UNIVERSITA' DEGLI STUDI DI TRIESTE

Studi Superiori in Ingegneria Clinica – Higher Education in Clinical Engineering SSIC – HECE

Direttore: Prof. Agostino Accardo

Master di I livello in "Ingegneria Clinica"

Tesi di Master in

INFORMATICA SANITARIA

HERMIONE: IMPLEMENTAZIONE DI UN "DIGITAL ASSISTANT" MOBILE PER IL SUPPORTO DEGLI OPERATORI MEDICI E INFERMIERISTICI NELLA CURA DEI PAZIENTI

Relatore: Prof. Sara Marceglia

Studente del Master:

Roberto Prandin

Anno Accademico 2016/2017

Indice

1 Introduzione

Negli ultimi anni c'è stato un cambiamento sempre più consistente in sanità, nei modi di fare sanità e di concepire le strutture sanitarie.

È grazie a tutta una serie di leggi e normative, come ad esempio le "Linee guida per la dematerializzazione della documentazione clinica in laboratorio e in diagnostica per immagini. Normativa e prassi" (Ministero della Salute, novembre 2006), oppure le "Linee guida in tema materia di Fse e dossier sanitario" (Ministero della Salute, 2009, aggiornate poi nel 2015), che la tecnologia ha preso sempre più importanza all'interno dell'ambiente sanitario.

L'applicazione delle tecnologie informatiche nel settore sanitario ha come principale obiettivo la razionalizzazione dei processi assistenziali apportando sostanziali benefici sia in termini di costi che di qualità delle prestazioni sanitarie erogate. Tra i tanti, uno dei benefici più importanti è sicuramente l'aumento della sicurezza, della tracciabilità dei dati trattati, siano essi anagrafici o clinici. A tal proposito il fine che si cerca di ottenere è quello di ridurre gli errori clinici e migliorare i processi ospedalieri attraverso l'implementazione di sistemi informativi per strutture sanitarie che siano provvisti di un'architettura integrata e complessa.

E con l'arrivo nel panorama tecnologico di dispositivi sempre più portatili e performanti si aprono nuove frontiere nel supporto dell'operatore clinico: una nuova forma di assistenza digitale sempre a portata di mano.

1.1 Gli assistenti digitali in Medicina

Il personale sanitario ha bisogno di accedere a informazioni aggiornate ovunque e in qualsiasi momento, e un Personal Digital Assistant (PDA) ha il potenziale per soddisfare questi requisiti. Un PDA è uno strumento mobile che è stato ampiamente impiegato per vari scopi nelle pratiche di assistenza sanitaria e si prevede che il livello del suo utilizzo aumenti. Sfruttando applicativi adeguati, un PDA può essere qualificato come lo strumento necessario al personale e agli studenti in assistenza sanitaria. Un PDA è comodo da utilizzare in situazioni cliniche e sul campo per una rapida gestione dei dati e le informazioni possono essere sincronizzate con un PC. Tramite una rete wireless, i dati possono essere scambiati in qualsiasi momento e da qualsiasi luogo verso e da un PDA e la rete fornirà accesso immediato a tutti i tipi di dati clinici e amministrativi necessari. I PDA in passato venivano identificati come computer palmari, ora sostituiti coi moderni smartphone e tablet, che li superano in portabilità e potenza di calcolo.

Si stima che quasi 2 miliardi di persone nel mondo posseggano uno smartphone che consente loro l'accesso immediato a una varietà di applicazioni tecnologiche. Le applicazioni mobili sono state sviluppate come aiuto per migliorare quasi ogni aspetto della vita, e da qualche anno l'attenzione si è spostata particolarmente sulla sanità e il miglioramento delle condizioni di salute.

Ci sono molteplici dispositivi e applicativi che “catturano” e analizzano i nostri parametri vitali, ma è ancora lungo il percorso a livello di sviluppo per rendere i nostri devices realmente utili e insostituibili nell’ambito dell’ Healthcare.

A questo riguardo, MEDArchiver srl, azienda leader del settore “Hospital Information and Communication Systems (HICT)” per il mercato italiano, particolarmente competente in materia di bioingegneria e informatica medicale, da qualche tempo ha introdotto nel panorama digitale l’applicativo per dispositivi Apple Hermione.

