoritmo decisionale in particolare:

- Nome e cognome dell'utente utilizzatore dell'app;
- Email di soccorso
- Telefono di soccorso
- Tempo, espresso in minuti, oltre il quale il sistema, rilevando un possibile incidente, invia la richiesta di aiuto
- Suono, se attivo 30 secondi prima dell'invio della notifica il dispositivo comincia ad emettere un suono per attirare l'attenzione evitando così dei falsi positivi
- Algoritmo, permette di selezionare l'attività (fisica) da rilevare (a piedi, in bici/moto, in auto, in volo)

3.2.5 Notifiche

Per interagire con l'applicazione, anche quando questa non è in primo piano, verrà generata una notifica che permette di vedere immediatamente lo stato attivo o inattivo del monitoraggio. Inoltre permette, attraverso due azioni, di avviare/fermare il servizio e di inviare manualmente il messaggio di aiuto. La notifica potrà essere visualizzata anche quando lo schermo è bloccato così da rendere assolutamente immediata l'interazione in caso di bisogno.

Notifica compressa.

Notifica espansa.

3.3 Material Design

Per rendere più gradevole ed intuitiva la *user experience* si è deciso di rispettare ed applicare le linee guida di usabilità suggerite da Google denominate *Material Design*[20].
Il Material Design è un linguaggio visivo sviluppato da Google che ha l'obiettivo di migliorare l'usabilità. E' un evoluzione del *Flat Design* e come tale ne eredità alcune caratteristiche.

I colori scelti sono stati selezionati dalla palette di colori suggerita da Google[21]. Questi colori se utilizzati correttamente rendono maggiormente fruibili le schermate, inoltre sono in linea con i canoni di "bellezza" del momento.

Anche le icone che sono state scelte sono quelle contenute nella libreria delle *Material Icons*[22] di Google. Questa soluzione è vantaggiosa in quanto l'utente Android si ritroverà immerso in un design "familiare" e quindi anche più chiaro ed intuitivo.

Un concetto molto importante che vale la pena menzionare e quello di *familiarità.* L'utente si troverà più a suo agio ad interagire con un'interfaccia simile per forma e responsività a quelli della piattaforma in cui è racchiuso. Per questo è importante sottolineare come le interfacce utente per iOS e per Android non debbano necessariamente essere identiche: ciò che viene fatto in un modo su iOS si fa in un modo che può essere diverso su Android. Si è data molta importanza al rispetto delle linee guida, prima di tutto perché queste linee guida sono quasi sempre frutto di ricerche molto sofisticate. In secondo luogo perché così facendo la sensazione di familiarità rimane invariata.

3.4 Rilevazione degli incidenti - La tecnologia disponibile

L'applicazione deve essere in grado di rilevare possibili situazioni di pericolo scaturite da un incidente o da un malore.

La chiave per captare queste situazioni attraverso uno smartphone sono i sensori interni al dispositivo. I sensori di un dispositivo si possono classificare in 3 macro categorie:

- sensori di **movimento**
- sensori **ambientali**
- sensori di **posizione**

Per questa applicazione non tutti i sensori sono utili allo scopo, qui di seguito vengono descritti quelli che sono stati sfruttati.

Il *giroscopio* rientra nella categoria dei sensori di movimento e riesce a percepire i movimenti del dispositivo nei 3 assi. Può essere utilizzato per capire come è rivolto il dispositivo e quali movimenti ha fatto.

L'*accelerometro* rientra nella categoria dei sensori di movimento, permette di percepire i valori di accelerazione nei 3 assi. Può essere utilizzato per registrare l'intensità delle accelerazioni che il dispositivo imprime nei 3 assi muovendosi nello spazio.

Il *barometro* rientra nella categoria dei sensori ambientali, permette di percepire i cambiamenti di pressione atmosferica. Può essere utilizzato per rilevare i cambiamenti di altitudine del dispositivo.

La *bussola* rientra nella categoria dei sensori di posizione sfruttando i cambiamenti del campo magnetico permette di capire in che direzione geografica è orientato il dispositivo.

Non tutti i dispositivi dispongono di tutti i sensori, inoltre indipendente da come sono stati implementati, alcuni sensori possono essere **hardware** (esiste un modulo elettronico dedicato sensibile ai movimenti o ai cambiamenti ambientali) oppure **software** (sono il risultato dell'elaborazione dei dati provenienti dai sensori hardware).

Esistono poi altri dispositivi elettronici che ci possono dare informazioni utili:

Il *GPS* (Global Position System) permette di conoscere le coordinate geografiche e l'altitudine del punto in cui si trova il dispositivo.

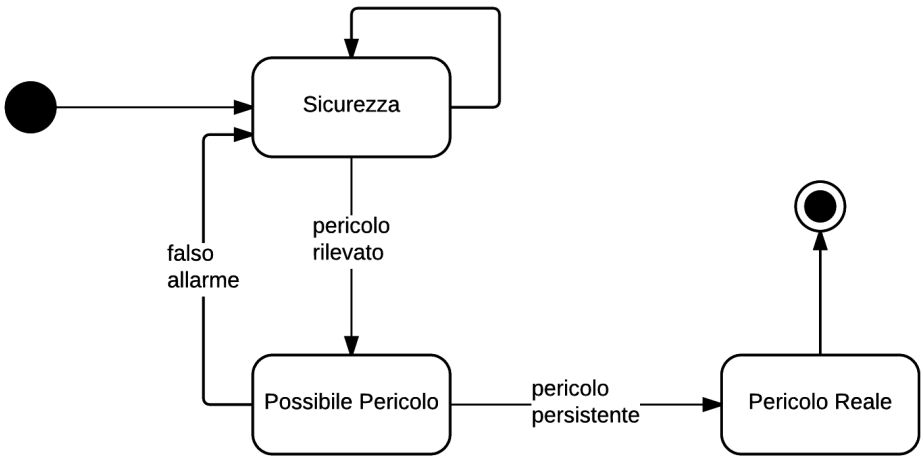
La *Rete Mobile* e la *Rete Wi-fi* permettono in alcune situazioni di determinate le coordinate geografiche in maniera più grossolana.

3.5 Rilevazione degli incidenti - La logica

Ogni sport o attività fisica, in condizioni normali, è caratterizzata da determinati movimenti. Per esempio la corsa è caratterizzata da un movimento sussultorio continuo, diversamente un uscita in bicicletta genera molte meno di queste accelerazioni ma prevalgono altre tipologie di movimenti.

Si può quindi tranquillamente dire che ogni sport, analizzandolo nel dettaglio, è identificato da un susseguirsi di eventi del tutto univoco. Ovviamente in questo caso sarebbe stato improponibile implementare un algoritmo per ogni sport esistente, per cui è stato necessario trovare un compromesso generalizzando e raggruppando il più possibile gli sport in base alle tipologie di movimenti comuni.

Fintanto che gli eventi rilevati rimangono all'interno di un determinato *pattern* allora è possibile asserire che la situazione in cui si trova l'atleta è di **sicurezza**, al contrario se si esce dal pattern prestabilito lo stato cambierà prima in situazione di **possibile pericolo** e se in breve tempo non si ritornerà all'interno del pattern prestabilito lo stato passerà irreversibilmente nella situazione **di pericolo reale** con conseguente invio dei messaggi di aiuto.



Per supportare il maggior numero di sport/attività, senza aggiungere troppa complessità alla UX (*User Experience*) e soprattutto evitando di

creare ridondanza all'interno del codice, sono stati identificati 5 algoritmi ognuno dei quali può essere utilizzato per svariati sport.

- *A piedi*: è adatto a tutte quelle attività che si svolgono a piedi come corsa, camminata, escursionismo;
- *In bici/moto*: è adatto a tutte quelle attività che si fanno utilizzando un mezzo di trasporto a due ruote (bicicletta, moto);
- *In auto*: è adatto a tutte quelle attività che si fanno utilizzando un mezzo a quattro o più ruote (automobilismo);
- *In volo*: è adatto a tutte quelle attività che si fanno volando (parapendio, deltaplano);

3.5.1 A piedi

Quando corriamo o camminiamo di solito siamo costantemente in movimento, in particolare il movimento sussultorio (generato da ogni singolo passo) che si sussegue durante questo tipo di attività fisica è un indizio evidente sulla situazione di sicurezza o normalità.
Attraverso i sensori di movimento captiamo questi movimenti e fintanto che vengono registrati la situazione rilevata rimane di sicurezza. In caso contrario l'assenza di movimenti consecutivi muta lo stato in situazione di possibile pericolo. Infine se per troppo tempo la situazione rimane in uno stato di possibile pericolo allora automaticamente si tramuta in situazione di pericolo.

Sicurezza: Ogni evento rilevante proveniente dall'accelerometro corrisponde ad una situazione di sicurezza.

Possibile Pericolo: L'assenza di eventi rilevanti provenienti dall'accelerometro corrisponde ad una situazione di possibile pericolo.

Reale Pericolo: L'assenza prolungata di eventi rilevanti provenienti dall'accelerometro corrisponde ad una situazione di reale pericolo.

3.5.2 In bici/moto

I mezzi a due ruote hanno la caratteristica di viaggiare in equilibrio. Su questo mezzo un malore o un incidente si conclude quasi sempre con una caduta. Sfruttando i sensori di movimento, in particolare il giroscopio, possiamo stabilire se il dispositivo, che avremmo prece-

dentemente posizionato sul manubrio del veicolo a due ruote, si è ribaltato. Ovviamente in questa situazione anche una curva accentuata può essere rilevata come possibile caduta. Per questo motivo la possibile caduta (possibile situazione di pericolo) si tramuta in caduta (situazione di reale pericolo) se il dispositivo rimane inclinato per troppo tempo.

