UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Corso di Laurea in Informatica

TESI DI LAUREA TRIENNALE

Sviluppo di un'applicazione Android per il tracking outdoor di persone con disabilità cognitive

RELATORE

Prof. Francesco Tarquini

LAUREANDO

Riccardo Armando Di Prinzio
Matr. 229032

Anno Accademico 2017-2018

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il lavoro di tesi si occupa di un sottoinsieme di un progetto più grande, il progetto Casa Futura, che verrà descritto più avanti.
La tesi verte sullo sviluppo di un'applicazione Android che ha il principale compito di geolocalizzare [1] gli utenti utilizzatori ed agire in base alla loro posizione. In particolare, controllare se esso si trova in una particolare area inscritta, denominata "area sicura".

L'applicativo è associato all'utente e pertanto fa uso di due possibili tipi di Login. Uno è semplicemente il metodo tradizionale, ovvero dei campi da riempire con i dati dell'utente registrato; se il login va a buon fine, i successivi vengono automatizzati per facilitare la procedura.
L'altro metodo utilizza le impronte digitali dell'utente in modo da semplificare ulteriormente il processo.

Fin da subito l'utente può scegliere due modalità d'uso dell'applicazione. La prima di esse è la navigazione libera, nella quale l'app mostra una mappa Google Maps [2] con un segnalatore che corrisponde alla posizione dell'utente.
In questa modalità l'utente può sapere dove si trova in qualsiasi momento.
Inoltre l'app reagirà se l'utente si trova nell'area sicura oppure esce da essa;
se l'utente rimane nell'area sicura l'app aggiorna la mappa con i suoi spostamenti;
invece se l'utente esce dall'area sicura, tempestivamente l'applicazione crea un popup per allertarlo con la possibilità di chiamare oppure no l'educatore, e in caso di assenza di interazioni entro 15 secondi, viene intrapresa una chiamata automatica ad esso.

In base ad una preferenza configurabile da un menù contestuale, è possibile inoltre far inviare automaticamente un SMS all'educatore contenente il link
Google Maps associato alla posizione dell'utente, in modo da facilitarne i compiti.

Nell'altra modalità d'uso, la navigazione per immagini, è possibile consultare una lista di percorsi associati all'utente, e in base alla sua scelta, si caricheranno l'immagine, il nome e la descrizione dei punti di interesse (POI) ad esso associati che guideranno l'utente durante il percorso.
Tali punti di interesse verranno uno dopo l'altro mostrati per guidare l'utente verso il più prossimo POI da raggiungere sul percorso specificato. Il percorso ha un tempo limite di percorrenza, alla sua scadenza e dopo di un lasso di tempo configurabile, l'utente viene allertato da un popup con dinamiche del tutto simili alla precedente modalità d'uso.

L'applicazione si poggia su un lato server preesistente a cui si accede mediante chiamate ad API specifiche per scambiarsi informazioni.
I dati scambiati coinvolgono lo stato dell'utente (l'utente utilizzatore si trova nell'area sicura oppure no), i percorsi ad esso associati, le richieste di download di immagini di punti di interesse e le richieste di upload di immagini che l'utente stesso può scattare e che potrebbero essere scelti come immagine di un punto di interesse.

Funzionalità aggiuntiva dell'applicativo è la possibilità per l'educatore di "risvegliare" in qualsiasi momento l'app da remoto e quindi consentire la geolocalizzazione dell'utente. Lo scopo è di controllare in qualsiasi momento se esso è nell'area sicura oppure no.
Tale funzionalità ha reso necessario lo sviluppo di un server con un'apposita pagina web che permette all'educatore di scegliere quale utente "risvegliare" oppure agire su tutti gli utenti.

Capitolo 2

Descrizione del progetto

La missione dell'ente *Enel Cuore Onlus* è di rappresentare un punto di riferimento costante per le persone con sindrome di Down ed altre disabilità cognitive e per assicurare loro una buona qualità di vita, anche in età adulta, ed una adeguata soluzione residenziale.
I suoi obiettivi principali sono di realizzare case-famiglia per persone con sindrome di Down ed altre disabilità, in quartieri ben collegati con il territorio in modo da fornire facili opportunità di spostamento e di integrazione in rete con servizi pubblici e privati.

2.1 Casa Futura

Tra le varie case-famiglia, c'è Casa Futura; una casa Famiglia a bassa assistenza e a gestione privata, che ospita 6 ragazzi con sindrome di Down abbastanza autonomi. Tale casa è localizzata a Roma.
L'applicazione e l'utilizzo della piattaforma domotica permetterebbe, oltre che di aumentare la sicurezza e le autonomie personali degli ospiti, anche di svincolare risorse operative attualmente dedicate a sollecitare verbalmente gli ospiti al rispetto dei tempi o ad interventi sulle autonomie che possono essere supportate dalle applicazioni previste nel progetto di domotizzazione.
Un ruolo fondamentale nel progetto è svolto dal Centro di eccellenza Dews della Facoltà di Ingegneria dell'Università dell'Aquila, che ha curato lo studio delle componenti domotiche idonee allo sviluppo dell'autonomia, ed alla tutela della persona "fragile" nello svolgimento e nella gestione delle attività quotidiane.
Le suddette risorse tecnologiche non sono ipotizzate come sostitutive della relazione educativa e della relazione umana con i caregiver, ma come integrazione e supporto alla presenza di questi.

2.2 Motivazioni

L'utilizzo delle tecnologie previste permetterebbe la realizzazione e l'implementazione di attività quali la permanenza in casa da soli in alcune ore della giornata, senza mettere a rischio la sicurezza degli ospiti:

- L'abitazione verrebbe dotata di supporti tecnologici, utili a sviluppare le capacità organizzative della persona ed a sollecitare una sua risposta attiva alla segnalazione di anomalie, allarmi o ritardi nell'adempimento di attività prefissate.
- Nell'abitazione ci sarà il controllo a distanza di luce, gas e riscaldamento.
- La possibilità di interventi vocali a distanza finalizzati alla sollecitazione della tempistica o ad una guida verbale in alcune situazioni di gestione della casa e delle autonomie interne o esterne.
- Localizzazione della posizione e degli spostamenti, degli abitanti nella struttura.
- Accessibilità a Video tutoriali per la preparazione dei pasti e per le autonomie nella gestione della casa.
- Programma informatizzato per la gestione della spesa in base al controllo dei prodotti assenti o presenti in casa, anche attraverso la possibilità di aggiornamento a distanza dello smartphone personale in dotazione alla persona disabile.
- Definizione e controllo attraverso uno smartphone, di un'area sicura negli spostamenti esterni, oltrepassata la quale la persona viene avvertita al cellulare o in caso di mancata risposta, attraverso i numeri programmati.

Permette infine di mantenere alta la propria autostima nell'ambito di una vita comunque autonoma e sufficientemente indipendente.

2.3 Obiettivi

Il progetto ha lo scopo di:

- Promuovere la vita indipendente delle persone con sidrome di Down attraverso percorsi verso la residenzialità che siano sostenibili dal punto di vista sociale, tecnologico, ambientale e finanziario.
- Sensibilizzare e accompagnare le famiglie.
- Consolidare il loro vivere insieme con il maggior grado di autonomia possibile.

- Per gli operatori/volontari e le famiglie il progetto ha il fine di far acquisire consapevolezza sulle potenzialità e i diritti delle persone con sidrome di Down.

- Per la FIVF (Fondazione Italiana Verso il Futuro Onlus) ha lo scopo di acquisire un modello replicabile in tutta Italia.

2.4 Risultati attesi

- Maggiore autonomia: Una maggiore autonomia delle persone con sidrome di Down coinvolte porterà ad una maggiore inclusione nella comunità.
  L'uso delle tecnologie accompagnerà la conquista di una maggiore autonomia ed un più alto livello di sicurezza nelle case.

- Replicabilità: Il progetto potrebbe costituire una sperimentazione destinata alla replicabilità in strutture residenziali a bassa/media assistenza sia per disabili che per anziani, potrebbe essere destinato anche a quelle categorie di persone "fragili" per le quali però il mantenimento delle proprie autonomie ricopre una vera e propria valenza terapeutica.

- Impatto sociale: L'utilizzo della domotica in un progetto di residenzialità permanente può aprire la strada ad una nuova prospettiva.
  Se è vero che le strutture a bassa assistenza non possono costituire una risposta destinata a tutte le persone disabili, è altrettanto vero che possono accogliere le persone che hanno ancora la possibilità di implementare le proprie autonomie, migliorando la propria autostima e la qualità di vita.
  La visibilità che tale progetto può avere attraverso i social network ed i mass media, aiuterebbe ad indurre un cambiamento culturale necessario sia nell'ambito delle famiglie delle persone disabili, che nella società.

Capitolo 3

Android

Android è un sistema operativo per dispositivi mobili sviluppato da Google Inc. e basato sul kernel Linux.
È un sistema embedded progettato principalmente per smartphone e tablet, con interfacce utente specializzate per televisori (Android TV), automobili (Android Auto), orologi da polso (Android Wear), occhiali (Google Glass), ed altri.
Android adotta una politica di licenza di tipo open source. La licenza (Licenza Apache) sotto cui è distribuito consente di modificare e distribuire liberamente il codice sorgente.
Inoltre, Android dispone di una vasta comunità di sviluppatori che realizzano applicazioni con l'obiettivo di aumentare le funzionalità dei dispositivi. Queste applicazioni sono scritte soprattutto in linguaggio di programmazione Java.

3.1 Architettura

Android è basato su un architettura a layer dove ogni livello può richiedere servizi al livello sottostante e ne fornisce al livello superiore. In figura 3.1 si può osservare uno schema che riassume graficamente i layer che compongono lo stack.

Le principali caratteristiche di ogni layer sono le seguenti:

- Linux Kernel: rappresenta la base, lo strato più basso, della piattaforma Android. Contiene l'implementazione dei vari driver che permettono l'interazione con l'hardware utilizzato dall'OS.
- Hardware Abstraction Layer (HAL): questo livello fornisce interfacce standard al livello più alto Java API Framework, per poter accedere alle funzionalità hardware del dispositivo. L'HAL è composto da vari moduli, ognuno dei quali implementa un'interfaccia per uno specifico componente hardware.