L’app risulta proprio voler essere un assistente digitale Mobile in grado di raggiungere e accompagnare ovunque il personale medico e infermieristico. La scelta dell’azienda di spingere in questa direzione è corroborata da analisi e studi fatti sui diversi tipi di terminali utilizzati come supporto alla cura del paziente, in cui si è subito notato come l’uso di PC desktop e stazioni centrali, come anche postazioni fisse vicine al letto del paziente, in certe situazioni risulti inefficiente, andando ad aumentare le tempistiche di lavoro.

Non sembra quindi esagerato pensare che le Strutture Sanitarie nell’immediato futuro saranno pronte a far diventare smartphone e tablet, come anche smartwatch e dispositivi wearable, parte integrante della tecnologia di supporto al processo di cura del paziente.

1.2 Obiettivo del lavoro

L’obiettivo del lavoro di tesi è stato quello di creare un Personal Digital Assistant per dispositivi Android basandosi sull’applicazione Hermione di MEDArchiver srl, già

disponibile per dispositivi iOS. MEDArchiver srl, in una fase iniziale ha preferito sviluppare l'app solo per dispositivi Apple, in quanto più utilizzati in ambito sanitario e dagli operatori sanitari. Ma essendo Android il Sistema Operativo Mobile più diffuso al mondo, la necessità di un porting su questo SO era stringente, per andare ad accontentare ogni tipologia di utente e struttura.

Significativi sono stati anche i miglioramenti introdotti a livello di flessibilità dell'applicazione, di grafica, di gestione delle notifiche (rapidità di ricezione) e di sicurezza

Un progetto del genere ha ovviamente richiesto una prima fase di analisi delle tecnologie e dei sistemi protagonisti.

Dopo aver esaminato nel dettaglio il funzionamento e i flussi di lavoro generati dall'app iOS mi sono concentrato sul sistema di notifiche push di iOS in relazione a quello di Android e poi sugli aspetti tecnici legati all'integrazione tra l'app e il database centralizzato utilizzato come contenitore dei dati.

È iniziata a quel punto la fase di modellazione dell'applicazione seguita dalla fase di sviluppo e scrittura del codice attraverso l'IDE di riferimento Android Studio.

L'ultima fase, tutt'ora in corso, è orientata a testare il corretto funzionamento dell'applicazione, dalla sua interazione col database centralizzato e i moduli applicativi di MEDArchiver®, al sistema di invio/ricezione notifiche, fino alla correzione di errori e bug riscontrati nel corso dell'attività.