Sicurezza: un inclinazione entro i limiti prestabiliti indica una situazione di sicurezza.

Possibile Pericolo: Se il giroscopio registra un inclinazione del dispositivo elevata siamo in una situazione di possibile pericolo.

Reale Pericolo: La permanenza prolungata del dispositivo ad una inclinazione elevata corrisponde ad una situazione di reale pericolo.

3.5.3 In auto

Quando ci troviamo in un mezzo a quattro ruote una situazione di pericolo è determinabile registrando una serie di eventi consecutivi. Il dispositivo in questo caso di maggiore utilità è il GPS per rilevare in maniera precisa gli spostamenti del veicolo. E' però in questo caso molto facile rilevare dei falsi positivi se si tengono in considerazione solamente i dati rilevati da un unico modulo. Una situazione di pericolo in questo caso è caratterizzata da una consistente decelerazione, rilevata dall'accelerometro, e da uno stato di quiete successivo in termini di movimento. Per evitare di considerare una brusca frenata ad un semaforo rosso una situazione di pericolo anche qui interviene in aiuto un timer che trasforma una situazione di possibile pericolo in una di effettivo pericolo dopo alcuni minuti.

Sicurezza: determinata dall'assenza di segnali rilevanti provenienti dall'accelerometro oppure da spostamenti geografici significativi registrati dal GPS.

Possibile Pericolo: Un segnale rilevante proveniente dall'accelerometro corrisponde ad una situazione di possibile pericolo.

Reale Pericolo: Se in seguito ad un segnale rilevante proveniente dall'accelerometro non si registrano spostamenti geografici rilevanti siamo in una situazione di reale pericolo.

3.5.4 In volo

Durante una sessione di volo libero (parapendio, deltaplano) la situazione di pericolo è caratterizzata da una repentina perdita di quota e da uno stato di quiete. Queste discipline sono caratterizzate da continue manovre che permettono di sfruttare le correnti ascensionali. Quindi lo stato di quiete può essere captato analizzando i dati della bussola elettronica. La perdita di quota invece è determinabile analizzando i dati provenienti dal barometro.

Sicurezza: Velocità verticali non troppo repentine identificano una situazione di sicurezza. In seguito ad un possibile pericolo manovre rilevate dalla bussola riportano lo stato in "Sicurezza".

Possibile Pericolo: Una perdita di quota elevata, registrata per mezzo del barometro, corrisponde ad una situazione di possibile pericolo.

Reale Pericolo: Se in seguito ad una perdita di quota non vengono registrati cambiamenti di direzione provenienti dalla bussola elettronica siamo in una situazione di reale pericolo.

3.6 Gestione dei falsi positivi

I falsi positivi in questa tipologia di applicazioni sono all'ordine del giorno. La vera difficoltà sta proprio nel capire quando considerare una situazione di reale pericolo invece che di possibile pericolo o di normalità. Basti pensare ad una scarpa slacciata che costringe l'atleta ad interrompere la sua corsa per qualche decina di secondi, oppure ad una curva parabolica durante una discesa in bicicletta che fa registrare al dispositivo qualcosa di simile ad una caduta. Questi sono solo alcuni esempi delle decine che possono capitare durante uno di questi sport. Il metodo utilizzato per evitare questi inconvenienti è stato quello di implementare 2 livelli di prevenzione ai falsi positivi:

- un conto alla rovescia che calcola un tempo predefinito oltre il quale una situazione di sospetto diventa certezza. Se questo primo livello di sfuggire ad un falso positivo non è sufficiente interviene un secondo livello di difesa: un segnale acustico.

- Un segnale acustico viene attivato 30 secondi prima dell'imminente passaggio allo stato di *pericolo reale.* In questo modo è possibile scongiurare il falso positivo o riprendendo l'attività o interrompendo il servizio di monitoraggio per poi riprenderlo in un secondo momento.

3.7 Gestione dei consumi

La tipologia di applicazione richiede un alta affidabilità. Un allenamento può durare anche alcune ore ed anche i dispositivi mobili meno potenti devono essere in grado di rimanere vigili per tutta la sessione di allenamento. Questo si traduce in una complessità tale da non mettere in crisi l'hardware del dispositivo con consumi eccessivi. Si ha quindi la necessità che l'applicazione sia il più efficiente possibile rimanendo in ogni caso efficace nel suo raggio di azione.

Caratteristica fondamentale per limitare i consumi è che l'applicazione sia in grado di lavorare in background. In questo modo i consumi calano drasticamente in quanto il display non è necessario che rimanga acceso.

Un'altro importante accorgimento è stato quello di limitare il più possibile l'uso del GPS. Quando è in primo piano alcune schermate richiedono la geo-localizzazione precisa del dispositivo per mezzo del GPS. Il GPS consente di ottenere questa informazione con una precisione accurata, ma ha lo spiacevole svantaggio di incrementare considerevolmente i consumi di batteria. Per questo motivo è necessario che rimanga acceso solamente per il tempo necessario e soprattutto solamente quando le schermate lo richiedono. Il monitoraggio avviene in background, quindi non è necessario che l'applicazione rimanga in primo piano per funzionare correttamente.

Anche gli algoritmi sono fatti in modo per limitare l'uso del GPS. Attualmente l'unico algoritmo che necessità del modulo GPS per funzionare nella fase di monitoraggio è l'algoritmo "in auto". Di solito in auto il consumo di batteria non è un fattore critico in quanto si ha a disposizione il collegamento con la rete di alimentazione, comunque sia anche in questa situazione si è optato per la filosofia del risparmio

e la localizzazione della posizione è effettivamente attivata solo in caso di bisogno per validare un risultato sospetto.

3.8 Il messaggio di aiuto

Il messaggio di aiuto che viene inviato quando viene rilevato un incidente è di fondamentale importanza. I canali di invio possono essere molteplici: notifiche push, e-mail, sms, condivisioni social. Senza dubbio il metodo più efficace per ottenere una eccellente *deliverability* del messaggio è l'uso degli SMS.

Infatti gli SMS (*Short Message Service*) possono sfruttare le reti GSM/UMTS che si è visto nel primo capitolo, sinergicamente, hanno una rete di infrastrutture che copre gran parte del pianeta in cui viviamo. In particolare dove non arrivano le reti UMTS ed LTE, che permettono ai nostri dispositivi di collegarsi ad internet in banda larga, arriva la rete GSM che come minimo ci permette di spedire un messaggio SMS.

In seconda istanza bisogna considerare che la maggior parte delle persone tende a dare maggiore priorità ad un SMS rispetto ad un messaggio e-mail o ad un post su un Social Network. Le motivazioni sono da ricercare soprattutto dall'uso quotidiano che si fa di queste tecnologie. Infatti quotidianamente la nostre caselle e-mail ricevono pubblicità, iscrizioni e spam che possono fare passare in secondo piano un e-mail importante. I social Network allo stesso modo sono invasi da pubblicità e contenuti virali che non possono garantire visibilità al nostro messaggio di aiuto. L'SMS invece, anche grazie all'uso moderato che si fa di questa tecnologia, ha acquisito una priorità maggiore. Non siamo più abituati a ricevere SMS e quindi, in maniera del tutto involontaria tendiamo a considerarli più importanti.

CAPITOLO 4
Implementazione

In questo capitolo vengono illustrate le tecnologie presenti all'interno del progetto e nel dettaglio verranno descritte le fasi salienti dell'implementazione dell'applicazione.

4.1 Ambiente di sviluppo

La piattaforma Android è sviluppata in Java, per cui è indispensabile installare nel proprio ambiente di sviluppo il JDK (*Java Development kit*).
Tutti i tool utili durante lo sviluppo e il testing delle App (come programmi, ed emulatori) sono contenuti nell'Android SDK (*Software Development kit*).
Infine è necessario un IDE (*Integrated Development Environment*) che include tutti gli strumenti utili alla programmazione. I due IDE più utilizzati per sviluppare su piattaforma Android sono Eclipse con l'aggiunta dell'add-on ADT (*Android development tools*) oppure il nuovo ambiente di sviluppo integrato Android Studio[23]. In questa tesi si è optato per Android Studio in quanto è l'IDE ufficiale per lo sviluppo delle applicazione su Android.

4.2 Retro-compatibilità

Come detto nel primo capitolo lo sviluppo su piattaforma Android mette lo sviluppatore di fronte ad un dilemma: sviluppare un'applicazione compatibile con anche le versioni più datate di Android rinunciando alle ultime funzionalità introdotte o limitare il bacino di utenza a favore di maggiori potenzialità? Nel caso specifico di questo progetto si è cercato di trovare un compromesso. Non volendo rinunciare ad un'interfaccia con un aspetto moderno e volendo raggiungere un bacino di utenza il più elevato possibile si è optato di supportare tutte le versioni più recenti della versione 15 (4.0.4). In questo modo ap-

prossimativamente la percentuale di utenti raggiunti sarà di oltre il 90% sul totale dell'utenza Android.

All'interno di un progetto Android l'SDK minimo supportato viene definito all'interno dello script Groovy *build.gradle*. Attraverso la sintassi *minSdkVersion* è possibile definire l'SDK minimo supportato, mentre con *compileSdkVersion* e con *targetSdkVersion* viene definita la versione di sdk usato per la compilazione.