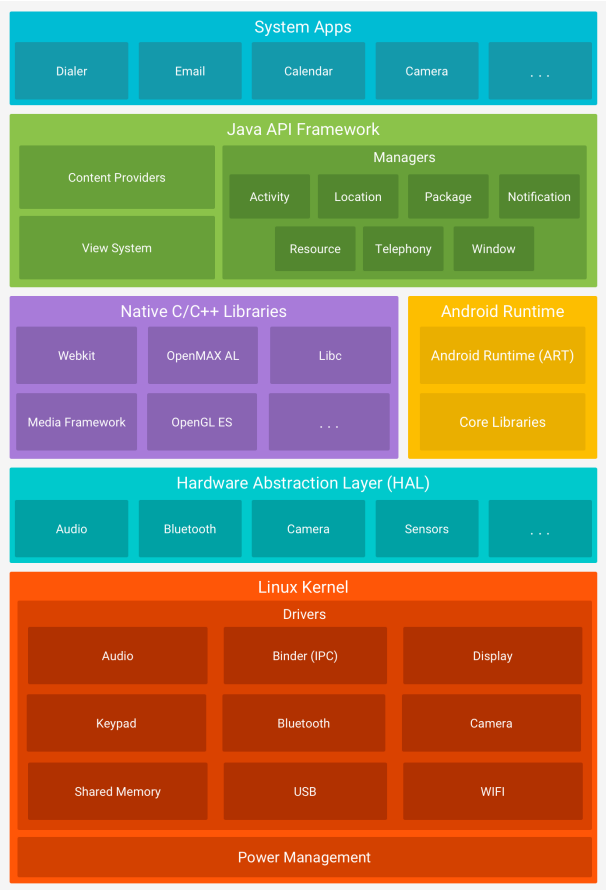

Figura 3.1: Architettura stratificata di Android

- Android Runtime (ART): è il nuovo runtime system introdotto con Android 4.4 KitKat e che ha sostituito poi definitivamente la virtual machine Dalvik. Mentre prima le app venivano compilate in parte dallo sviluppatore, per poi ogni volta essere eseguite e compilate definitivamente da Dalvik in tempo reale, grazie ad ART l'intera compilazione del codice avviene durante l'installazione dell'app e non durante l'esecuzione della stessa. Ciò ha portato ad un notevole guadagno in termini di prestazioni e gestione di risorse.

- Native C/C++ Libraries: si tratta di componenti implementati per lo più in codice nativo, e quindi C/C++, ma che espongono delle interfacce Java per l'interazione con servizi classici di un dispositivo mobile.

- Java API Framework: questo livello sfrutta i servizi forniti da ART e dalle Native Libreries e mette a disposizione, tramite API scritte in Java, l'intero set di funzionalità del sistema operativo Android. Tali API costituiscono i blocchi necessari per la costruzione delle applicazioni Android.

- System Apps: Android è dotato di una serie di applicazioni principali per la gestione di email, messaggistica SMS, calendari, navigazione internet, contatti ed altro.
  Le applicazioni di sistema funzionano sia come app comuni per l'utente finale e sia per fornire funzionalità chiave agli sviluppatori, a cui gli stessi possono accedere tramite una propria app sviluppata.
  Ad esempio se si vuole consegnare un messaggio SMS non è necessario creare le funzionalità di base, ma basta invocare l'applicazione per la gestione degli SMS già installata.

3.2 Applicazioni

Le applicazioni (o app) sono la forma più generica per indicare i software applicativi installabili su Android. Esse possono essere scaricate sia dal catalogo ufficiale Google Play, sia da altri cataloghi.
Le applicazioni Android possono anche essere installate direttamente a partire da un file APK fornito dal distributore del software.
APK è l'acronimo di Android Application Package, sono dei file archivio al cui interno è possibile trovare tutti i file e gli elementi necessari all'installazione e al funzionamento dell'app che si è scelta.
Il certificato deve essere presente in qualsiasi pacchetto, altrimenti Android non installerà l'applicazione al suo interno.

3.3 Componenti principali

Android mette a disposizione una serie di componenti tramite cui è possibile sviluppare le applicazioni.
Si riportano di seguito dunque i principali componenti che compongono la piattaforma Android [3].

3.3.1 Activity

Le Activity sono quelle classi scritte in linguaggio Java che compongono un'applicazione Android e subiscono un'interazione diretta con l'utente.
All'avvio di ogni applicazione viene eseguita una Activity, la quale può eseguire delle operazioni e può anche aprire/eseguire altre Activity.
Le Activity create estendono la classe Activity da cui ereditano proprietà e metodi.
La visualizzazione di una serie di schermate è organizzata in una struttura a stack, ovvero una pila di tipo FIFO (first-in-first-out), in cui l'Activity più in alto corrisponde a quella visibile sullo schermo del dispositivo Android.

L'Activity subisce una serie di variazioni di stato, dal momento in cui viene eseguita fino a quando viene mandata in background ed infine chiusa: il ciclo di vita di un'Activity, il cui flusso è riportato in figura 3.2.

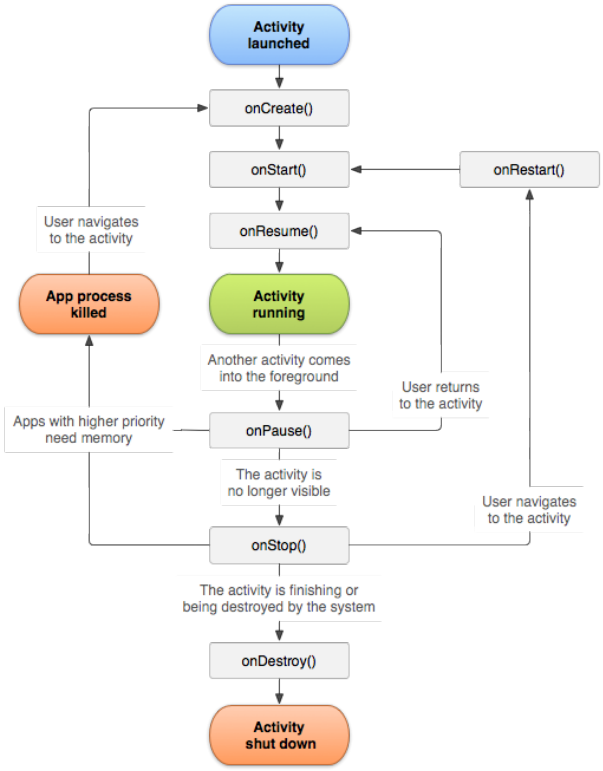

Figura 3.2: Ciclo di vita di una activity

Per gestire la transizione tra uno stato e l'altro vengono utilizzata una serie di callbacks.
Vediamo di seguito alcuni di questi, visibili anche in figura:

- onCreate(): è necessario implementare questo callback, che viene richiamato quando il sistema crea l'Activity in questione. Al suo interno dovrebbero essere inizializzati i componenti essenziali dell'Activity, ma soprattutto questo è il punto in cui è necessario chiamare il metodo setContentView() per definire il layout dell'interfaccia relativa all'Activity stessa.
- onStart(): l'Activity diventa visibile all'utente.
- onResume(): il sistema invoca questo callback prima che l'Activity inizi ad interagire con l'utente. A questo punto l'Activity è in cima allo stack e cattura tutti gli input dell'utente. Buona parte delle funzionalità dell'app sono implementate qui.

- onPause(): il sistema richiama questo callback quando l'Activity viene spostata in basso nello stack, per esempio in corrispondenza della pressione del pulsante di sistema Back da parte dell'utente. In questo metodo non vanno implementate funzionalità che eseguano richieste a risorse di rete o chiamate a database. Una volta terminata l'esecuzione di onPause(), la successiva chiamata può essere onStop() o onResume().

- onStop(): il sistema richiama questo callback quando l'Activity non è più visibile all'utente. Ciò può accadere sia perché l'Activity è stata distrutta o perché una nuova sta iniziando ad esempio.

- onRestart(): il sistema invoca questo callback quando un'Activity nello stato Stopped sta per riiniziare. Viene quindi ripristinato lo stato dell'Activity dal momento in cui era stato stoppato.

- onDestroy(): il sistema invoca questo callback prima che l'Activity sia distrutta. In genere qui vengono rilasciate tutte le risorse occupate precedentemente dall'Activity.

Ciascuna schermata definisce principalmente due aspetti: l'insieme degli elementi grafici e la modalità di interazione con essi.
Ciascun elemento grafico verrà descritto da particolari specializzazioni della classe View, che vengono posizionate sullo schermo secondo determinate regole di layout, definite principalmente attraverso opportuni files XML di layout.
Una Activity avrà quindi la responsabilità di gestione dei componenti della UI (User Interface) e delle interazioni con i servizi di gestione dei dati.

3.3.2 Service

I servizi sono quelle applicazioni che per loro natura svolgono delle operazioni autonome, senza che l'utente debba interagire con esso, e che vengono richiamati dalle Activity al bisogno.
Il sistema operativo fornisce alle applicazioni vari servizi già pronti all'uso, per ottenere l'accesso all'hardware o a risorse esterne.
I servizi sono oggetti di classe Services.
Vi sono una serie di mansioni che è consigliabile affidare ai servizi, come ad esempio chiamate alla rete internet.
Questi componenti, come altri, vengono eseguiti nel thread principale del loro processo di hosting, sebbene sia possibile comunque realizzare servizi che sopravvivono alla chiusura dell'applicazione.
Vi sono due tipologie di Service:

- unbounded: il Service viene avviato con il metodo startService() e viene eseguito in background per un tempo indefinito anche se il componente che li ha avviati viene terminato.

Possono essere interrotti con il metodo stopService() o si interrompono autonomamente con il metodo stopSelf().

- bounded: vengono eseguiti nel momento in cui un client ne richiede il servizio ed interrotti nel momento in cui terminano di servire il client.

Per implementare un Service è necessario realizzare una classe che estenda la classe astratta Service, aggiungere il servizio all'interno del Manifest ed infine avviare il servizio.
Una specializzazione della classe Service è IntentService, sottoclasse di Service, che viene utilizzata per eseguire determinate attività in background.
Mentre la classe Service utilizza il thread principale dell'applicazione, IntentService crea un thread di lavoro e utilizza quel thread per eseguire il servizio; al termine di esso, l'istanza di IntentService termina automaticamente.
Esso crea una coda che passa un Intent alla volta al callback onHandleIntent().
La classe Service richiede un arresto manuale con stopSelf(), mentre IntentService si arresta automaticamente quando termina l'esecuzione.

3.3.3 Broadcast Receiver

I Broadcast receivers permettono alle app di ricevere segnali rivolti a tutte le app in esecuzione, per la condivisione di dati o di segnali di servizio.
Per l'implementazione di uno di essi occorre creare una classe che estende BroadcastReceiver e implementare il metodo onReceive();

3.3.4 Fragment

Il Fragment è quella porzione di codice che gestisce la parte grafica, in base alle possibilità del dispositivo su cui è stato installato.
Una classe generica che permette lo sviluppo di un'applicazione con la parte grafica slegata da quella "decisionale", in modo da rendere agevole l'adattamento dell'applicazione alle varie situazioni.
Il programmatore creerà vari frammenti della parte grafica e poi Android la ridisegnerà correttamente per il dispositivo in uso.
I Fragment consistono in una porzione di UI, all'interno di una singola Activity. Si può pensare infatti ad essi come una sezione modulare di un Activity, con un proprio ciclo di vita, e che è possibile rimuovere ed aggiungere mentre l'Activity è in esecuzione.
Un Fragment deve sempre essere inserito in un Activity ed il suo ciclo di vita è condizionato da quello dell'Activity in cui è inserito.
L'introduzione dei Fragment ha permesso di ottenere un più dinamico e flessibile design UI su schermi larghi, come quelli dei tablet.