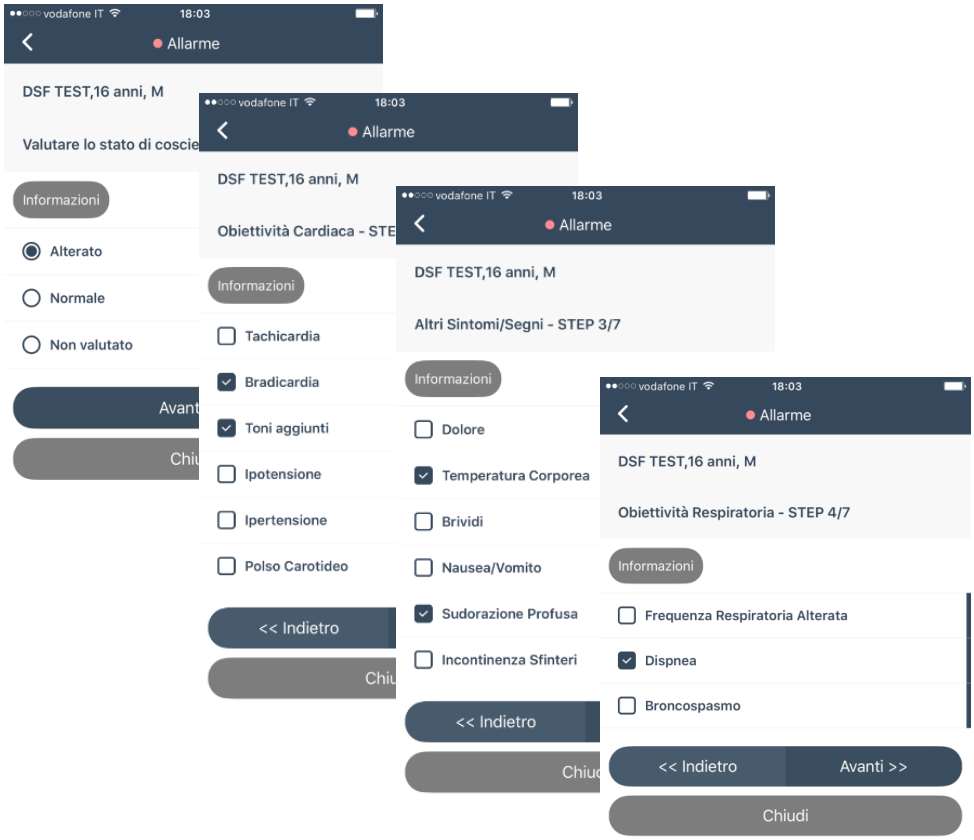


Figura 1: Hermione per iOS, esempi di schede di refertazione

2 Metodi

2.1 Ecosistema MEDArchiver®

Implementando soluzioni all-digital attraverso un'esclusiva piattaforma software proprietaria, MEDArchiver® copre tutte le aree ospedaliere: amministrativa, clinica, chirurgica, infermieristica ed assistenziale. Una peculiarità è sicuramente quella di essere in grado di integrare qualsiasi dispositivo, di qualsiasi produttore, con qualsiasi protocollo, tramite vari interfacciamenti mono o bidirezionali che consentono agli utenti, siano essi operatori o pazienti, di risparmiare tempo, evitare errori, archiviare tutte le informazioni in modo strutturato e fruibile e, molto importante, sfruttare al meglio la capacità diagnostica strumentale disponibile. A livello di sistemi informativi l'obiettivo è quello di rimuovere le distinzioni tra aree amministrative, cliniche e sanitarie, tra cartella clinica e sistema informativo ospedaliero. Nel mondo reale gli aspetti amministrativi, organizzativi e logistici vanno di pari passo con quelli diagnostici, clinici e assistenziali, ed allo stesso modo nel mondo virtuale di un sistema informativo, essi devono esistere e svolgersi in modo integrato ed efficiente. La piattaforma MEDArchiver® è quindi in grado di gestire tutti i processi all'interno di una struttura sanitaria, dall'accettazione alla dimissione, dalla prenotazione alla refertazione, alla fatturazione e rendicontazione, comprendendo anche tutti i moduli per la gestione verticale e specialistica del dato, convogliando tutto in un avanzato sistema di gestione unificata della cartella clinica.

MEDArchiver® è disponibile sia in configurazione stand-alone, sia client-server (più utilizzata), basata su un server a cui si connettono più postazioni di lavoro. Ogni stazione

è costituita da un elaboratore corredato dal software gestionale ed interfacciato con la strumentazione biomedica mediante canali di comunicazione basati su protocolli seriali o di rete. Le funzioni della piattaforma sono organizzate in sei gruppi logici, e a questi gruppi di funzioni e processi si aggiungono le funzionalità per l'interoperabilità dedicate ai dispositivi medicali ed ai sistemi informativi di terze parti:

- Funzioni generali: tutte le funzioni trasversali utilizzate da diversi processi e moduli della soluzione (es.: anagrafiche pazienti, gestione consensi, firma digitale, conservazione legale, statistiche sui dati, privacy, gestione magazzino ecc...);
- Ambulatori: funzioni cliniche e amministrative per la gestione del paziente ambulatoriale (es.: liste d'attesa, CUP, accettazione, ecc...);
- Degenze: tutte le funzioni cliniche e amministrative per la gestione del paziente prima, durante e dopo la degenza (es.: programmazione ricoveri, percorsi prericovero, ADT, gestione letti, order entry, SDO/DRG ecc...);
- Sale operatorie: funzioni per la gestione ottimale delle sale operatorie, dalle richieste di intervento fino alla chiusura dei verbali (es.: richieste intervento, programmazione, tempi operatori, consensi informati, checklist operatorie ecc...);
- Verticali di reparto: moduli specialistici per la gestione dedicata dei processi nelle diverse specialità cliniche (es.: Radiologia RIS, Laboratorio analisi LIS, Cardiologia, Pronto Soccorso PS ecc...);
- Flussi: totalità dei flussi per la rendicontazione verso soggetti pubblici, istituzionali e convenzionati (es.: generazione DRG/SDO, prestazioni ambulatoriali, farmaci ecc...).