Oltre a queste direttive è possibile definire anche una serie di dipendenze che in fase di compilazione si occuperanno di definire quali sono le librerie da importare per supportare alcune parti del progetto.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:23.0.1'
    compile 'com.android.support:design:23.0.1'
    compile 'com.google.android.gms:play-services:8.1.0'
}
```

In questo progetto vengono importate le librerie di supporto *appcompat* per supportare il widget *ViewPager* e *Toolbar* e la libreria per *TabLayout*.

Inoltre viene importata la libreria Google Play Service che consente l'utilizzo delle mappe all'interno dell'applicazione.

4.3 Il file Manifest

Ogni applicazione sviluppata su Android deve avere un file denominato *AndroidManifest.xml*. Questo file contiene le informazioni essenziali che il sistema operativo deve conoscere per eseguire l'applicazione.

All'interno di questo file vengono descritte le componenti utilizzate (Activity, Service, Broadcast receiver), il nome di app e package ma soprattutto vengono dichiarati i permessi necessari all'utilizzo dell'App. In particolare all'interno di questo progetto vengono richiesti i seguenti permessi:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
```

4.4 Componenti Grafici utilizzati

All'interno di questo progetto, per l'implementazione grafica delle schermate, si è fatto largo uso dei *Widget*. Con il termine widget si intendono tutte le sottoclassi della superclasse *View*. La definizione di questi widget viene fatta all'interno dei singoli file xml contenuti nella cartella delle risorse chiamata *layout*. Generalmente per ogni schermata, quindi per ogni Activities, esiste uno di questi file. Al suo interno sono elencati, attraverso un metalinguaggio di markup (XML) tutti i widget utilizzati dalla schermata.
Qui di seguito vengono elencati i widget utilizzati:

- **TextView**
  Questo widget consente di mostrare un etichetta testuale all'utente.

- **EditText**
  Al contrario della TextView l'EditText è configurata per essere editabile.

- **Space**
  É una semplice classe che permette di inserire uno spazio visuale tra gli altri elementi.

- **Chronometer**
  Questo widget consente di inserire un cronometro testuale. Attraverso i metodi start() e stop() è possibile avviare o interrompere il contatore del cronometro, inoltre attraverso il metodo setBase è possibile specificare la data ed ora di avvio del cronometro, in modo tale da inizializzarlo al riavvio dell'activity all'interno del quale è incluso.

- **ImageButton**

É una sottoclasse di ImageView, e consente di aggiungere alle nostre activities un contenuto di tipo immagine.

- **CheckBox**

Questo widget stampa a video un campo di input di tipo casella di controllo. É un tipo di dato bollano (true, false).

- **Switch**

Lo switch è un widget con un funzionamento simile al checkbox. Anch'esso consente la scelta tra due stati (true, false) che però possono avere un valore specificatamente definito. L'utente può, attraverso il drag and drop di un icona passare da uno stato all'altro dello switch.
All'interno di questo progetto è stato utilizzato per consentire all'utente di selezionare l'orientamento (landscape, portrait) di montaggio del dispositivo sul manubrio della bicicletta, utile per il corretto funzionamento all'interno dell'algoritmo bici/moto.

- **Spinner**

Il widget Spinner consente la scelta di un valore da un elenco. É il widget utilizzato per permettere la scelta dell'algoritmo da utilizzare durante il monitoraggio.

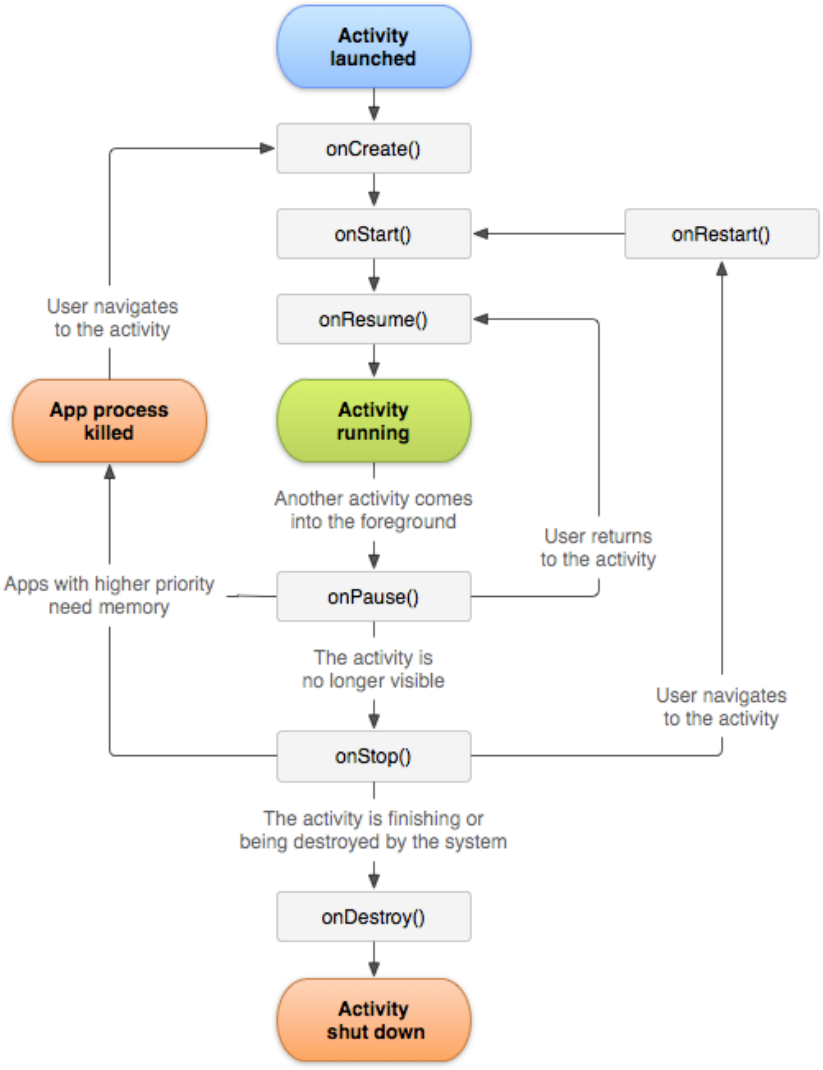
4.5 Classi e Componenti Android utilizzate

In questo paragrafo vengono illustrate le classi e le componenti Android più interessanti che sono state utilizzate all'interno del progetto.

4.5.1 Activities

Una *Activity* è una classe che consente di disegnare un'interfaccia visuale sul display. Ad ogni Activity in genere corrisponde una schermata dell'applicazione. Essa eredita tutte le proprietà e i metodi per interagire con il ciclo di vita dell'activity e quindi dell'applicazione stessa.

Il ciclo di vita di un Activity è chiaramente rappresentato da questo diagramma:



All'interno di questa applicazione è stata utilizzata una *Action Bar* per permettere una moderna navigazione tra le schermate. Per garantire la retro-compatibilità di questa caratteristica è stato usato un particolare tipo di Activity denominato *AppCompatActivity*.

Le Activity sono un componente introdotto fin dalla primissima versione di Android (Api Level 1).

4.5.2 Fragment

Un *Fragment* rappresenta una porzione dell'interfaccia dell'applicazione e viene utilizzata in simbiosi con le *Activities*. In qusta applicazione è stato utilizzato come componente per creare l'interfaccia a schede che caratterizza la navigazione dell'applicazione (*Swipe Tab*). Il suo ciclo di vita è molto simile a quello di un'Activity.

I Fragment sono un componente introdotto da Android 3.0 (Api Level 11).

4.5.3 IntentService

Il cuore dell'applicazione è il componente *IntentService* (che estende Service). L'intentService, a differenza del Service, incoropora al suo interno un *Thread*. In questa tesi il servizio implementa anche un Listener dei sensori (*SensorEventListener*). Questo tipo di servizio può essere avviato o interrotto dall'utente. Viene eseguito in background e non ha bisogno di un'interfaccia visuale.

Per eseguire un Servizio è sufficiente che un componente chiami il metodo *startService*. A quel punto verrà chiamato il metodo *onCreate*, dopodiché verrà eseguito il Thread implementato nel metodo *onHandleIntent*.

Per interrompere il servizio un componente dell'applicazione dovrà chiamare il metodo *stopService*.

Gli IntentService sono un componente introdotto nell'Api Level 3.

4.5.4 Service

É stato necessario anche fare uso di un *Service* per permettere l'avvio di un servizio per la geo-localizzazione. Non è possibile im-

plementare un Listener della posizione (*LocationListener*) all'interno dell'intent Service, per cui si è creato un Servizio ad-hoc.

Per eseguire un Servizio è sufficiente che un componente chiami il metodo *startService*. A quel punto verrà chiamato il metodo *onCreate*.

Per interrompere il servizio un componente dell'applicazione dovrà chiamare il metodo *stopService*.

I Service sono un componente introdotto fin dalla primissima versione di Android (Api Level 1).

4.5.5 Listener

I listener come, dice la parola stessa, rimangono in ascolto di determinati eventi provenienti dall'hardware del sistema. In questo progetto sono stati implementati all'interno del Servizio di monitoraggio per ascoltare gli eventi provenienti dai sensori (*SensorEventListener*) e dal servizio di localizzazione (*LocationListener*). Ad ogni modulo è collegato uno specifico listener.

Il *LocationListener* implementa alcuni metodi che vengono richiamati allo scatenarsi di un evento. In particolare il metodo *onLocationChanged* viene richiamato appena viene rilevato un cambiamento della posizione. L'avvio del listener avviene attraverso il comando *requestLocationUpdates* nel quale si possono definire dei parametri utili come il *provider,* l'intervallo minimo di tempo tra due rilevazioni di posizione consecutive, la differenza minima di distanza tra due rilevazioni consecutive.
La scelta del *provider* può essere automatizzando configurandone dei criteri (per mezzo della classe *Criteria*) di scelta che si basano essenzialmente su precisione e consumo batteria.