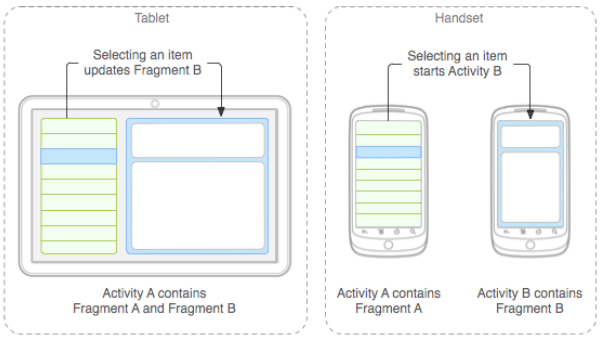

Figura 3.3: Fragments

3.3.5 Intent

Un Intent è un oggetto che è possibile utilizzare per richiedere un'azione da parte di un altro componente dell'applicazione.
Un Intent si può tradurre come un "intenzione".
Tramite un Intent, un'applicazione può dichiarare la volontà di compiere una particolare azione senza sapere come questa verrà poi eseguita.
Molte azioni disponibili su Android passano dall'utilizzo dell'Intent, di cui tre fondamentali:

- Avviare una Activity, passando un Intent al metodo startActivity(). L'intent descrive l'Activity da avviare e trasporta alcuni dati necessari.
- Avviare un Service, che è un componente che esegue operazioni in background senza l'interfaccia utente, passando un Intent al metodo startService(). L'intent descrive il Service da avviare e trasporta alcuni dati necessari.
- Inviare un broadcast, che è un messaggio che ogni app può ricevere, passando un Intent al metodo sendBroadcast().

3.4 Strumenti per lo sviluppo

Per la realizzazione di applicazioni Android è necessario predisporre un adeguato ambiente di sviluppo sulla propria macchina.
L'ambiente è composto da:

- Java Development Kit (JDK): è un insieme di strumenti per lo sviluppo di programmi Java, prodotto della Oracle.
  È l'ambiente di sviluppo più utilizzato dai programmatori Java soprattutto per applicazioni desktop.

- Android SDK: consiste in un pacchetto di sviluppo per applicazioni Android, attraverso cui uno sviluppatore può ottenere il codice sorgente (API e librerie) di una certa versione di Android, necessario per poter sviluppare e testare un'applicazione.
- Android Studio: è l'IDE ufficiale per lo sviluppo di applicazioni Android ed al suo interno in realtà sono inclusi già strumenti per lo sviluppo come Android SDK.

Capitolo 4

Progettazione

Come già anticipato nel capitolo introduttivo, si è deciso di sviluppare un'applicazione client da installare su smartphone Android e che sia in grado di interfacciarsi con il server preesistente permettendo quindi di soddisfare le richieste dell'utilizzatore.
Si è deciso di progettare in aggiunta un server di gestione di notifiche Firebase [4] allo scopo di permettere all'educatore di "risvegliare" l'applicazione sul telefono dell'utente contattato.
Tale server verrà descritto più avanti.
Si procederà ad una descrizione dei requisiti funzionali e non funzionali dell'applicazione, per poi elencare i vari casi di studio emersi.
Verranno descritte le classi che permettono il funzionamento dell'app e verrà mostrato il class diagram relativo alle sole classi che modellano il dominio dell'applicazione.
Infine verrà mostrato il navigation diagram, che illustra l'interazione tra le varie finestre, e il wireframing.

4.1 Applicazione Android

4.1.1 Requisiti funzionali

- Il sistema deve permettere all'utente di eseguire il login.
- Il sistema deve permettere all'utente di accedere ad una lista di percorsi e scegliere uno di essi.
- Il sistema deve permettere all'utente di visionare i dati del POI corrente durante la navigazione per immagini.

- Il sistema deve permettere all'utente di visionare la sua posizione GPS attraverso una mappa Google Maps durante la navigazione libera.
- Il sistema deve allertare l'utente se questi è fuori dall'area sicura.
- Il sistema deve chiamare automaticamente l'educatore in caso di mancanza di interazioni dell'utente dopo 15 secondi dall'alert.
- Il sistema, in caso di alert all'utente, deve inviare un sms all'educatore e al numero di emergenza contenente il link a Google Maps con la posizione utente.
- Il sistema, in caso di alert all'utente, deve inviare l'SMS in modo automatico oppure no in base ai dati di configurazione del sistema.
- Il sistema deve permettere all'utente di chiamare l'educatore.
- Il sistema deve inviare automaticamente un SMS all'educatore e al numero d'emergenza, contenente il link a Google Maps con la posizione utente, in caso di chiamata all'educatore.
- Il sistema deve permettere all'utente di scattare fotografie.
- Il sistema deve avviarsi automaticamente e geolocalizzare l'utente in caso di ricezione di notifica Firebase.
- Il sistema deve permettere all'utente di inserire i dati necessari alla corretta configurazione del sistema.

4.1.2 Requisiti non funzionali

- Dependability: Il sistema dovrebbe essere sempre disponibile nei momenti in cui l'utente ne faccia richiesta (Availability), e inoltre deve portare a termine i compiti richiesti correttamente (Reliability).
- Usability: Il sistema dovrebbe garantire all'utente semplicitá, chiarezza e usabilità della UI.
- Efficiency: Il sistema dovrebbe garantire una risposta all'utente in tempi accettabili, nell'ordine dei secondi.

4.1.3 Specifica dei requisiti: i casi d'uso

Assunzione: Si assume che l'utente utilizzatore configuri il sistema con l'aiuto dell'educatore.

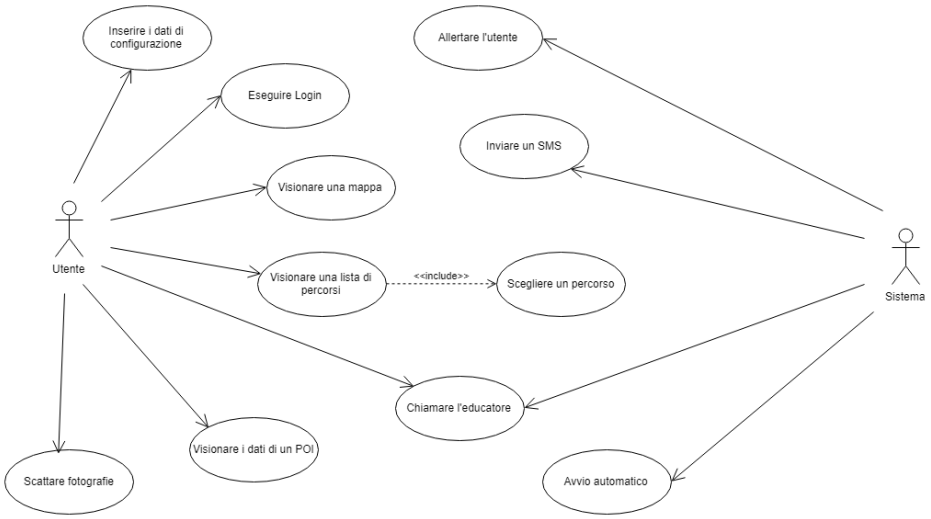

Figura 4.1: Diagramma dei casi d'uso

Eseguire login

Nome use case	Eseguire login
Attori partecipanti	Utente
Descrizione	Funzione che permette all'utente di accedere al sistema mediante l'utilizzo del lettore d'impronte oppure mediante l'inserimento dei dati necessari al login
Evento scatenante	L'utente vuole accedere al sistema
Conseguenza	L'utente accede al sistema

Tabella 4.1: Eseguire login

Visionare una lista di percorsi

Nome use case	Visionare una lista di percorsi
Attori partecipanti	Utente
Descrizione	Funzione che permette all'utente di consultare una lista di percorsi e scegliere uno di essi per essere utilizzato nella navigazione per immagini
Evento scatenante	L'utente vuole visionare la lista di percorsi
Conseguenza	L'utente utilizza la navigazione per immagini associata al percorso scelto

Tabella 4.2: Visionare una lista di percorsi

Visionare i dati di un POI

Nome use case	Visionare i dati di un POI
Attori partecipanti	Utente
Descrizione	Funzione che permette all'utente di visionare uno ad uno i POI durante la navigazione per immagini del percorso scelto
Evento scatenante	Scelta di un percorso dalla lista di percorsi
Conseguenza	Inizio della navigazione per immagini, viene fatto partire un countdown relativo al tempo massimo di percorrenza del percorso, allo scadere di esso l'utente viene allertato secondo lo stesso pattern presente nella navigazione libera

Tabella 4.3: Visionare i dati di un POI

Visionare una mappa

Nome use case	Visionare una mappa
Attori partecipanti	Utente
Descrizione	Funzione che permette di mostrare una mappa contenente un Marker alla posizione GPS dell'utente durante la navigazione libera
Evento scatenante	L'utente vuole accedere all'area di navigazione libera oppure l'app viene "risvegliata" da una notifica Firebase
Conseguenza	La posizione dell'utente viene monitorata, se questi esce dall'area sicura il sistema reagisce con un alert

Tabella 4.4: Visionare una mappa

Chiamare l'educatore

Nome use case	Chiamare l'educatore
Attori partecipanti	Utente, Sistema
Descrizione	Funzione che permette di chiamare l'educatore, il numero viene composto automaticamente e viene intrapresa la chiamata
Evento scatenante	L'utente preme il tasto SOS oppure il sistema avvia la chiamata automaticamente a seguito dell'alert all'utente
Conseguenza	Viene effettuata la chiamata all'educatore e inoltre viene mandato un SMS contenente il link a Google Maps con la posizione utente all'educatore e al numero d'emergenza

Tabella 4.5: Chiamare l'educatore

Scattare fotografie

Nome use case	Scattare fotografie
Attori partecipanti	Utente
Descrizione	Funzione che permette all'utente di accedere alla fotocamera e scattare una fotografia
Evento scatenante	L'utente preme il tasto in-app che permette di scattare una foto
Conseguenza	L'utente scatta la foto e l'applicazione la invia al server

Tabella 4.6: Scattare fotografie

Inserire i dati di configurazione

Nome use case	Inserire i dati di configurazione
Attori partecipanti	Utente
Descrizione	Funzione che permette all'utente di inserire i dati necessari alla configurazione del sistema
Evento scatenante	L'utente vuole accedere al menù delle preferenze
Conseguenza	Le preferenze dell'utente vengono salvate, i dati relativi all'utente quali il nome e il numero di telefono vengono inviati al server di gestione di notifiche Firebase insieme al token univoco (associato all'utente) che permette alla piattaforma Firebase di contattare il client specifico

Tabella 4.7: Inserire i dati di configurazione

Allertare l'utente

Nome use case	Allertare l'utente
Attori partecipanti	Utente, Sistema
Descrizione	Funzione che permette al sistema di creare un popup che consente all'utente di chiamare oppure no l'educatore, successivamente viene mandato un SMS contenente il link a Google Maps con la posizione dell'utente all'educatore e al numero d'emergenza, tale invio può essere automatico oppure no in base alle scelte salvate nelle preferenze
Evento scatenante	L'utente esce dall'area sicura
Conseguenza	Il sistema allerta l'utente

Tabella 4.8: Allertare l'utente

Inviare un SMS

Nome use case	Inviare un SMS
Attori partecipanti	Sistema
Descrizione	Funzione che permette al sistema di comporre un SMS contenente il link a Google Maps con la posizione dell'utente e di inviarlo all'educatore e al numero d'emergenza
Evento scatenante	L'utente decide di chiamare l'educatore o, in base alla voce nelle preferenze, l'utente dà il proprio consenso all'invio dell'SMS, oppure viene intrapresa l'azione automaticamente
Conseguenza	Viene inviato un SMS contenente il link a Google Maps con la posizione dell'utente all'educatore e al numero d'emergenza

Tabella 4.9: Inviare un SMS

Avvio automatico

Nome use case	Avvio automatico
Attori partecipanti	Sistema
Descrizione	Funzione che permette al sistema di avviarsi automaticamente in seguito alla ricezione di una notifica Firebase
Evento scatenante	L'educatore invia un segnale dal server di gestione di notifiche Firebase
Conseguenza	La notifica risveglia l'applicazione che apre automaticamente la finestra di navigazione libera. In essa l'utente viene geolocalizzato al fine di allertarlo in caso di uscita dall'area sicura

Tabella 4.10: Avvio automatico

4.1.4 Le classi

Si andrà ora a descrivere tutte le classi che compongono l'applicativo (divise per packages) e successivamente verrà mostrato il class diagram che descriverà le classi che costituiscono il dominio dell'applicazione.