Tra i moduli più importanti presenti nell'ecosistema MEDArchiver® sono da citare sicuramente i seguenti:

1. MED-Cardiology: permette di gestire le informazioni, il workflow e il ricovero con una piattaforma informatica completa ed intuitiva, implementando le più avanzate funzionalità gestionali, cliniche e scientifiche garantendo interoperabilità nel rispetto delle norme vigenti in materia di trattamento dei dati. Questo attraverso un evoluto approccio a tre direzioni: integrazione bottom-up, integrazione top-down e integrazione orizzontale. MED-Cardiology si integra perfettamente con qualsiasi dispositivo o sistema terzo (standard HL7, DICOM, SQL), offrendo così uno strumento per la gestione e ottimizzazione digitale del workflow ed un sistema di refertazione flessibile, espandibile e completo, ove tutti i dati vengono raccolti alla fonte (come per esempio i dispositivi medici che effettuano le misurazioni) e condivisi e gestiti con le varie cartelle cliniche mediche e il diario clinico;

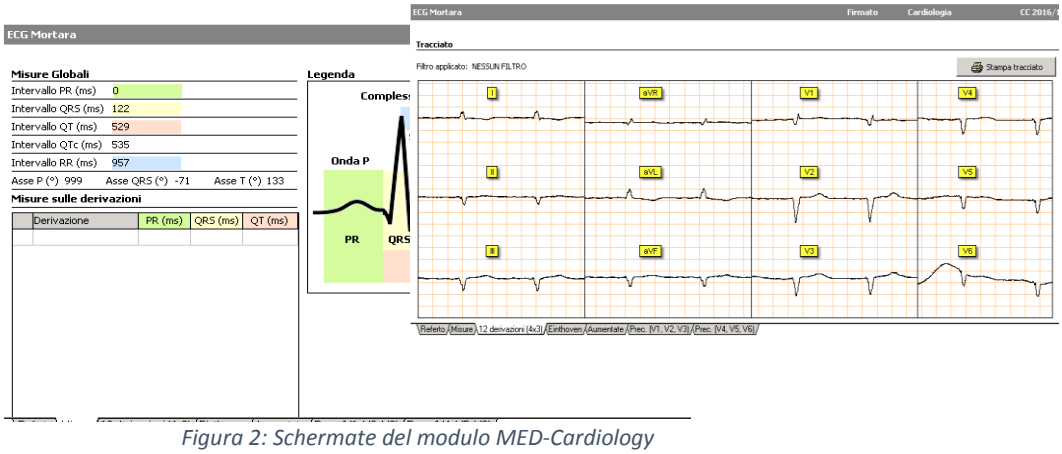


Figura 2: Schermate del modulo MED-Cardiology

2. OMM-Oncology Medication Managment: è la prima soluzione dedicata per la gestione completa del processo del farmaco chemioterapico, sviluppata a seguito di una lunga sperimentazione e raccolta di numerosi risultati in diversi

progetti europei. OMM è una piattaforma modulare, che consente l'implementazione graduale del sistema attraverso dei moduli verticali altamente specifici che si integrano in una soluzione completa e interagiscono tra loro in maniera trasparente. La soluzione prevede un sistema di prenotazione evoluto, un sistema di collaborazione basato su attività, una soluzione avanzata per la prescrizione e la somministrazione (con dosaggio e regole per controllo) e un sistema per la gestione automatica delle richieste della farmacia;