Il *SensorEventListener* implementa principalmente il metodo *onSensorChanged* che viene richiamato quando un listener rileva lo scatenarsi di determinati eventi. Un listener viene registrato per mezzo del metodo *registerListener.* Quando esso non è più necessario viene distrutto per mezzo del metodo *unregisterListener.*

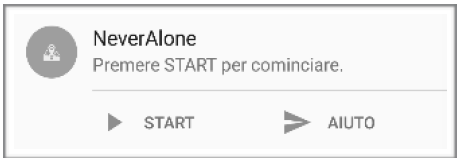
I *SensorEventListener* sono stati introdotti in Android dalla versione Api Level 3, mentre i *LocationListener* sono stati introdotti dalla prima versione di Android Api Level 1.

4.5.6 Broadcast Receiver

Il *BroadcastReceiver* è un componente che ha la capacità di recepire degli eventi, tipicamente quelli di sistema. come la ricezione di un messaggio, il superamento di una certa soglia di batteria, l'accensione o lo spegnimento del dispositivo. In questo progetto sono stati utilizzati per recepire delle azioni contenute nelle notifiche: avvio e interruzione del servizio di monitoraggio ed invio manuale dell'sms di aiuto. I *BroadcastReceiver* sono un componente introdotto fin dalla primissima versione di Android (Api Level 1)

4.5.7 Notifiche

Vengono utilizzate dal sistema e dalle applicazioni per dare informazioni specifiche all'utente, anche quando queste ultime non sono in primo piano. In questo progetto sono utilizzate per informare l'utente sullo stato attivo/inattivo del sistema di monitoraggio mostrando di conseguenza delle azioni appropriate.



Per implementare questa importante interfaccia interviene in nostro aiuto la classe *Notification Builder*. Questo costruttore permette rapidamente l'implementazione di una notifica nel formato standard. Di base in una notifica sono presenti diverse aree ognuna delle quali occupata da una particolare parte della notifica: Un titolo, un icona, un contenuto testuale, delle informazioni accessorie, la data e ora di creazione e un icona piccola.

Le Notifiche sono presenti dalla prima versione di Android, ma la classe costruttrice *NotificationBuilder* utilizzata all'interno di questo progetto è stata introdotta dall'Api Level 11.

4.5.8 Notifiche Toast

Le Notifiche di tipo *Toast* sono delle finestrelle rettangolari, con sfondo nero e testo bianco, che appaiono nella parte bassa del display e

con una visibilità limitata a qualche secondo di tempo. Il nome è dovuto al caratteristico effetto a comparsa nel basso dello schermo di una finestrella rettangolare. Sono un efficace metodo per interagire con l'utente comunicandoli l'avvenimento di determinate azioni. In questa tesi sono state utilizzate per comunicare all'utente l'avvio e l'interruzione del servizio di monitoraggio e per notificare l'avvenuta spedizione del messaggio di aiuto.

Sono molto semplici da implementare. Anche qui esiste una classe builder *Toast.makeText* che permette con pochi parametri l'esecuzione di un notifica Toast.

I Widget *Toast* sono stati introdotti nella prima versione di Android Api Level 1.

4.5.9 SmsManager

L'*SmsManager* è la classe che viene utilizzata per l'invio dei famosi messaggi di aiuto. Anche in questo caso è molto semplice l'implementazione. Una volta istaziata la classe è sufficiente chiamare il metodo *sendTextMessage* passandogli parametri di destinatario e messaggio di testo.

SmsManager è stato introdotto dall'Api Level 4 di Android.

4.5.10 SharedPreferences

L'applicazione deve disporre di un luogo in cui salvare le preferenze del contatto per non doverle reimmeterle ad ogni avvio dell'applicazione. A questo scopo è stata utilizzata la classe *SharedPreferences* che mette a disposizione un insieme di metodi per archiviare in modo persistente i dati rappresentati con una coppia chiave-valore. Le SharedPreferences possono essere private all'applicazione o condivise con le altre applicazioni. Di default vengono definite private. Attraverso il metodo *getDefaultSharedPreferences* è possibile recuperare l'oggetto contenente le informazioni memorizzate in precedenza. Tramite i metodi getter (getString, getInt, etc…) è possibile richiedere il valore a partire da una specifica chiave. La memorizzazione di una coppia chiave-valore avviene invece prima instanziando un costrutto *SharedPreferencesEditor,* poi utilizzando i metodi setter (putString,

putInt, etc…) ed infine confermando le modifiche con il metodo *commi* dell'oggetto Editor.

Le *SharedPreferences* sono un componente standard di Android presente fin dagli albori (Api Level 1)

4.5.11 Action Bar

La *Action Bar* è un quel componente all'interno del quale sono localizzate oltre al nome della App o della schermata che stiamo visualizzando anche i pulsanti azione e il menu di navigazione. All'interno di questo progetto trovano spazio l'importante icona di aiuto rapido oltre che al bottone per accedere alla schermata di configurazione. Questo componente è supportato da Android 3.0 (Api Level 11) ed è quindi inferiore al livello minimo che si vuole mantenere per lo sviluppo di questo progetto.

4.6 Rilevazione degli incidenti - Implementazione

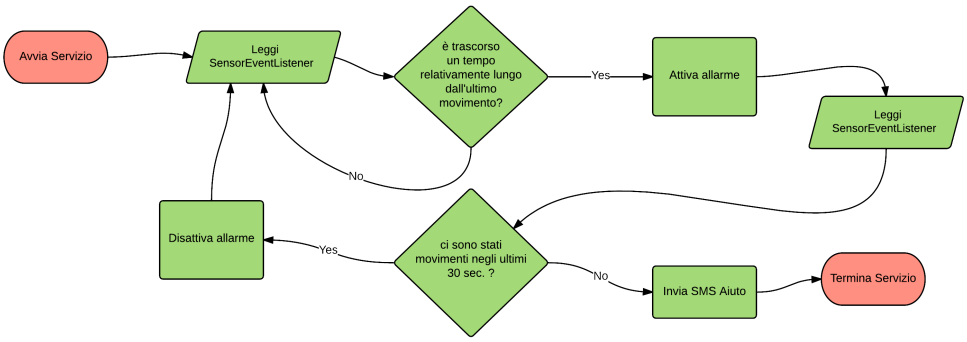
Come è stato detto nel precedente capitolo ad ogni algoritmo di monitoraggio degli incidenti corrisponde un *pattern* che determina una situazione di Sicurezza e un *pattern* a cui corrisponde una situazione di pericolo. Se la situazione di pericolo si prolunga nel tempo automaticamente si tramuta in una reale situazione di pericolo.

Tutti gli algoritmi cominciano il loro ciclo di vita con l'avvio del servizio di monitoraggio e terminano o con l'invio del messaggio d'aiuto o con la terminazione da parte dell'utente del servizio.

Qui di seguito è stato illustrato dettaglio implementativo degli algoritmi per ogni disciplina.

4.6.1 A piedi

L'algoritmo "*A piedi"* inizializza il listener sull'accelerometro. Dopodiché l'algoritmo procede con una serie di letture consecutive degli eventi registrati dal listener interessato. Nel caso dell'algoritmo "A Piedi" il listener interessato si occupa di ascoltare il sensore accelerometro scartando le accelerazioni irrilevanti.



Nel primo step viene verificato se ci sono stati dei movimenti significativi. Ogni qualvolta il *listener* registra un movimento significativo, la data di ultimo movimento viene aggiornata. Il thread principale ad ogni iterazione calcola il tempo trascorso dall'ultimo movimento e nel caso questo sia superiore ad una certa soglia (presa dalle preferenze del contatto) viene avviato un allarme che avverte dell'imminente invio del messaggio di aiuto. Se durante questi ultimi 30 secondi continuano a non essere registrati movimenti allora viene inviato il messaggio di aiuto.

L'accelerometro rileva i movimenti espressi in $\mathbf{m/s^2}$ sui 3 assi (asse x, asse y, asse z). Un movimento viene considerato significativo se: il valore assoluto della differenza tra la somma dei valori registrati sui 3 assi di due letture consecutive diviso per il tempo trascorso tra le due letture è superiore ad una certa soglia.

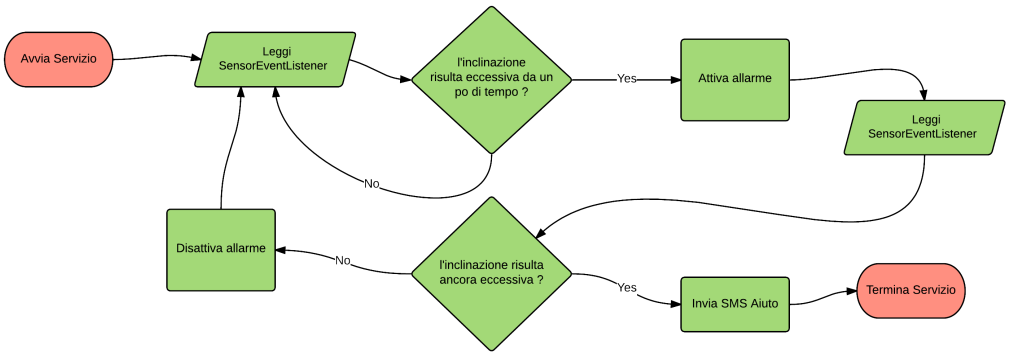
```
float speed = Math.abs(
        x_sen_acc + y_sen_acc + z_sen_acc -
        last_sen_acc_x - last_sen_acc_y - last_sen_acc_z)/ diffTime * 10000;

if (speed > SHAKE_THRESHOLD_ONFOOT) {

    Log.i("ACCELEROMETRO", x_sen_acc + " " + y_sen_acc + " " + z_sen_acc);

    setLastSecurityStatusTime(curTime);
    updateMainActivityInfo("Sembra tutto ok!");

}
```

4.6.2 In Bici/Moto

Per i veicoli a due ruote (piuttosto che tutti quelli che in una situazione normale sono in equilibrio) viene considerata l'inclinazione del dispositivo che, se posizionato adeguatamente sul manubrio del mez-

zo, permette efficacemente di determinare l'inclinazione del mezzo di trasporto.