- **Package casatracking.model**: Il package contiene le classi che modellano il dominio dell'applicazione.
  - **classe Utente**: classe che modella i dati di cui l'utente ha bisogno durante l'utilizzo dell'applicativo; i dati legati ad esso sono il nome, il numero di telefono, il numero di telefono dell'educatore e il numero di telefono d'emergenza.
  - **classe POI**: classe che modella i dati relativi ad un POI (Point of Interest); i dati legati ad esso sono il nome, l'id del percorso ad esso associato, la foto e la descrizione del POI.
  - **classe Percorso**: classe che modella i dati relativi ad un percorso; i dati legati ad esso sono l'id, il nome e il tempo massimo di percorrenza.
- **Package casatracking.services**: Il package contiene le classi servizio dell'applicazione.

  - **classe CasaTrackingFirebaseInstanceIDService**: classe che estende la classe servizio FirebaseInstanceIDService, tale classe si occupa di inviare al server di gestione notifiche Firebase il token che identifica univocamente l'utente utilizzatore, e che sarà utilizzato allo scopo di "risvegliare" l'applicazione nel momento in cui l'educatore ne faccia richiesta.
  - **classe CasaTrackingFirebaseMessagingService**: classe che estende la classe servizio FirebaseMessagingService, tale classe si occupa di gestire la ricezione di una notifica Firebase grazie all'implementazione del metodo onMessageReceived(), l'implementazione di tale metodo verrà descritta nel capitolo successivo.
  - **classe RequestService**: servizio che si occupa di comunicare con il server e di interpretare i dati ricevuti.
  - **classe Services**: servizio che si occupa di gestire compiti legati a richieste dell'utente quali la chiamata automatica all'educatore e l'invio automatico di SMS.

- **Package casatracking.utils**: Il package contiene classi utili allo svolgimento di particolari compiti.
  - **classe GenericFileProvider**: classe necessaria al salvataggio in locale di foto scattate dall'utente in attesa di essere inviate al server oppure di immagini scaricate dal server.
  - **classe Images**: classe necessaria alla gestione delle immagini.
  - **classe Player**: classe necessaria alla gestione della vibrazione e del suono a seguito di alert dell'utente.
  - **classe Preferences**: classe necessaria alla gestione delle preferenze utente.
  - **classe Request**: classe utilizzata dal servizio RequestService allo scopo di comunicare con il server.
  - **classe Timer**: classe necessaria alla corretta gestione del countdown di percorrenza di un percorso utente.

- **Package casatracking**: Il package contiene le classi responsabili della gestione delle finestre dell'applicazione.
  - **classe ChoiceActivity**: classe Activity che gestisce la finestra che permette all'utente di scegliere tra la modalità di navigazione libera e la modalità di navigazione per immagini. Inotre permette all'utente di eseguire chiamate automatiche all'educatore.
  - **classe LoginActivity**: classe Activity che gestisce la finestra che permette all'utente di inserire i dati necessari al login nel sistema.

- **classe MainActivity**: classe Activity avviata con l'avvio dell'applicazione. Essa gestisce la finestra che permette all'utente di accedere al menù delle preferenze, al metodo di autenticazione con l'impronta digitale e al metodo di autenticazione mediante inserimento dei dati necessari.
- **classe NavigazioneLiberaActivity**: classe Activity che gestisce la navigazione libera. La finestra contiene un Fragment che consiste nella mappa Google Maps su cui è possibile visionare la posizione GPS relativa all'utente utilizzatore. Inoltre permette di gestire la logica necessaria all'alerting dell'utente. Infine permette all'utente di scattare fotografie e di eseguire chiamate automatiche all'educatore.
- **classe PercorsiAdapter**: classe che estende RecycleView.Adapter. Essa si occupa di creare e dare logica agli elementi della lista dalla quale è possibile scegliere un percorso.
- **classe POIActivity**: classe Activity che gestisce la navigazione per immagini. La finestra contiene l'immagine, il nome e la descrizione del POI corrente. I POI uno dopo l'altro verranno mostrati all'utente in movimento. Inoltre permette all'utente di scattare fotografie e di eseguire chiamate automatiche all'educatore.
- **classe ScegliPercorsoActivity**: classe Activity che gestisce la finestra che permette all'utente di scegliere un percorso da una lista. Inoltre permette all'utente di eseguire chiamate automatiche all'educatore.
- **classe SettingsActivity**: classe Activity che gestisce le preferenze dell'applicazione. La finestra permette all'utente di immettere i dati relativi ad esso e di settare alcuni parametri utilizzati dal sistema.

Class diagram

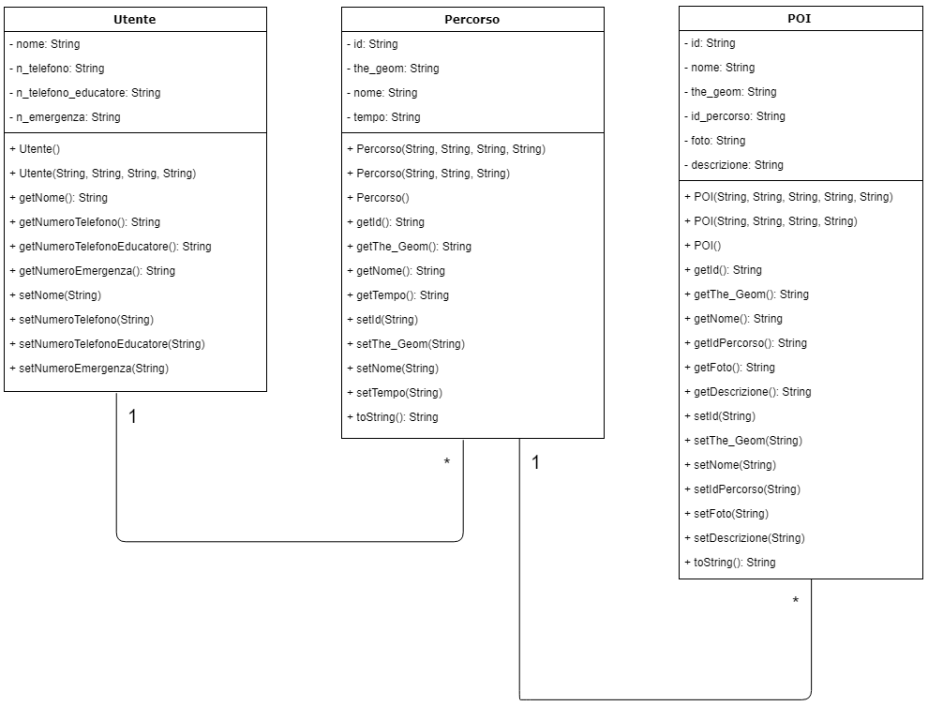

Figura 4.2: Class diagram

4.1.5 Navigation Diagram

Si andrà ora a mostrare il navigation diagram che illustra l'interazione tra le finestre dell'applicazione.