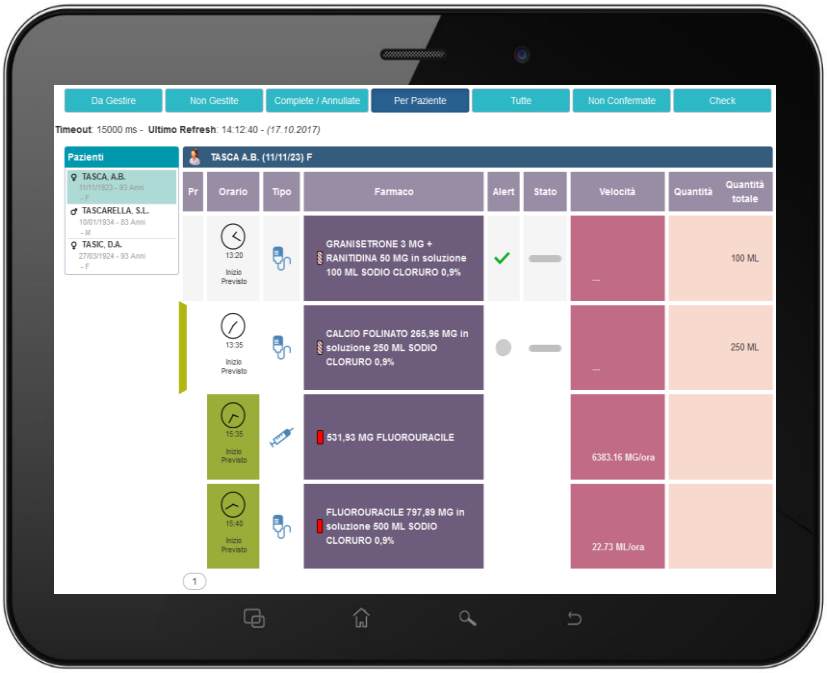


Figura 3: OMM Drug Monitor interfaccia web su tablet

3. MED-OR Management: è il modulo per la gestione informatizzata delle attività della sala operatoria, dalla richiesta d'intervento alla gestione di materiali e risorse, fino alla gestione della documentazione. Il sistema si basa su tre fondamentali principi: sicurezza, qualità e ottimizzazione;

Figura 4: Schermata modulo MED-Or

4. MED PHC- Private Hospital and Clinic: singola piattaforma concepita per assistere sia i processi amministrativi che quelli clinici in strutture multi-nodali private/convenzionate. Alle funzioni dei moduli amministrativi e alla struttura orizzontale di Cartella Clinica, vengono infatti affiancati i moduli verticali in grado di soddisfare le specifiche esigenze della realtà clinica. Dal punto di vista dell'interfaccia utente, il sistema si presenta completamente integrato, come un unico ambiente in cui le funzionalità disponibili sono rese accessibili a seconda dell'utente in modo contestualizzato e con gli strumenti di lavoro ottimali, dalle interfacce ad alto contenuto informativo per le attività di back office, a quelle semplificate ed estremamente efficienti per l'accesso in mobilità;
5. CLIO- Closed Loop Interlock Oncology: sistema di supporto decisionale in tempo reale dedicato all'Oncologia, che consente l'ottimizzazione e la personalizzazione

della terapia farmacologica grazie a funzioni evolute di monitoraggio automatico. Il sistema è integrato con la piattaforma OMM, e prevede l'interazione con l'operatore mediante un terminale touch screen collegato a tutti i sistemi informativi e a tutti i dispositivi medicali in grado di fornire informazioni sul paziente e sullo stato delle somministrazioni.

Figura 5: CLIO - Stazione con centrale di monitoraggio e pompe infusionali

2.2 Applicazioni Mobile

In questo contesto, vista la costante crescita a livello di utilizzo delle piattaforme mobile coi relativi devices, quali smartphone e tablet, sono stati sviluppati applicativi per facilitare ulteriormente l'esperienza utente nelle strutture. In particolar modo per la cardiologia è stato prodotta MED-Ecg, app per la visualizzazione dei referti ECG dei pazienti in mobilità.

Ma soprattutto, nata in parallelo al progetto CLIO, ma fruibile anche in altri contesti, è l'app Hermione, che si configura come un assistente digitale mobile in grado di raggiungere e accompagnare ovunque il personale medico e infermieristico. Hermione permette di ricevere sul proprio device notifiche push (alert) relative ad eventi che vanno a rappresentare tutte le attività identificate dal sistema di gestione OMM, dalla cartella clinica HEBB ed a tutti gli allarmi generati dal modulo CLIO, consentendo di intraprendere le azioni necessarie in base al tipo di notifica. Ma non solo, perché Hermione permette anche di effettuare e documentare decisioni cliniche e valutazioni mediche, notificando immediatamente a tutto lo staff le disposizioni necessarie. Tutto questo in mobilità, anche da remoto, nel palmo della mano, per reagire con tempestività e garantire elevati livelli di sicurezza.