Se viene registrata un inclinazione eccessiva che permane per un primo intervallo di tempo viene attivato l'allarme. Se l'inclinazione non viene ripristinata nel secondo intervallo di tempo allora viene inviato il messaggio di aiuto.



In questo caso viene inizializzato il sensore *vettore di rotazione*. Il vettore di rotazione è un sensore software che sfrutta le capacità del *giroscopio* per determinare angolo di imbardata, di beccheggio e di rollio. Il vettore di rotazione rappresenta l'orientamento del dispositivo come combinazione degli angoli rispetto ciascun asse.

Prima di tutto il vettore di rotazione è convertito in una matrice attraverso il metodo *getRotationMatrix*. A questo punto la matrice è pronta per essere data in pasto al metodo *getOrientation* che restituisce gli angoli di inclinazione sui 3 assi.

I valori sono espressi in radianti e sono positivi per rotazioni nel senso orario e negativi per rotazione in senso antiorario.

- *Azimuth* o imbardata: valore di rotazione attorno all'asse Z, (-π, π)
- *Pitch* o beccheggio: valore di rotazione attorno all'asse X, (-π/2, π/2)
- *Roll* o rollio: valore di rotazione attorno all'asse Y, (-π, π)

Il dispositivo deve essere fissato sul manubrio, a seconda che sia stato fissato *orizzontalmente* o *verticalmente* rispetto al terreno, oppure con un orientatamento *portrait* o *landscape* sarà necessario rimappare il sistema di coordinate con il metodo *remapCoordinateSystem* che permette di invertire i valori tra i vari assi.

La tabella seguente riepiloga le tipologie di mappature possibili:

	Portrait Vertical	Portrait Horizontal	Landscape Vertical	Landscape Horizontal
Azimuth	Y	Z	Y	Z
Pitch	X	X	Z	X
Roll	Z	Y	X	Y

I valori interessanti per capire quando la bicicletta si trova a terra o in piedi sono quelli di beccheggio e quelli di rollio. Infatti cadendo lateralmente il sensore registrerà un movimento di rollio, mentre un cappottamento sulla ruota anteriore (impuntata) o un cappottamento sulla ruota posteriore (impennata) farà registrare un movimento di beccheggio. In questo modo è possibile garantire una copertura della quasi totalità delle tipologie di cadute.

```
if(
    (Math.abs(orientationVals[1]) > Math.PI/4) ||
    (Math.abs(orientationVals[2]) > Math.PI/4))
{
    updateMainActivityInfo("Rilevata possibile Caduta!");
} else {
    setLastSecurityStatusTime(curTime);
    updateMainActivityInfo("Sembra tutto ok!");
}
```

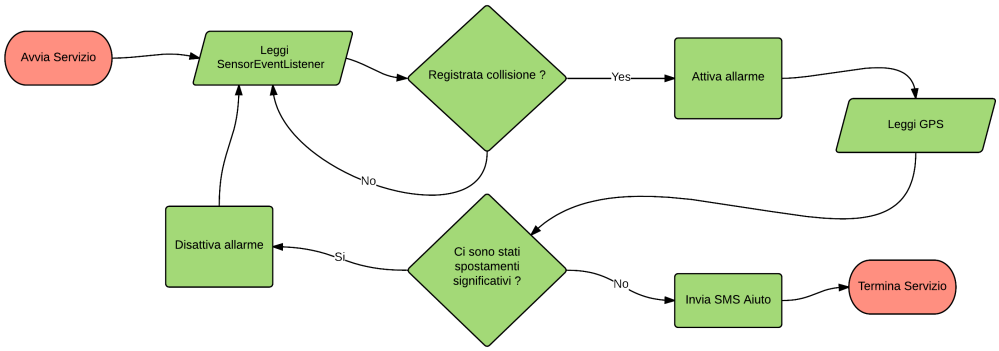
Il valore massimo di rollio e beccheggio prima di entrare in stato di possibile pericolo è di 45 gradi, ovvero di $\pi/4$ radianti o 1/8 di angolo giro.

4.6.3 In auto

Per i veicoli a quattro ruote non esiste inequivocabilmente un evento che determini una situazione di pericolo come quello di un incidente. Per questo l'algoritmo implementato procede interpretando i dati provenienti da due fonti distinte in due momenti differenti:

All'inizio l'algoritmo rimane costantemente in ascolto di un valore elevato proveniente dall'accelerometro. Un incidente automobilistico infatti genera delle forze di accelerazione/decelerazione elevate. Quando l'accelerometro registra dei valori significativamente grandi di

accelerazione, l'algoritmo passa in stato di possibile pericolo ed entra nel secondo step. Il secondo step consiste nel validare questo stato in uno di reale pericolo. La validazione avviene monitorando gli spostamenti successivi alla presunta collisione, in modo accurato, attraverso il modulo GPS. Se nei successivi 60 secondi vengono registrati degli spostamenti che fanno allontanare il dispositivo dal luogo della collisione allora lo stato di allerta viene ripristinato in quanto si è verificato un evidente episodio di falso positivo.



Nel primo step la funzione di determinazione della collisione è uguale a quella utilizzata come contattassi nell'algoritmo "a piedi", con la differenza che i valori di scuotimento sono più elevati:

```
//Axis Acceleration
float x_sen_acc = event.values[0];
float y_sen_acc = event.values[1];
float z_sen_acc = event.values[2];


if ((curTime - last_sen_acc_move) > SENSOR_LISTENER_INTERVAL) {
    long diffTime = (curTime - last_sen_acc_move);
    last_sen_acc_move = curTime;

    float speed = Math.abs(
            x_sen_acc + y_sen_acc + z_sen_acc -
            last_sen_acc_x - last_sen_acc_y - last_sen_acc_z)/ diffTime * 10000;

    if (speed > SHAKE_THRESHOLD_4WHEELS) {

        last_collision = curTime;
        updateMainActivityInfo("Rilevata possibile collisione!");

    }

    last_sen_acc_x = x_sen_acc;
    last_sen_acc_y = y_sen_acc;
    last_sen_acc_z = z_sen_acc;
```

Nel secondo step vengono abilitati gli aggiornamenti della posizione provenienti dal *LocationListener*, e ad ogni spostamento viene calcolata la distanza percorsa rispetto al punto di impatto in metri attraverso il metodo *distanceBetween*:

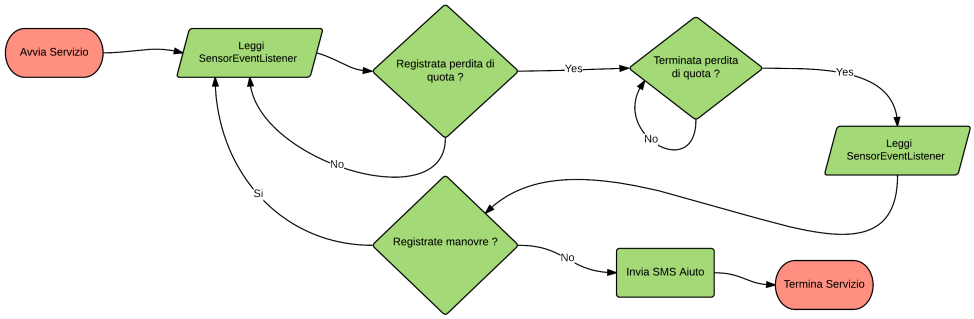
```
if(last_collision_location != null) {

    Location.distanceBetween(
            location.getLatitude(),
            location.getLongitude(),
            last_collision_location.getLatitude(),
            last_collision_location.getLongitude(),
            results);

    ((MyApplication) this.getApplication()).setDistance(results[0]);

} else {
    last_collision_location = location;
}
```

Questo metodo prende in input le coordinate geografiche latitudine e longitudine di due punti distinti. In questo caso prenderà in input le coordinate della posizione corrente e le coordinate della posizione in cui è stata registrata la collisione. Se dopo 60 secondi la distanza calcolata non supera i 100 metri viene inviato l'sms di aiuto in quanto

significa che l'auto non si sta muovendo e molto probabilmente è avvenuto veramente un incidente.

4.6.4 In Volo

Gli sport come il parapendio o il deltaplano, a differenza del volo con veicoli a motore, volando sfruttando le correnti ascensionali provenienti da colonne di aria calda (volo in termica) oppure le correnti ascensionali incanalate dalla morfologia del territorio (volo in dinamica), in ogni caso una sessione di volo è determinata da ripetute manovre per sfruttare queste tipologie di correnti. Una situazione di pericolo è ovviamente sempre anticipata da una perdita di quota che può degenerare in una caduta al suolo nei peggiori dei casi, oppure si può evolvere in una regolare attività di volo. La regolare attività di volo è intercettata dal sensore magnetico (bussola) a patto che contemporaneamente ad essa non venga registrata anche una perdita di quota rilevata dal sensore barometro. In questo caso il pattern di pericolo vince su quello di sicurezza.