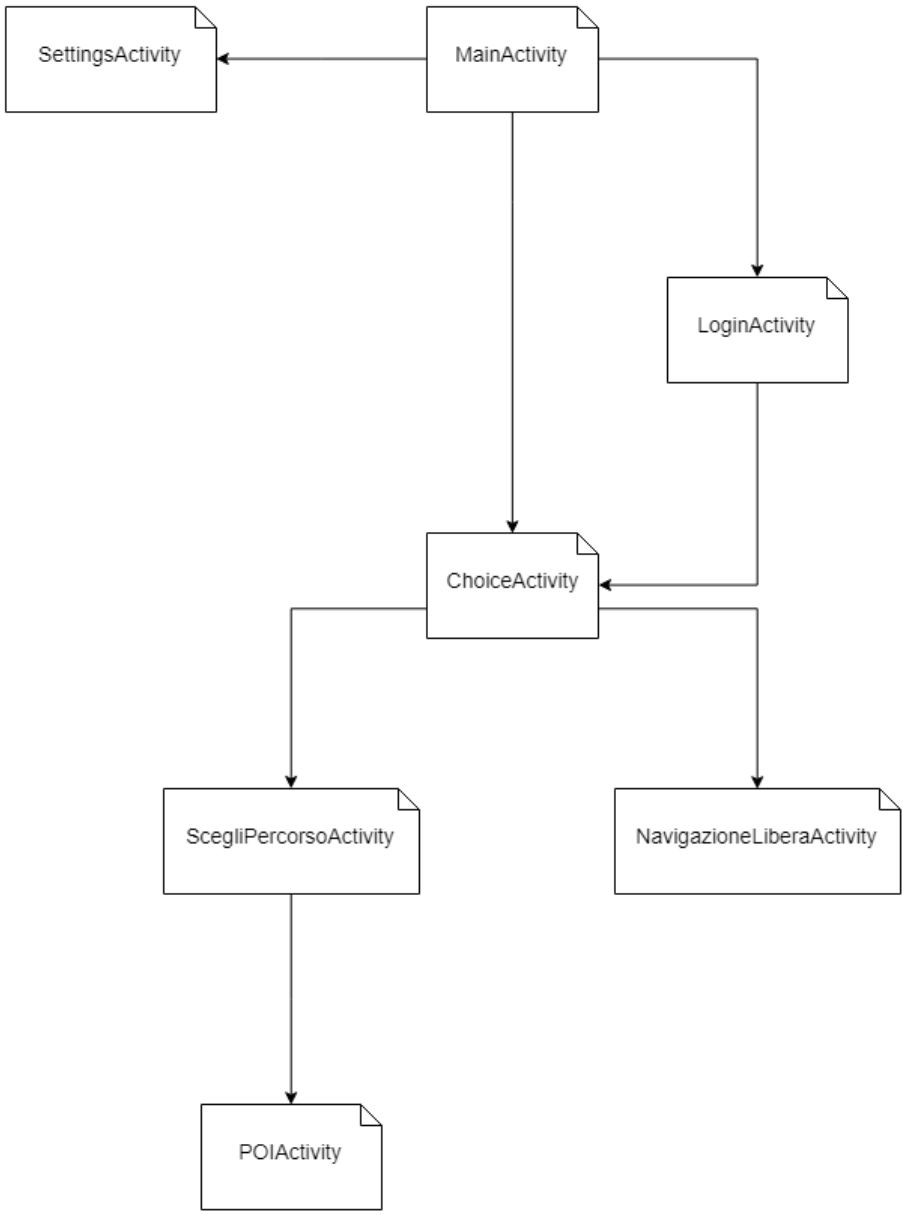

Figura 4.3: Navigation diagram

4.1.6 Wireframing

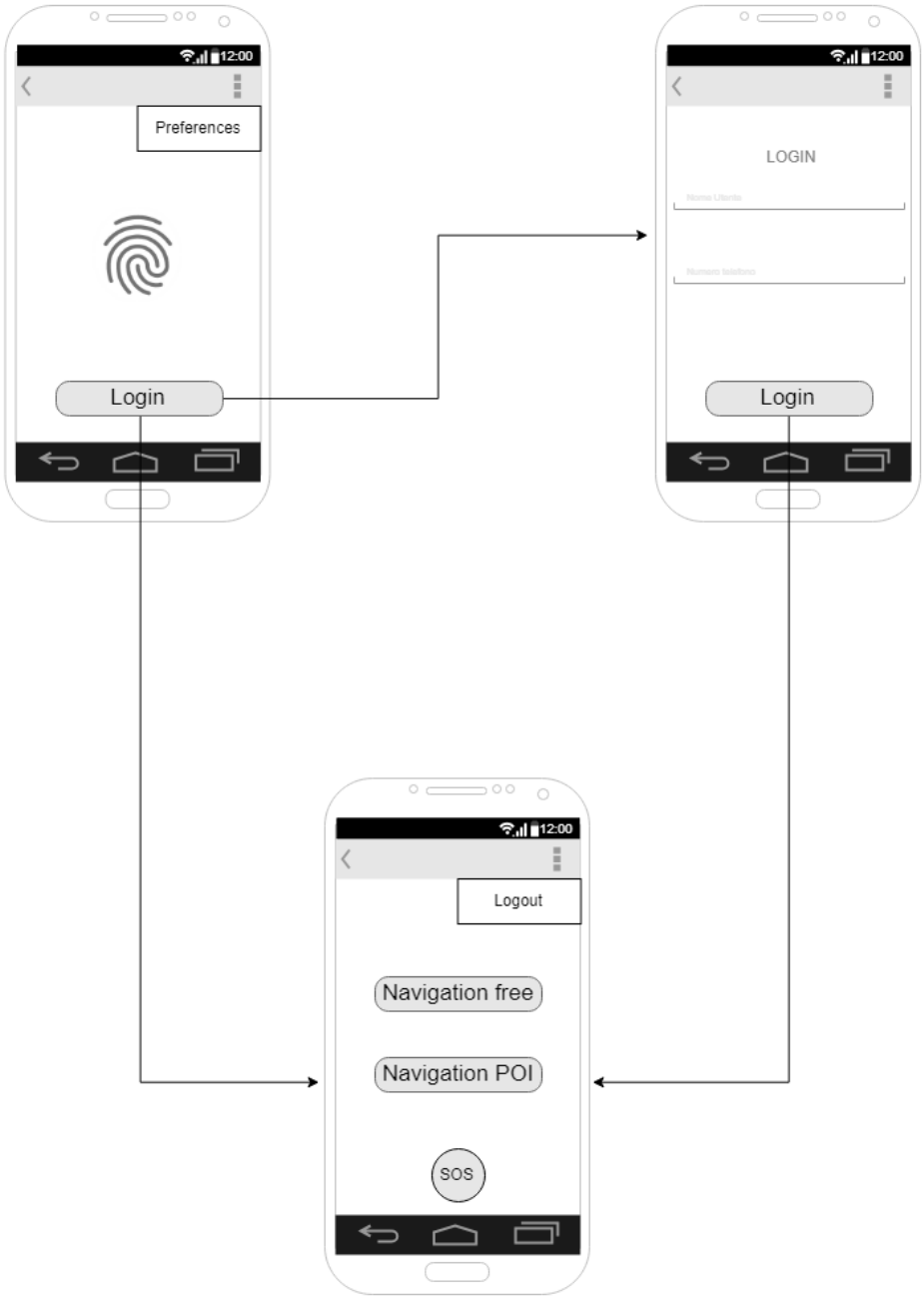

Figura 4.4: Descrizione della fase di Login

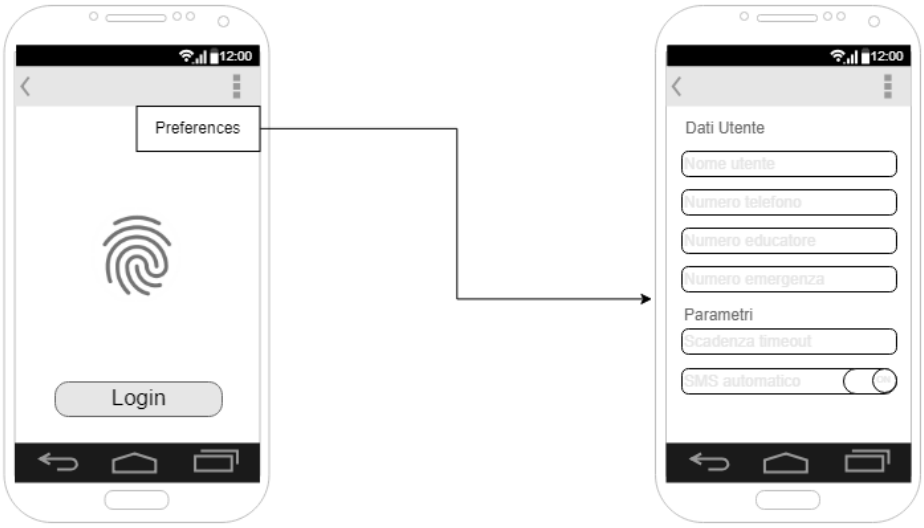

Figura 4.5: Descrizione della configurazione delle preferenze

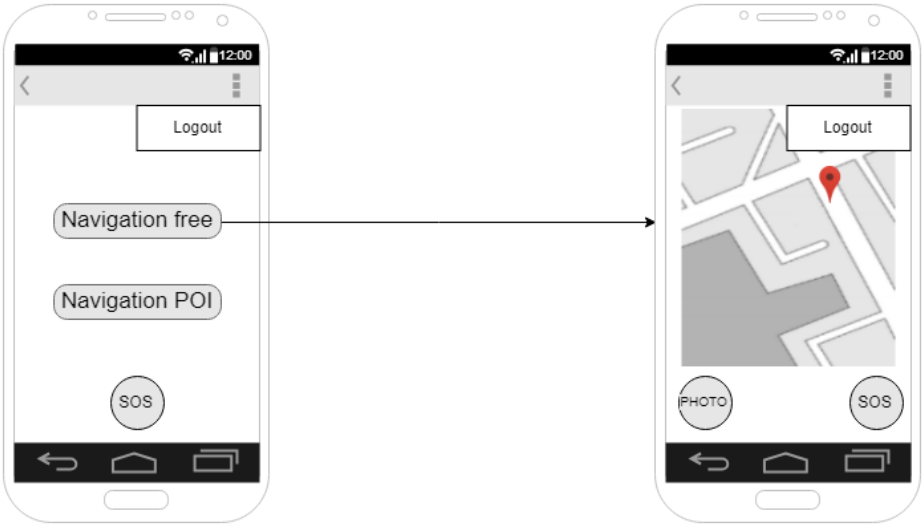

Figura 4.6: Descrizione della navigazione libera

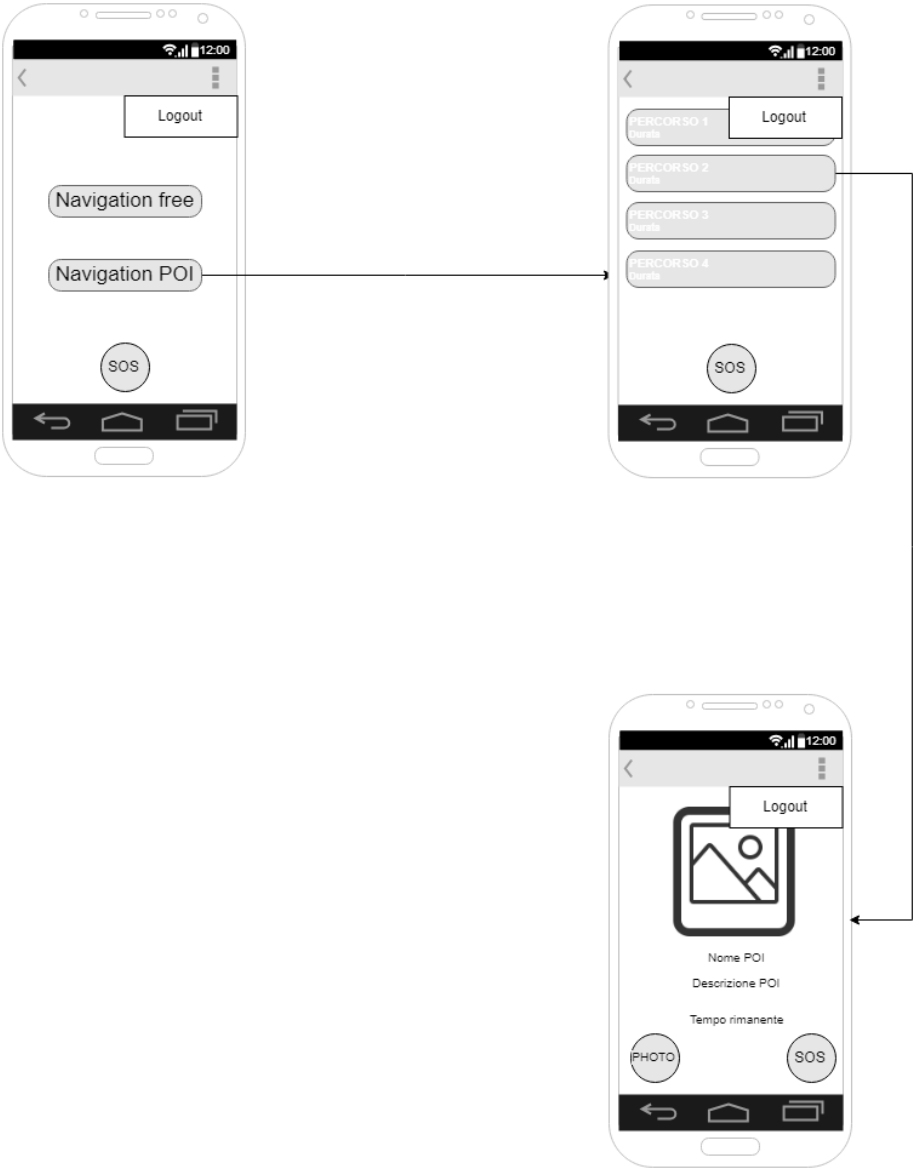

Figura 4.7: Descrizione della navigazione per immagini

4.2 Server di gestione notifiche Firebase

Al fine di raccogliere tutti i token Firebase e di permettere all'educatore di inviare una notifica Firebase all'utente scelto, si è deciso di progettare un server con logica PHP/JS ed una pagina web che permette di interfacciarsi ad esso.
La pagina web mostra una select contenente, per ogni utente registrato, il nome e

il numero di telefono corrispondente.
Si veda nelle prossime immagini la conformazione della pagina web, l'implementazione del server verrà trattata nel prossimo capitolo.

Figura 4.8: Screenshot Server Firebase: nessun utente registrato

Figura 4.9: Screenshot Server Firebase: select non utilizzata

Figura 4.10: Screenshot Server Firebase: select utilizzata

Capitolo 5

Implementazione

In questo capitolo verranno riportate alcune porzioni del codice prodotto durante la fase implementativa del sistema progettato.
Verranno mostrati dettagli sia sull'applicazione Android sia sulla parte del server per la gestione di notifiche Firebase.
Per quanto riguarda il lato Firebase si vedranno porzioni di codice relativo al PHP utilizzato per gestire l'invio della notifica al client target.
Per quanto concerne invece il lato dell'applicazione Android, saranno mostrati alcuni punti chiave che implementano le principali funzionalità.