Il rilevamento dell'altitudine è possibile per mezzo del metodo *getAltitude* che analizzando la pressione atmosferica corrente con la pressione atmosferica al livello del mare (questo dato è memorizzato nella costante *PRESSURE_STANDARD_ATMOSPHERE*) restituisce l'altitudine in metri.

```
float altitude = SensorManager.getAltitude(SensorManager.PRESSURE_STANDARD_ATMOSPHERE, event.values[0]);

if (last_altitude == 0) {
    last_altitude = altitude;
}
```

La velocità di caduta è calcolata attraverso la formula

```
if ((curTime - last_altitude_time) > SENSOR_LISTENER_INTERVAL) {

    long diffTime = (curTime - last_altitude_time);
    last_altitude_time = curTime;

    float fallSpeed = (last_altitude - altitude)/ diffTime * 10000;


    //se la perdita di quota è eviedente
    if (fallSpeed > FALL_THRESHOLD) {

        if(start_fall_time == 0) {
            start_fall_time = curTime;
            return;
        }

    } else {
        lastSecurityStatusTime = curTime;
        start_fall_time = 0;
        updateMainActivityInfo("Sembra tutto ok!");
    }

    last_altitude = altitude;
```

Al termine di una caduta viene verificato la staticità del velivolo analizzando i dati provenienti dal campo magnetico (*Sensor.TYPE_MAGNETIC_FIELD*).

Il sensore restituisce i valori del campo magnetico, nei 3 assi, espressi in micro-Tesla (uT). *ROTATION_ANGLE_THRESHOLD* è una costante che definisce l'angolo di rotazione minimo per considerare un movimento una manovra.

```
float x_sen_magf = event.values[0];
float y_sen_magf = event.values[1];
float z_sen_magf = event.values[2];

if(
    Math.abs(Math.round(x_sen_magf) - last_sen_magf_x) > ROTATION_ANGLE_THRESHOLD ||
    Math.abs(Math.round(y_sen_magf) - last_sen_magf_y) > ROTATION_ANGLE_THRESHOLD ||
    Math.abs(Math.round(z_sen_magf) - last_sen_magf_z) > ROTATION_ANGLE_THRESHOLD) {

    last_veer = curTime;
    updateMainActivityInfo("Sembra tutto ok!");

    last_sen_magf_x = x_sen_magf;
    last_sen_magf_y = y_sen_magf;
    last_sen_magf_z = z_sen_magf;
}
```

4.7 Implementazione del servizio

Tutti gli algoritmi di rilevazione degli incidenti sono implementati all'interno di un unico servizio. Quando l'utente decide di avviare il monitoraggio l'applicazione istanzia un servizio in background che seleziona l'algoritmo da utilizzare.

Il primo metodo che viene chiamato all'avvio del servizio è il metodo *onCreate*(). In questa fase a seconda dell'algoritmo selezionato vengono inizializzati i soli oggetti utili al funzionamento dello specifico algoritmo. Così facendo si evita di occupare risorse inutilmente risparmiando memoria.

```
//Sensor listener
senSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);


//recupero l'algoritmo selezionato
SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
int algo = sharedPref.getInt("edit_algo", 0);

switch (algo) {
    case 0:
        senAccelerometer = senSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        senSensorManager.registerListener(this, senAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
        break;
    case 1:
        senRotationvector = senSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
        senSensorManager.registerListener(this, senRotationvector, SensorManager.SENSOR_DELAY_NORMAL);
        break;
    case 2:
        senAccelerometer = senSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        senSensorManager.registerListener(this, senAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);

        Intent service = new Intent();
        service.setComponent(new ComponentName(this, MyServiceLocation.class));
        this.startService(service);

        break;
    case 3:
        senPressure = senSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
        senMagneticfield = senSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
        senSensorManager.registerListener(this, senPressure, SensorManager.SENSOR_DELAY_NORMAL);
        senSensorManager.registerListener(this, senMagneticfield, SensorManager.SENSOR_DELAY_NORMAL);
        break;
}
```

Dopo l'*onCreate* viene richiamato il metodo *onHandleIntent*() che implementa il nostro Thread. Anche in questo caso il costrutto switch si occupa di richiamare il metodo contenente l'algoritmo di controllo selezionato.

```
protected void onHandleIntent(Intent intent) {

    //set Global variables
    ((MyApplication) this.getApplication()).setServiceRunning(true);
    ((MyApplication) this.getApplication()).setServiceStartTime(SystemClock.elapsedRealtime());

    //recupero l'algoritmo selezionato
    SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
    int algo = sharedPref.getInt("edit_algo", 0);

    //recupero l'orientamento
    landscape = sharedPref.getBoolean("edit_orientation", false);


    switch (algo) {
        case 0:
            onHandleIntentOnFoot(intent);
            break;
        case 1:
            onHandleIntent2Wheels(intent);
            break;
        case 2:
            onHandleIntent4Wheels(intent);
            break;
        case 3:
            onHandleIntentInFlight(intent);
            break;
    }

}
```

Al termine del monitoraggio similmente a sopra vengono distrutti i *listener* avviati, viene ripristinato la user interface ad uno stato iniziale (viene quindi azzerato il cronometro) e vengono fermati eventuali allarmi sonori avviati dall'applicazione:

```
@Override
public void onDestroy() {
    super.onDestroy();
    Log.i("SERVICE", "Distruzione Service");
    myThread.interrupt();
    ((MyApplication) this.getApplication()).setServiceRunning(false);
    updateMainActivityStatus("off", 0);
    updateNotificationStatus("off");

    //recupero l'algoritmo selezionato
    SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
    int algo = sharedPref.getInt("edit_algo", 0);

    switch (algo) {
        case 0:
            senSensorManager.unregisterListener(this, senAccelerometer);
            break;
        case 1:
            senSensorManager.unregisterListener(this, senRotationvector);
            break;
        case 2:
            senSensorManager.unregisterListener(this, senAccelerometer);
            this.getApplication().stopService(locationService);
            break;
        case 3:
            senSensorManager.unregisterListener(this, senPressure);
            senSensorManager.unregisterListener(this, senMagneticfield);
            break;
    }


    if(rt.isPlaying()) {
        rt.stop();
    }
    baseOrientationVals = null;
}
```

L'implementazione degli stessi listener è stata suddivisa in più metodi a seconda dell'algoritmo, più che altro per motivi di organizzazione e leggibilità del codice sorgente:

```
@Override
public void onSensorChanged(SensorEvent event) {

    SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
    int algo = sharedPref.getInt("edit_algo", 0);

    switch (algo) {
        case 0:
            onSensorChangedOnFoot(event);
            break;
        case 1:
            onSensorChanged2Wheels(event);
            break;
        case 2:
            onSensorChanged4Wheels(event);
            break;
        case 3:
            onSensorChangedInFlight(event);
            break;
    }


}
```

Le informazioni comuni al servizio come per esempio la data di avvio del servizio di monitoraggio e i metodi per aggiornare il cronometro nell'attività principale sono contenuti nella classe *Application* i cui metodi e attributi sono accessibili da tutte le classi dell'applicazione.

L'avvio e l'interruzione del servizio è comandato dall'utente stesso che può avviare il servizio di monitoraggio attraverso la *MainActivity* oppure attraverso la notifica creata dall'applicazione.

Se siamo nella schermata principale alla pressione del bottone Start viene richiamato il metodo *switchService* che si occupa di avviare o interrompere il servizio utilizzando un ricevitore broadcast:

```
public void switchService(View view) {

    if(((MyApplication) this.getApplication()).isServiceRunning()) {
        sendBroadcast(new Intent(this, StopServiceReceiver.class));
    } else {
        sendBroadcast(new Intent(this, StartServiceReceiver.class));
    }
}
```

Rispettivamente i *Broadcast Receiver* implementati per l'avvio e l'interruzione del servizio si chiamano *StopServiceReceiver* e *StartServeiceReceiver* e sono implementati nel modo seguente:

```
public class StartServiceReceiver extends BroadcastReceiver {

    static final public String INTENT_ID = "NeverAlone"+".mainactivityuiupdater";

    public StartServiceReceiver() {
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO: This method is called when the BroadcastReceiver is receiving
        // an Intent broadcast.
        Intent service = new Intent();
        service.setComponent(new ComponentName(context, MyService.class));
        context.startService(service);

        Toast.makeText(context, "Il servizio è stato avviato.", Toast.LENGTH_SHORT).show();
    }
}
```

```java
public class StopServiceReceiver extends BroadcastReceiver {

    static final public String INTENT_ID = "NeverAlone"+".mainactivityuiupdater";

    public StopServiceReceiver() {
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO: This method is called when the BroadcastReceiver is receiving
        // an Intent broadcast.
        Intent service = new Intent();
        service.setComponent(new ComponentName(context,MyService.class));
        context.stopService(service);

        Toast.makeText(context, "Il servizio è stato concluso.", Toast.LENGTH_SHORT).show();
    }
}
```

La differenza più importante tra il BroadcastReceiver di avvio e quello di interruzione è la chiamata del metodo startService o stopService.

4.7.1 Il suono di allerta

Il suono di allerta viene attivata in ogni algoritmo 30 secondi prima dell'invio dell'sms di aiuto. Il controllo viene fatto calcolando i secondi trascorsi dall'ultimo stato di sicurezza.

```java
//se i suoni sono abilitati
if(alertSoundEnabled) {
    //se il tempo non è scaduto e se a breve sta per scadere il tempo
    if(
        !timeOut(lastSecurityStatusTime, paniclimitMillis, 0) &&
        timeOut(lastSecurityStatusTime, paniclimitMillis, SOUNDALERT_THRESHOLD)
    ) {

        //riproduco il suono di allarme
        if (!rt.isPlaying()) {
            rt.play();
        }

    } else {

        //interrompo il suono di allarme
        if (rt.isPlaying()) {
            rt.stop();
        }
    }
}
```

Con il metodo *timeOut* viene verificato se il tempo è scaduto e se mancano meno di 30 secondi all'invio dell'sms.

4.7.2 Invio dell'sms di aiuto

L'invio dell'sms di aiuto avviene attraverso una chiamata al servizio broadcast *AlertServiceReceiver* quando il tempo trascorso dall'ultimo stato di sicurezza supera i 30 secondi, a questo punto inoltre è possibile interrompere il servizio di monitoraggio che ha terminato il suo compito:

```
//se è passato troppo tempo dall'ultimo movimento spedisco il messaggio di aiuto
if(timeOut(lastSecurityStatusTime, paniclimitMillis, 0)) {

    Log.i("INVIO SMS AIUTO", "...");
    sendBroadcast(new Intent(this, AlertServiceReceiver.class));
    this.stopSelf();
    break;

}
```

Il *broadcast receiver* per l'invio dell'sms di aiuto implementa al suo interno le funzioni di localizzazione e richiama i metodi per la creazione e l'invio dell'sms:

```
@Override
public void onReceive(final Context context, Intent intent) {

    //recupero le preferenze di Allerta
    sharedPref = PreferenceManager.getDefaultSharedPreferences(context);
    edit_phonepanic_sp = sharedPref.getString("edit_phonepanic", null);
    edit_messagepanic_sp = sharedPref.getString("edit_messagepanic", null);


    // Acquire a reference to the system Location Manager
    locationManager = (LocationManager) context.getSystemService(Context.LOCATION_SERVICE);

    //set criteria for best location provider
    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_FINE);
    String provider = locationManager.getBestProvider(criteria, true);

    // Define a listener that responds to location updates
    LocationListener locationListener = new LocationListener() {
        public void onLocationChanged(Location location) {
            // Called when a new location is found by the network location provider.
            Log.i("LOCATION LISTENER", location.toString());
            locationResult.gotLocation(location);
            locationManager.removeUpdates(this);
        }

        public void onStatusChanged(String provider, int status, Bundle extras) {}

        public void onProviderEnabled(String provider) {}

        public void onProviderDisabled(String provider) {}


    };
```

Solo quando la posizione viene trovata allora vengono eseguiti i metodi per l'invio dell'SMS di aiuto.

```
//riceve la posizione dal listener
locationResult = (LocationResult) (location) → {
        //Got the location!
        //faccio qualcosa con la posizione rilevata.
        Log.i("LOCATION RESULT", location.toString());

        //se ho un numero di telefono spedisco l'sms
        if(edit_phonepanic_sp != null) {

            try {
                SmsManager smsManager = SmsManager.getDefault();
                smsManager.sendTextMessage(
                        edit_phonepanic_sp.toString(),
                        null,
                        edit_messagepanic_sp.toString()+": maps.google.com/maps?q=" + location.getLatitude() + "," + location.getLongitude(),
                        null,
                        null);
                Toast.makeText(context, "Messaggio di aiuto spedito!", Toast.LENGTH_SHORT).show();
            } catch (Exception e) {
                Log.i("Exception", e.toString());
            }

        }


};
```

4.7.3 Aggiornamento della UI

L'aggiornamento della user interface, nello specifico la schermata principale ed il messaggio di notifica, avviene per mezzo di due metodi, rispettivamente per la schermata principale e per la notifica.

```
updateMainActivityStatus("off", 0);
updateNotificationStatus("off");
```

L'aggiornamento della schermata principale avviene mediante il sistema di invio delle informazioni *Extra* che è possibile includere in una chiamata *Broadcast*.

```
private void updateMainActivityStatus(String status, long startTime) {

    LocalBroadcastManager broadcaster = LocalBroadcastManager.getInstance(this);
    Intent intentMainActivity = new Intent(INTENT_ID);

    if(status.equals("on")) {
        intentMainActivity.putExtra("status", "on");
        intentMainActivity.putExtra("startTime", startTime);
    } else {
        intentMainActivity.putExtra("status", "off");
    }

    broadcaster.sendBroadcast(intentMainActivity);
}
```

La notifica di sistema viene aggiornata invece ridefinendo la notifica stessa con le informazioni aggiornate:

```
mNotifyBuilder = new Notification.Builder(this)
        .setOngoing(true)
        .setShowWhen(false)
        .setSmallIcon(R.drawable.ic_actionbar_icon)
        .setContentIntent(pIntent);

mNotifyBuilder = mNotifyBuilder.setContentTitle("NeverAlone");


if(status.equals("on")) {
    mNotifyBuilder = mNotifyBuilder
            .addAction(R.drawable.ic_stop, "STOP", pIntentStopService)
            .setContentText("Sentinella in ascolto...");
} else {
    mNotifyBuilder = mNotifyBuilder
            .addAction(R.drawable.ic_play, "START", pIntentPlayService)
            .setContentText("Premere START per cominciare.");
}

mNotifyBuilder = mNotifyBuilder
        .addAction(R.drawable.ic_send, "AIUTO", pintentAlertService);

this.notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

this.notificationManager.notify(NOTIFICATION, mNotifyBuilder.build());
```

4.8 Schermate

In questo paragrafo vengono illustrate le schermate presenti evidenziandone le implementazioni salienti.

4.8.1 Splash Screen

All'avvio dell'applicazione viene eseguita una schermata denominata *SplashScreen*. L'unico compito di questa schermata è quella di stampare a video il logo ed il nome dell'applicazione. Questa schermata non ha una vera e propria utilità, ma per motivi di marketing è utile ad impressionare il logo nella mente dell'utente.

L'implementazione di questa schermata prevede la visualizzazione dell'immagine attraverso il widget *ImageView* che rimane visualizzato sullo schermo per un tempo di circa 1,5 secondi.

Per implementare questo ritardo è stato utilizzato un *Thread* che viene messo a riposo per il tempo necessario attraverso il metodo *Thread.sleep*

4.8.2 Monitor

La schermata Monitor è basata su di un layout lineare *LinearLayout* che contiene al suo interno delle *TextView* per la visualizzazione dei contenuti visuali e di un ulteriore layout lineare interno a sua volta composto da una *TextView* ed un *Chronometer* che rappresenta il bottone principale. Il design del bottone è creato utilizzando un background personalizzato definito all'interno della risorsa shape.xml che contiene un oggetto designabile di tipo *ripple.* Questo oggetto consente di ottenere un feedback in risposta al tocco dell'utente sul bottone, perfettamente in linea con le linee guida sul Material Design.

4.8.3 Mappa

Il supporto alla mappa è dato da una serie di componenti che vengono importati all'interno del progetto:

```
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.model.LatLng;
```

Queste librerie consentono di definire una mappa all'interno di un Fragment completa di controlli per la navigazione all'interno della mappa.

4.8.4 Posizione

La schermata posizione è composta essenzialmente da *TextView* che servono alla visualizzazione delle coordinate. Il sistema utilizzato per la localizzazione del dispositivo è lo stesso utilizzato nell'algoritmo di rilevazione degli incidenti.

Una funzionalità interessante inserita all'interno di questa schermata è la possibilità di copiare la posizione corrente negli appunti del dispositivo, per mezzo del componente *ClipboardManager,* così da utilizzarla su altre applicazioni.

Scrivere la posizione all'interno del clipboard è molto semplice:

```
ClipboardManager clipboard =
        (ClipboardManager) getActivity().getSystemService(getActivity().CLIPBOARD_SERVICE);
clipboard.setText("Lat: "+latvalue.getText()+", Lng: "+lngvalue.getText());
```

4.8.5 Configurazione

Oltre agli algoritmi, l'applicazione dispone anche della schermata di configurazione. La schermata è stata costruita utilizzando i template xml ed è composta da un contenitore esterno scrollabile denominato *ScrollView* e da un contenitore più interno del tipo *LinearLayout.* Al suo interno sono allocati tutti i widget che servono all'inserimento dei dati. Il layout a due colonne con le etichette a sinistra e i campi di input a destra sono stati creati attraverso un layout tabellare *TableLayout.*

Una funzionalità interessante introdotta all'interno di questa schermata è la possibilità di selezionare il contatto a cui scrivere in caso di aiuto direttamente scegliendolo dalla rubrica del telefono. Questa caratteristica la si ottiene inizializzando un Listener sull'icona desiderata che al click richiama attraverso un Intent la schermata di selezione dalla rubrica in base alle definizioni richieste *ContactsContract.CommonDataKinds.* Lo specifico contatto selezionando è identificabile richiamando il metodo *getColumnIndex* per ottenere l'indice ed utilizzando all'interno della lista restituita dal metodo *getContentResolver().query().*

```
Uri contactUri = data.getData();
Cursor cursor = getContentResolver().query(contactUri, null, null, null, null);
cursor.moveToFirst();
int column;

switch (requestCode) {

    case PICK_CONTACT_PHONE:

        column = cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER);
        final EditText phoneEditText = (EditText) findViewById(R.id.edit_phonepanic);
        phoneEditText.setText(cursor.getString(column));
        break;

    case PICK_CONTACT_EMAIL:

        column = cursor.getColumnIndex(ContactsContract.CommonDataKinds.Email.ADDRESS);
        final EditText emailEditText = (EditText) findViewById(R.id.edit_emailpanic);
        emailEditText.setText(cursor.getString(column));
        break;

}
```

Un altra particolarità della schermata di configurazione è che non c'è una vera call to action per il salvataggio dei dati, ma la memorizzazione nelle SharedPreferences è invocata alla chiusura dell'Activity.