5.1 Applicazione Android

L'implementazione dell'applicazione è stata realizzata utilizzando come ambiente di sviluppo Android Studio.
Come già evidenziato all'interno della sezione di progettazione la struttura del progetto ha portato a una suddivisione logica del codice in packages.
La struttura dell'applicazione viene descritta nelle figure 5.1 e 5.2.
L'Activity selezionata è quella aperta all'avvio dell'applicazione.

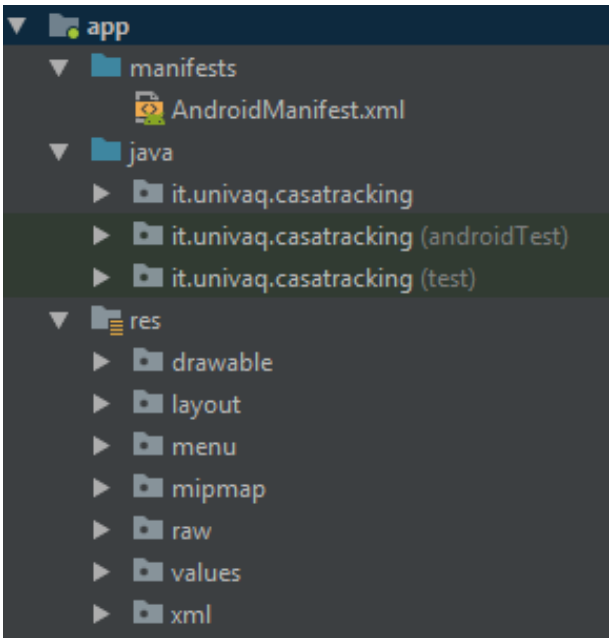

Figura 5.1: Struttura applicazione Android - 1

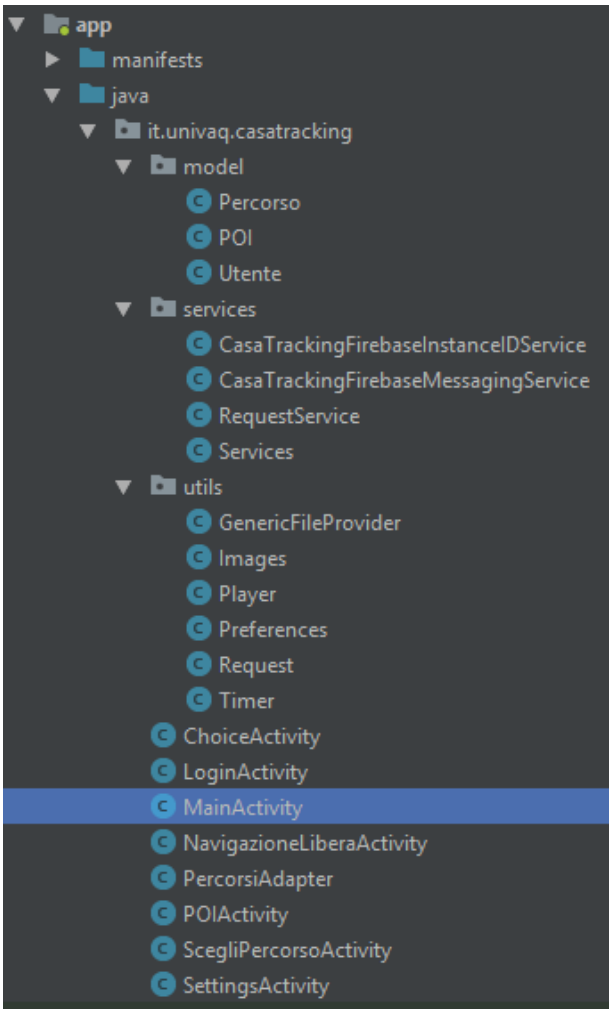

Figura 5.2: Struttura applicazione Android - 2

- Nel package casatracking sono implementate le Activity e i Fragment che costituiscono il cuore dell'applicazione.

- Nel package casatracking.models sono presenti le classe che modellano il dominio dell'applicazione, ovvero le classi Utente, Percorso e POI.

- Nel package casatracking.utils sono state implementate tutte le classe ausiliarie necessarie al corretto funzionamento dell'applicazione. Ovvero le classi utilizzate per la connessione al server, all'implementazione delle preferenze e al richiamo di API esterne.

- Nel package casatracking.services sono presenti le classi che svolgono servizi all'utente, quali il servizio che permette la connessione al server e il servizio che permette la chiamata e l'invio di SMS.

- All'interno della directory res sono presenti tutte le risorse richiamate all'interno del progetto come immagini, stringhe e nella sottodirectory layout, tutti i layout utilizzati.

5.1.1 La funzione alert()

In questa sezione verrà mostrato ciò che è probabilmente la funzione più importante dell'applicazione.
Tale funzione si trova nella classe Activity NavigazioneLiberaActivity.
La funzione alert() viene chiamata all'uscita dell'utente dall'area sicura. Essa crea il popup che permette all'utente di chiamare l'educatore ed invia l'SMS ad esso e al numero d'emergenza in modo automatico oppure no, basandosi sulle preferenze dell'applicazione.
Sarà ora listato il codice relativo funzione.

```
1  private void alert(final LatLng loc){
2
3  if(alertIsActive){
4  return;
5  }
6
7  alertIsActive = true;
8
9  Player.getInstance(getApplicationContext()).startPlaying();
10
11 dismissed = false;
12
13 AlertDialog.Builder builder = new AlertDialog.Builder(
       NavigazioneLiberaActivity.this, android.R.style.
       Theme_Material_Dialog_Alert);
14
15 builder.setTitle(getApplicationContext().getString(R.string.
       alert_title))
16
```

```
17 .setMessage(getApplicationContext().getString(R.string.
       alert_call_educatore))
18
19 .setPositiveButton(R.string.button_si, new DialogInterface.
       OnClickListener() {
20 @Override
21 public void onClick(DialogInterface dialog, int which) {
22
23 Player.getInstance(getApplicationContext()).stopPlaying();
24
25 Intent doAlert = new Intent(getApplicationContext(), Services.
       class);
26 doAlert.setAction(Services.ACTION_ALERT);
27 doAlert.putExtra("sms_body", "SONO FUORI DALLA MIA AREA SICURA");
28 doAlert.putExtra("loc", loc);
29 startService(doAlert);
30
31 notify_cancelled = true;
32 dismissed = true;
33 dialog.dismiss();
34
35 alertIsActive = false;
36
37 getWindow().clearFlags(WindowManager.LayoutParams.
       FLAG_DISMISS_KEYGUARD | WindowManager.LayoutParams.
       FLAG_SHOW_WHEN_LOCKED | WindowManager.LayoutParams.
       FLAG_TURN_SCREEN_ON);
38 autoCallHandler.removeCallbacks(autoCallRunnable);
39 }
40 })
41
42 .setNegativeButton(R.string.button_no, new DialogInterface.
       OnClickListener() {
43 @Override
44 public void onClick(DialogInterface dialog, int i) {
45
```

```
46 Player.getInstance(getApplicationContext()).stopPlaying();
47
48 if(!Preferences.checkAutomaticSMS(getApplicationContext())){
49
50 AlertDialog.Builder builder2 = new AlertDialog.Builder(
       NavigazioneLiberaActivity.this, android.R.style.
       Theme_Material_Dialog_Alert);
51
52 builder2.setTitle(getApplicationContext().getString(R.string.
       alert_title))
53 .setMessage(getApplicationContext().getString(R.string.
       alert_sms_educatore))
54 .setPositiveButton(R.string.button_si, new DialogInterface.
       OnClickListener() {
55 @Override
56 public void onClick(DialogInterface dialogInterface, int i) {
57 Intent sms = new Intent(getApplicationContext(), Services.class);
58 sms.setAction(Services.ACTION_SEND_SMS);
59 sms.putExtra("sms_body", "SONO FUORI DALLA MIA AREA SICURA");
60 sms.putExtra("loc", loc);
61 startService(sms);
62
63 dialogInterface.dismiss();
64 }
65 })
66
67 .setNegativeButton(R.string.button_no, new DialogInterface.
       OnClickListener() {
68 @Override
69 public void onClick(DialogInterface dialogInterface, int i) {
70 dialogInterface.dismiss();
71 }
72 })
73
74 .setIcon(android.R.drawable.ic_dialog_alert)
75 .setCancelable(false)
```

```
76 .show();
77
78 } else {
79
80 Intent sms = new Intent(getApplicationContext(), Services.class);
81 sms.setAction(Services.ACTION_SEND_SMS);
82 sms.putExtra("sms_body", "SONO FUORI DALLA MIA AREA SICURA");
83 sms.putExtra("loc", loc);
84 startService(sms);
85 }
86
87 notify_cancelled = true;
88 dismissed = true;
89 dialog.dismiss();
90
91 alertIsActive = false;
92
93 getWindow().clearFlags(WindowManager.LayoutParams.
       FLAG_DISMISS_KEYGUARD | WindowManager.LayoutParams.
       FLAG_SHOW_WHEN_LOCKED | WindowManager.LayoutParams.
       FLAG_TURN_SCREEN_ON);
94 autoCallHandler.removeCallbacks(autoCallRunnable);
95 }
96 })
97
98 .setIcon(android.R.drawable.ic_dialog_alert)
99 .setCancelable(false)
100 .show();
101
102 getWindow().addFlags(WindowManager.LayoutParams.
       FLAG_DISMISS_KEYGUARD | WindowManager.LayoutParams.
       FLAG_SHOW_WHEN_LOCKED | WindowManager.LayoutParams.
       FLAG_TURN_SCREEN_ON);
103
104 autoCallRunnableLatLng = loc;
```

```
105 autoCallHandler.postDelayed(autoCallRunnable,
        TIME_OUT_AUTOMATIC_CALL);
106 }
```

Si può notare che nella parte iniziale della funzione (righe 1-7) viene settata a true una variabile booleana di nome alertIsActive. Grazie ad essa viene servita una sola chiamata alla volta di alert(), le altre vengono scartate poichè la chiamata è già attiva.
Per adempiere a tale compito la variabile viene settata a false alla chiusura del popup (righe 35 e 91).

Successivamente viene utilizzata un'istanza di Player per avviare la vibrazione del telefono e l'alert sonoro (riga 9).

A seguire viene creato il popup con il messaggio "Chiamare l'educatore?" grazie alla classe AlertDialog.Builder e viene data logica al bottoni "SI" e "NO" mediante le rispettive chiamate a setPositiveButton() e setNegativeButton() (righe 13-100).

Nella funzione setPositiveButton() (righe 19-40) viene innanzitutto stoppata la vibrazione e l'alert sonoro, richiamando la classe Player. Successivamente viene creato e settato l'Intent che permette la comunicazione con la classe servizio Services, e viene iniziato il servizio mediante la chiamata a startService().
Il servizio si occuperà di chiamare l'utente ed inviare l'SMS con la posizione GPS.

Nella funzione setNegativeButton() (righe 42-96) viene innanzitutto stoppata la vibrazione e l'alert sonoro, richiamando la classe Player.
Successivamente viene intrapresa un'azione in base alla preferenza dell'invio automatico dell'SMS:

- Se l'invio automatico risulta attivo, viene creato l'Intent che attiverà il solo invio dell'SMS con la posizione automatica mediante la classe servizio Services.
- Se l'invio automatico risulta disattivo, viene creato il popup con il messaggio "Inviare l'SMS con la posizione?" grazie alla classe AlertDialog.Builder e viene data logica al bottoni "SI" e "NO" mediante le rispettive chiamate a setPositiveButton() e setNegativeButton().
  La chiamata a setPositiveButton() (righe 54-65) prepara l'Intent che attiverà l'invio dell'SMS con la posizione automatica mediante la classe servizio Services.
  La chiamata a setNegativeButton() (righe 67-72) non esegue alcuna azione.
  Infine viene settato handler che si occuperà di chiamare automaticamente l'educatore se l'utente non interagirà con il popup entro 15 secondi (righe 104-105).

5.1.2 Notifiche Firebase

In questa sezione verrà mostrato il codice relativo alla gestione delle notifiche Firebase lato client.
In primo luogo verrà presentata la parte che ha il compito di ricevere il token che identifica univocamente il telefono dalle API ufficiali di Firebase, ed invia esso al server di gestione notifiche Firebase, dove i token vengono associati all'utente.
In seguito verrà mostrata la funzione che alla ricezione di una notifica Firebase esegue l'azione di risvegliare l'applicazione sul dispositivo target.

```
1  public static boolean sendRegistrationToServer(@Nullable String
       token, Context context, Utente utente) {
2
3  if(token == null){
4  return false;
5  }
6
7  boolean success = false;
8  HttpURLConnection connection = null;
9
10 try {
11
12 URL url = new URL(context.getString(R.string.firebase_server_path)
       );
13
14 connection = (HttpURLConnection) url.openConnection();
15
16 connection.setDoOutput(true);
17 connection.setDoInput(true);
18
19 connection.setRequestMethod("POST");
20
21 DataOutputStream dos = new DataOutputStream(connection.
       getOutputStream());
22
23 dos.writeBytes("task=" + "store_data" + "&token=" + token.trim() +
        "&phone=" + utente.getNumeroTelefono().trim() + "&userName="
      + utente.getNome());
```

```
24 dos.flush();
25 dos.close();
26
27 if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
28 success = true;
29 } else {
30 success = false;
31 }
32
33 } catch (IOException e) {
34 e.printStackTrace();
35
36 } finally {
37 if(connection != null)
38 connection.disconnect();
39
40 }
41
42 return success;
43 }
```

Nella parte iniziale della funzione viene eseguito un controllo di nullità sulla variabile token. (righe 3-5)
La variabile token viene fornita dalla chiamata alle API ufficiali di Firebase, essa è una stringa che identifica univocamente il client.
Successivamente viene creata la connessione al server che si occupa di gestire i token Firebase, e ad esso viene inviato il token, il numero di telefono dell'utente e il nome. (righe 12-25)
Tali dati vengono utilizzati per riconoscere l'utente a cui mandare la notifica da server.
Infine la connessione viene chiusa (riga 38).

Ora verrà listata la parte di gestione della ricezione delle notifiche.

```
1 @Override
2 public void onMessageReceived(RemoteMessage remoteMessage) {
3
4 super.onMessageReceived(remoteMessage);
5
6 if(!Preferences.checkFirstAccess(getApplicationContext()))
```

```
7  startNavigazioneLiberaActivity();
8  }
9
10 private void startNavigazioneLiberaActivity() {
11
12 Intent i = new Intent(getApplicationContext(),
       NavigazioneLiberaActivity.class);
13 startActivity(i);
14
15 }
```

Alla ricezione di una notifica Firebase, il sistema esegue una chiamata al metodo onMessageReceived().
Nella funzione viene controllato se l'utente ha fatto mai l'accesso in precedenza, se è così allora viene fatta una chiamata al metodo startNavigazioneLiberaActivity() che avvia tale Activity. (righe 2-8)
Essa una volta avviata inizia la geolocalizzazione dell'utente per capire se esso è dentro oppure fuori l'area sicura e intraprende le successive azioni.

Per permettere il corretto funzionamento sono necessare le seguenti librerie, aggiunte nella sezione dipendenze del file build.gradle relativo all'app.

```
1 dependencies {
2
3     implementation 'com.google.firebase:firebase-core:15.0.2'
4     implementation 'com.google.firebase:firebase-messaging:15.0.2'
5
6 }
```

5.1.3 Login tramite impronta digitale

In questa sezione verrà mostrato il codice relativo alla gestione dell'autenticazione tramite utilizzo di impronte digitali.
Il codice è strettamente integrato con l'Activity MainActivity.
Per questione di semplicità verrà mostrato solamente il codice relativo alla gestione dell'autenticazione tramite impronta, il codice della restante parte dell'Activity non verrà mostrata.
Verranno aggiunti dei commenti che semplificheranno ulteriormente la comprensione del codice.

```
1  public class MainActivity extends AppCompatActivity implements
       FingerPrintAuthCallback {
2
3  private FingerPrintAuthHelper mFingerPrintAuthHelper;
4
5  @Override
6  protected void onCreate(Bundle savedInstanceState) {
7  super.onCreate(savedInstanceState);
8  setContentView(R.layout.activity_main);
9
10 mFingerPrintAuthHelper = FingerPrintAuthHelper.getHelper(this,
       this);
11
12 }
13
14 @Override
15 protected void onResume() {
16 super.onResume();
17
18 boolean isFirstAccess = Preferences.checkFirstAccess(
       getApplicationContext());
19
20 if (isFirstAccess) {
21
22   //Handle first access
23
24 } else {
25
26   mFingerPrintAuthHelper.startAuth();
27
28 }
29
30 }
31
32 @Override
33 protected void onStop() {
```

```
34 super.onStop();
35
36 mFingerPrintAuthHelper.stopAuth();
37
38 }
39
40 @Override
41 public void onNoFingerPrintHardwareFound() {
42 //Device does not have finger print scanner.
43 //other auth method
44
45 messaggio.setText(getApplicationContext().getString(R.string.
       textview_no_hardware_for_scan_fingerprint));
46
47 }
48
49 @Override
50 public void onNoFingerPrintRegistered() {
51
52 //There are no finger prints registered on this device.
53 //Show popup if is first time
54
55 if(Preferences.checkNoFingerprintRegisteredFirstTime(
       getApplicationContext())){
56 //first time no fingerprint registered
57 //show popup
58
59 AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity
       .this, android.R.style.Theme_Material_Dialog_Alert);
60
61 builder.setTitle(getApplicationContext().getString(R.string.
       alert_title))
62
63 .setMessage(getApplicationContext().getString(R.string.
       alert_no_fingerprints))
```

```
64 .setPositiveButton(R.string.button_si , new DialogInterface.
       OnClickListener() {
65 public void onClick(DialogInterface dialog, int which) {
66
67 //redirect to settings/fingerprints
68 FingerPrintUtils.openSecuritySettings(MainActivity.this);
69
70 dialog.dismiss();
71
72 }
73 })
74
75 .setNegativeButton(R.string.button_no, new DialogInterface.
       OnClickListener() {
76 @Override
77 public void onClick(DialogInterface dialog, int i) {
78
79 messaggio.setText(getApplicationContext().getString(R.string.
       textview_no_fingerprint_registered));
80 dialog.dismiss();
81 }
82 })
83
84 .setIcon(android.R.drawable.ic_dialog_alert)
85 .setCancelable(false)
86 .show();
87
88 Preferences.cancelNoFingerprintRegisteredFirstTime(
       getApplicationContext());
89
90 } else {
91
92 //else no action, only show message  in TextView
93 messaggio.setText(getApplicationContext().getString(R.string.
       textview_no_fingerprint_registered));
94
```

```
95 }
96
97 }
98
99 @Override
100 public void onBelowMarshmallow() {
101
102 //Device running below API 23 version of android that does not
       support finger print authentication.
103 //other auth method
104
105 messaggio.setText(getApplicationContext().getString(R.string.
       textview_no_hardware_for_scan_fingerprint));
106
107 }
108
109 @Override
110 public void onAuthSuccess(FingerprintManager.CryptoObject
       cryptoObject) {
111 //Authentication successful.
112 //handle auth
113
114 //next page
115 Intent i = new Intent(getApplicationContext(), ChoiceActivity.
       class);
116 startActivity(i);
117
118 }
119
120 @Override
121 public void onAuthFailed(int errorCode, String errorMessage) {
122
123 //Parse the error code for recoverable/non recoverable error.
124 switch (errorCode) {
125
126 case AuthErrorCodes.CANNOT_RECOGNIZE_ERROR:
```

```
127 //Cannot recognize the fingerprint scanned.
128 messaggio.setText(getApplicationContext().getString(R.string.
        textview_riprova));
129 break;
130
131 case AuthErrorCodes.NON_RECOVERABLE_ERROR:
132 //This is not recoverable error. Try other options for user
        authentication. like pin, password.
133 messaggio.setText(getApplicationContext().getString(R.string.
        textview_error_not_recoverable));
134 break;
135
136 case AuthErrorCodes.RECOVERABLE_ERROR:
137 //Any recoverable error. Display message to the user.
138 messaggio.setText(getApplicationContext().getString(R.string.
        textview_riprova));
139 break;
140 }
141 }
142
143 }
```

Nel metodo onCreate() (righe 6-12) viene inizializzata un'istanza di FingerPrintAuthHelper, classe che si occupa della rilevazione delle impronte digitali.

Nel metodo onResume() (righe 15-30) nel caso in cui l'utente ha già inserito i suoi dati nel sistema, viene iniziata la rilevazione delle impronte mediante la chiamata al metodo della classe FingerPrintAuthHelper chiamato startAuth().

Nel metodo onStop() (righe 33-38) viene fermata la rilevazione delle impronte mediante la chiamata al metodo della classe FingerPrintAuthHelper chiamato stopAuth().

Nel metodo onNoFingerPrintHardwareFound() (righe 41-47) viene gestita la possibile mancanza del lettore d'impronte nel telefono, in tal caso viene avvisato l'utente di utilizzare il metodo di autenticazione tradizionale, ovvero immettendo i dati utente in un'apposita form.

Nel metodo onNoFingerPrintRegistered() (righe 50-97) viene gestita la possibile mancanza di impronte registrate nel telefono client.

L'utente viene avvisato solamente una volta di tale mancanza per evitare di appesantire l'utilizzo dell'applicazione.
Nel caso in cui sia la prima volta che non si rilevano impronte digitali registrate, si crea un popup che avvisa l'utente con il messaggio "Nessuna impronta registrata, inserirle?", la successiva azione dipende dalla scelta dell'utente.
Le scelte possibili sono "SI" e "NO". Le azioni possibili sono gestite rispettivamente dalle funzioni setPositiveButton() e setNegativeButton(). (righe 59-86)
Alla scelta "SI" l'utente verrà diretto nelle impostazioni del telefono dove è possibile inserire le impronte. Tale azione è possibile chiamando il metodo della classe FingerPrintUtils di nome openSecuritySettings(). (riga 68)

Alla scelta "NO" non verrà eseguita alcuna azione.

Nel metodo onBelowMarshmallow() (righe 100-107) viene gestito il fatto che al di sotto dell'API 23 Android non è supportata l'autenticazione tramite impronte. In tal caso viene avvisato l'utente di utilizzare il metodo di autenticazione tradizionale.