4.9 Testing

Con la fase di *unit testing* è stata testata ogni singola funzionalità dell'applicazione. L'obiettivo era quello di verificarne la compatibilità dalla versione 4.1.2 ovvero dalle API 16. Per questi dispositivi meno recenti ho utilizzato i device virtuali contenuti nell'l'SDK di Android Studio.

Per le prove reali sul campo si è utilizzato il mio device personale (Android Nexus 4 con Lollipop).

Tutti gli algoritmi ad eccezione dell'algoritmo In volo sono stati adeguatamente testati sul campo. Con l'avvento della stagione estiva si avrà modo di testare anche quest'ultimo algoritmo.

CAPITOLO 5
Conclusioni

L'applicazione *NeverAlone per Android* L'applicazione sviluppata ha rispettato quasi tutte le specifiche prefissate. Difficoltà implementative non hanno permesso la realizzazione di alcune funzionalità utili in questa prima versione. Funzionalità che saranno implementate negli sviluppi futuri.

5.1 Sviluppi Futuri

L'App sviluppata è efficace e distribuibile, però per essere realmente appetibile deve implementare alcune funzionalità in più. Alcune delle caratteristiche che la renderebbero più duttile riguardano i canali di spedizione:

Si potrebbero implementare altri canali oltre agli SMS. Per esempio l'e-mail (purché la configurazione del client di posta in uscita sia semplice ed immediata) e le notifiche push ad altri utenti nei paraggi.

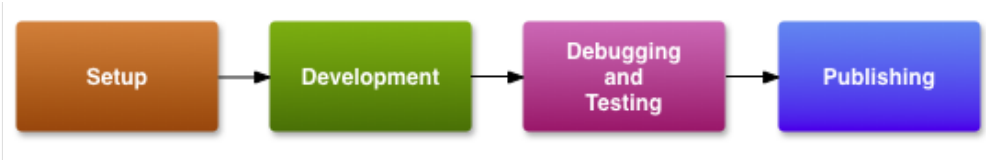
Un'altra caratteristica interessante è la possibilità di inserire un elenco di utenti fidati piuttosto che solamente uno.

Ci sarebbero inoltre una miriade di caratteristiche migliorabili, come per esempio un bottone "Falso Allarme" nell'interfaccia per scongiurare l'invio di un falso positivo senza interrompere il monitoraggio.

5.2 Processo di pubblicazione

Il processo di pubblicazione di un applicazione Android prevede l'esecuzione di una checklist[24] che consiste in una serie di controlli da effettuare prima della pubblicazione.

Prima di tutto bisogna conoscere come funziona il processo di pubblicazione, ovvero quali sono le macro fasi che compongono la produzione di un applicazione:



Dopo le prime fasi di configurazione ambiente e sviluppo vero e proprio si procede alla fase di debugging e testing, solo dopo avere superato con successo questa fase è possibile procedere con il vero e proprio processo di pubblicazione.

Nella fase di pubblicazione si può scegliere dove ed in che modo distribuire un'applicazione:

- *Pubblicare sul Marketplace di Google:* Questa è la soluzione ottimale se si vuole distribuire l'applicazione al maggior numero di persone.
- *Pubblicare su altro Marketplace*: anche questa soluzione permette di raggiungere molte persone, il marketplace di Amazon è una buona alternativa a Google Play.
- *Distribuire via email*: è possibile anche distribuire l'applicazione via email, sarà sufficiente allegare la release al messaggio e recapitarla ad un contatto. Quando questo messaggio verrà aperto da un dispositivo Android, con permessi di installazione di App provenienti da origini sconosciute, automaticamente comparirà un bottone per eseguire l'installazione.
- *Distribuire via web*: è possibile anche distribuire l'applicazione attraverso un sito web, consentendo semplicemente il download della release. Anche in questo caso un dispositivo android con permessi di installazione di App provenienti da origini sconosciute sarà in grado di eseguire l'installazione.

Se si sceglie di distribuire l'applicazione su Google Play bisognerà inserire una serie di dati che consentiranno una perfetta indicizzazione e categorizzazione dell'App all'interno del marketplace.

In fase di pubblicazione sarà necessario specificare una *valutazione sui contenuti*, ovvero se l'Applicazione è adatta ad un solo pubblico adulto o a tutti. I livelli tra cui scegliere sono i seguenti:

- Everyone

- Low maturity
- Medium maturity
- High maturity

Si dovrà anche scegliere in quasi paesi distribuire l'App. Il valore di default è in tutto il mondo.

Un limite per la pubblicazione di un App è inoltre la sua dimensione: non può superare i 100 MegaByte, in caso contrario sarà necessario usare *APK Expansion files* che consente di spezzettare la release in più parti.

Un'altro step consiste nel confermare le piattaforme su cui distribuire (le versioni del sistema operativo) e le dimensioni di schermi in modo tale da impedire la distribuzione su piattaforme che non sono compatibili.

Procedendo con la checklist viene anche richiesto il prezzo che dovrà avere l'applicazione:

Considerando i prezzi di vendita e le funzionalità offerte dai principali concorrenti si ritiene che il prezzo di vendita dell'App sviluppata in questa tesi dovrà essere compreso tra i 3 e i 10 €.

5.3 Rating & Download

Voto medio e numero di download totali sono i due indicatori principali utilizzati all'interno del marketplace per valutare la qualità di un applicazione. Tanto più questi due numeri sono elevati maggiore sarà la rilevanza e la visibilità dell'applicazione all'interno del negozio virtuale.

La qualità generale dell'applicazione e l'assenza di bug saranno sicuramente le caratteristiche fondamentali che indirizzeranno nella giusta direzione le valutazioni. Bisogna però tenere in considerazione comunque alcuni dettagli per evitare incomprensioni con gli utenti. Prima di tutto sarà necessaria una descrizione e categorizzazione adeguata all'interno del marketplace, in modo tale da non lasciare spazio ad incomprensioni che potrebbero degenerare in valutazioni negative.

Si ritiene inoltre sia sconsigliato proporre l'applicazione gratuitamente perché pubblicando l'applicazione ed offrendola a costo zero si rischia di attirare l'attenzione dei curiosi oltre che quella degli utenti effettivamente interessati. Non sarebbe una cosa negativa se non fosse che tutti possono valutare l'applicazione su Google Play Store una volta installata. Il *rating* è un arma a doppio taglio, infatti non esiste un vero e proprio sistema di moderazione. Ognuno può dire la sua. Per esempio se un utente si aspetta una funzionalità che invece non è presente tende a valutare l'App negativamente. Lo stesso succede se un utente non ha capito l'effettivo utilizzo dell'applicazione. Curiosamente spesso capita che il rating venga utilizzato come arma di ricatto con lo sviluppatore (quante volte abbiamo letto la frase "5 stelle se inserite questa funzionalità..."). Infine il grado di istruzione di chi commenta incide notevolmente sulla valutazione finale.

Per sensibilizzare un attenzione maggiore a ciò che gli utenti installano sul proprio device ritengo sia consigliabile proporre l'applicazione ad un prezzo modico in modo tale da attirare solamente gli utenti realmente interessati che acquisteranno l'applicazione solamente dopo essersi adeguatamente informati.

5.4 Materiale Grafico

NeverAlone ed il suo logo sono di proprietà di Marco Casadio.

Ringraziamenti

Colgo l'occasione per ringraziare tutti coloro che mi hanno supportato e incoraggiato in questo percorso accademico. Un ringraziamento particolare va alla mia famiglia a cui dedico questo importante momento e alla mia compagna di vita Laura che mi ha incoraggiato a non abbandonare ciò che avevo costruito fino ad oggi. Con lei ringrazio anche la sua famiglia che ha contribuito alla mia crescita personale. Ringrazio inoltre i miei colleghi di lavoro con cui ho condiviso momenti di stress ma soprattutto che hanno contribuito in maniera decisiva alla mia crescita professionale. Un ringraziamento particolare spetta al mio collega Marco Casadio ideatore del marchio *NeverAlone* a cui si ispira questa tesi e che mi ha trasmesso l'interesse per questo ambito. Ringrazio poi compagni di studio ed amici vicini e lontani. Infine, ma non meno importanti, i miei ringraziamenti vanno all'Università di Bologna, a questo ateneo e al dott. Mirko Ravaioli relatore di questa tesi per la sua disponibilità.

Riferimenti Bibliografici

[1] http://marco.casadio.me/

[2] https://itunes.apple.com/it/app/neveralone+/id797694169?mt=8

[3] http://blogs.cisco.com/ar/updated-cisco-mobile-visual-networking-index

[4] http://www.slideshare.net/wearesocialit/social-digital-mobile-in-europa-2014

[5] http://www.unifi.it/

[6] kantar.com http://www.panorama.it/mytech/smartphone-tablet/windows-phone-android-ios-quote-mercato-italia/

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

[8] https://developer.android.com/about/dashboards/index.html

[9] http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[10] https://play.google.com/store

[11] http://runsafe.me/

[12] http://getbsafe.com/

[13] http://www.siamosicure.it/

[14] http://www.protechtor.it/

[15] https://itunes.apple.com/it/app/panik-personal-assault-alarm/id588969126?mt=8

[16] http://explore.garmin.com/it-IT/edge/

[17] https://www.lucidchart.com/

[18] https://moqups.com/

[19] https://trello.com/

[20] https://www.google.com/design/spec/material-design/introduction.html

[21] https://www.google.com/design/spec/style/color.html

[22] https://www.google.com/design/icons/

[23] https://developer.android.com/sdk/index.html

[24] http://developer.android.com/distribute/tools/launch-checklist.html