Nel metodo onAuthSuccess() (righe 110-118) viene gestita l'autenticazione avvenuta con successo.
Ovvero viene avviata l'Activity ChoiceActivity che permette la scelta tra navigazione libera e navigazione per immagini.

Nel metodo onAuthFailed() (righe 121-141) viene gestito un qualsiasi errore nell'autenticazione.
Viene mostrato un messaggio all'utente in base all'errore incontrato.

Per utilizzare il lettore d'impronte digitali si è scelta la seguente libreria [5], aggiunta nella sezione dipendenze del file build.gradle relativo all'app.

```
1 dependencies {
2
3     implementation 'com.multidots:fingerprint-auth:1.0'
4
5 }
```

5.1.4 Screenshot dell'applicazione

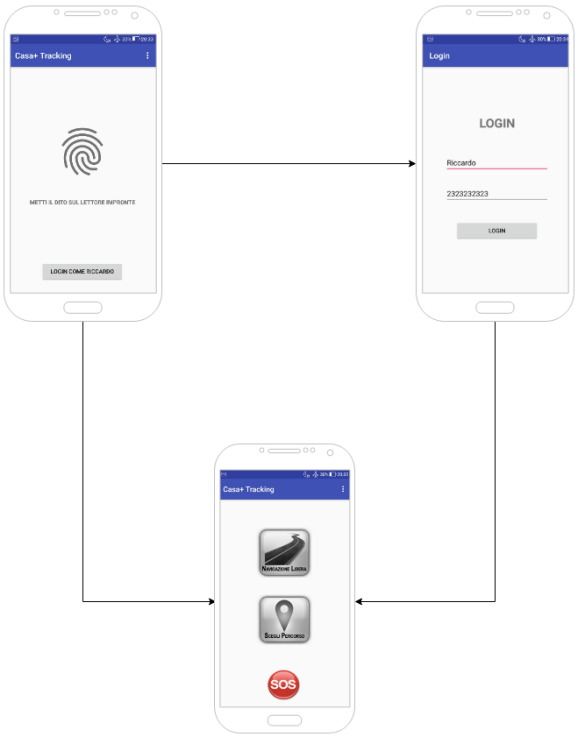

Figura 5.3: Screenshot della fase di login

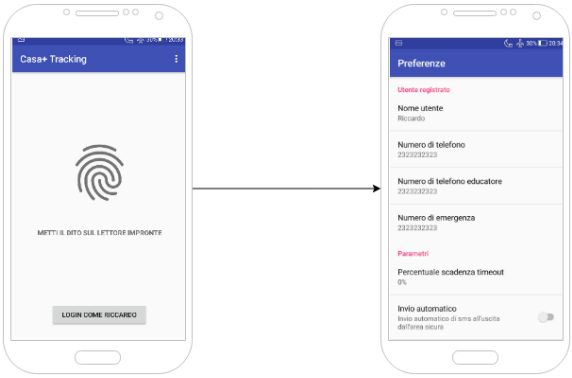

Figura 5.4: Screenshot della configurazione delle preferenze

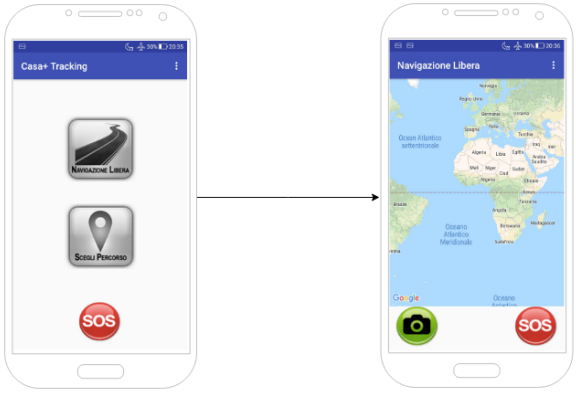

Figura 5.5: Screenshot della navigazione libera

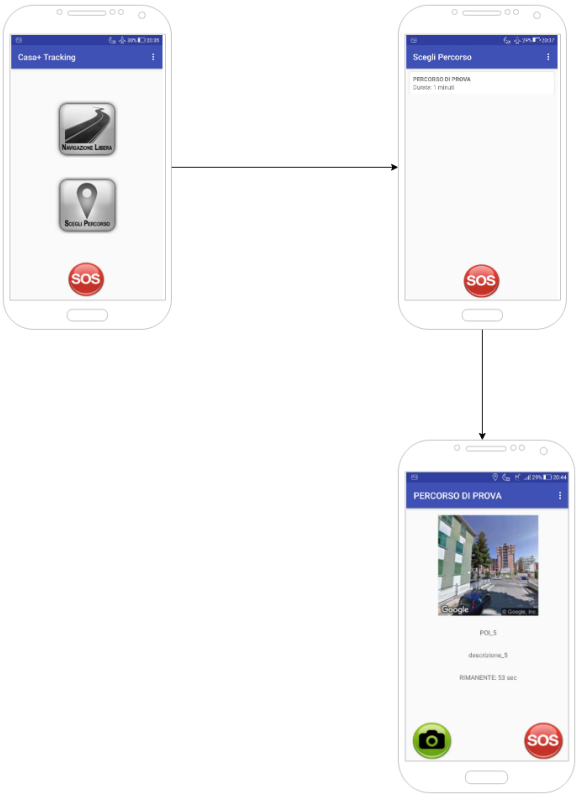

Figura 5.6: Screenshot della navigazione per immagini

5.2 Server di gestione notifiche Firebase

In questa sezione verrà mostrato il codice relativo all'implementazione del server di gestione delle notifiche Firebase. La pagina web è stata realizzata in HTML e CSS. Per la programmazione lato client si è utilizzato JavaScript e per la programmazione lato server si è utilizzato PHP.

In figura 5.7 verrà mostrata la struttura del filesystem del server.

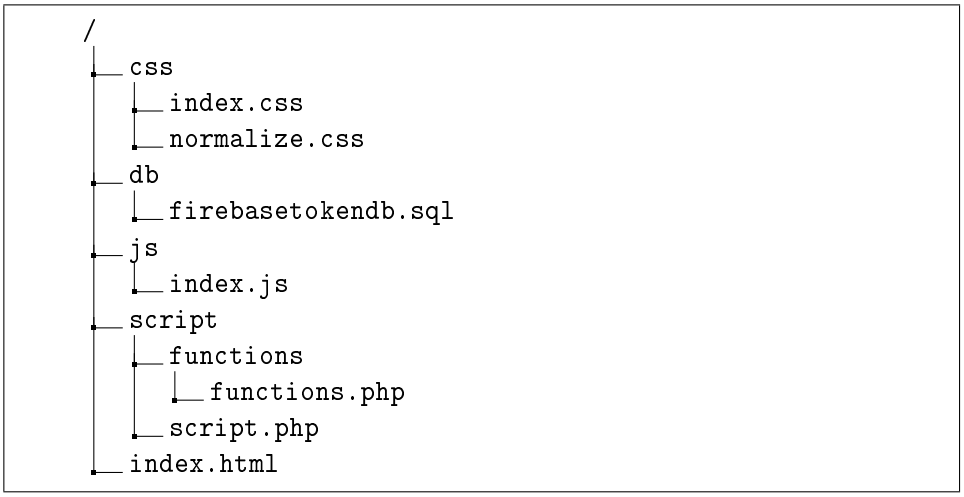

Figura 5.7: Struttura filesystem del server Firebase

- Nella cartella css ci sono i file CSS responsabile della grafica del sito.

- Nella cartella js c'è il file JavaScript che permette di far dialogare l'interfaccia web con la logica PHP.

- Nella cartella script ci sono i file PHP responsabili della logica del server.

5.2.1 Invio della notifica Firebase

Per poter "risvegliare" l'applicazione client sul telefono dell'utente target, è necessario inviare una notifica ad esso sfruttando la piattaforma Firebase.
Una volta che la notifica raggiungerà la piattaforma Firebase, sarà la piattaforma stessa ad occuparsi dell'inoltro al client giusto basandosi sul token univoco presente nei dati della notifica.

Ora verrà listato il codice PHP responsabile dell'invio della notifica alla piattaforma Firebase.

```php
<?php

function sendNotify($tokenID, $serverTokenID) {

  $url = "https://fcm.googleapis.com/fcm/send";

  //Settaggio del contenuto della notifica
  $message = "Hello World";

  //HTTP header
  $headers = array (
            "Authorization: key=$serverTokenID",
            "Content-Type: application/json"
    );

  //Settaggio array data
    $data = array (
                    "message" => $message
            );

    //Settaggio notifica di tipo data
    $fields = array (
            'to' => trim($tokenID),
            'data' => $data
    );
    $fields = json_encode ( $fields );

    //Invio notifica a piattaforma Firebase
  $options = array(
      'http' => array(
          'header'  => $headers,
          'method'  => 'POST',
          'content' => $fields
      )
```

```
35    );
36    $context = stream_context_create( $options );
37    $result = file_get_contents($url, false, $context);
38
39      return $result;
40  }
41
42  ?>
```

Vengono settati i parametri necessari richiesti per l'invio della notifica Firebase alla piattaforma ufficiale che si occupa di inoltrare la notifica al client target. (riga 5-25)
Tra essi vi è la variabile $tokenID che contiene il token univoco che rappresenta il client target e la variabile $serverTokenID, necessaria al corretto invio della notifica.
Il contenuto della notifica di tipo data non è importante e quindi può essere qualsiasi stringa, nel nostro caso è "Hello World" (riga 8).
Successivamente viene settato l'array $fields, che racchiude i dati da inviare al server e viene codificato in JSON. (riga 26)
Viene creata la richiesta con i dati settati in precedenza e viene infine inviata all'URL specificato (righe 29-37).
La funzione ritorna la risposta del server (riga 39).

Si è scelto una notifica di tipo data poichè essa ha la peculiarità che una volta ricevuta dal telefono Android, nel caso l'applicazione dovesse risultare disattiva, il sistema la attiva chiamando automaticamente il metodo onMessageReceived(). Tale metodo come si è già visto si occupa di avviare l'Activity responsabile della geolocalizzazione per controllare se l'utente si trova oppure no nell'area sicura.
Invece una notifica di tipo tradizionale creerebbe soltanto un messaggio nel menù a tendina del telefono Android in questione. Alla pressione di tale messaggio, l'app verrebbe aperta nella pagina MainActivity come qualsiasi altro avvio. Ma non sarebbe questo il modo in cui vorremmo gestire l'avvio automatico dell'app.

Bibliografia

[1] Google. *Location and Maps*. URL: https://developer.android.com/guide/topics/location/. (Ultimo accesso: 12/06/2018).

[2] Google. *Maps SDK for Android*. URL: https://developers.google.com/maps/documentation/android-sdk/intro. (Ultimo accesso: 12/06/2018).

[3] Google. *Developer Guides*. URL: https://developer.android.com/guide/. (Ultimo accesso: 12/06/2018).

[4] Google. *Firebase*. URL: https://firebase.google.com/docs/. (Ultimo accesso: 12/06/2018).

[5] Multidots. *Authenticate your users with fingerprint in the Android*. URL: https://medium.com/@multidots/authenticate-your-user-with-fingerprint-de876618ce8d. (Ultima modifica: 18/10/2016).