Hermione non solo riceve notifiche, ma permette anche di gestirle. Nella schermata principale dell'app vengono infatti elencate tutte le notifiche ricevute. Le form di risposta e reazione ad ogni notifica sono configurabili, come anche è configurabile la consultazione delle informazioni contestuali alla notifica, e le informazioni gestite e documentate su Hermione vengono immediatamente trasferite al sistema che ha originato la segnalazione, con possibilità di documentare informazioni, decisioni e disposizioni direttamente in cartella clinica.

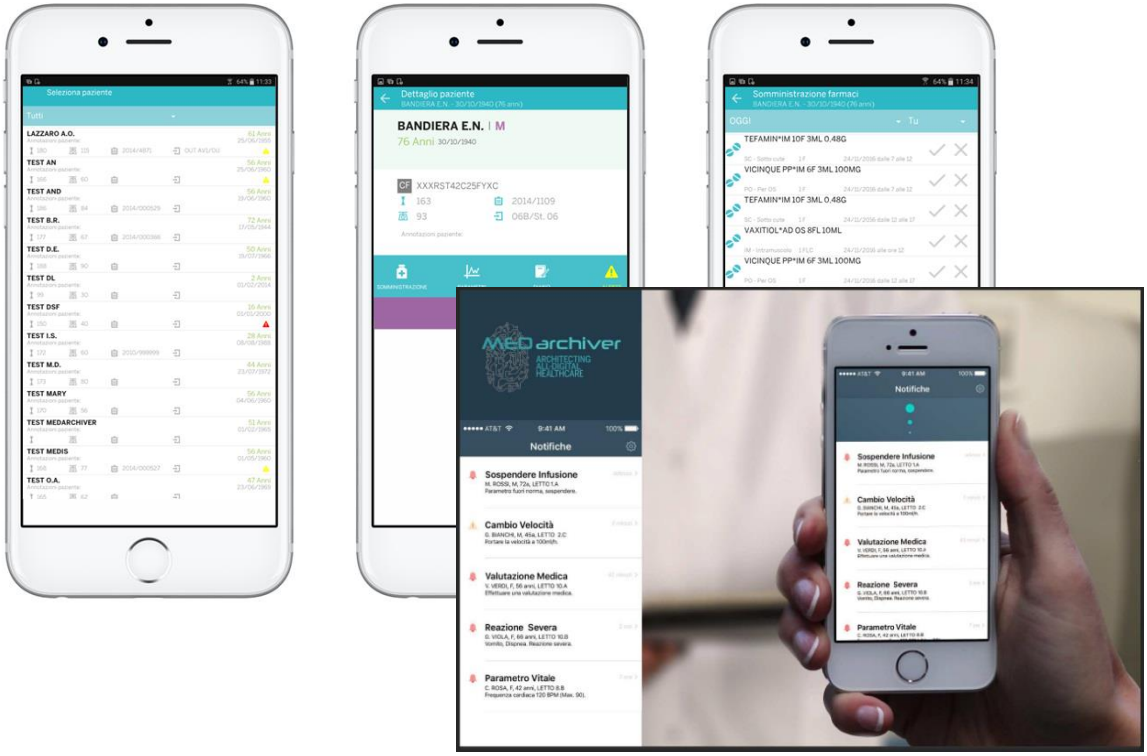


Figura 6: MED-Ecg e Hermione su dispositivi iOS

2.3 Android come Sistema Operativo Mobile

Per parlare di Android come SO Mobile non si può non iniziare da alcuni numeri che rendono abbastanza l'idea di quello che è un fenomeno in espansione. Il sistema operativo di Google ha infatti percorso tantissima strada dal lancio nel 2008 sul primo dispositivo HTC. Basti pensare che, secondo stime accurate, negli ultimi due anni circa 80 smartphone venduti su 100 montano al loro interno una qualsiasi release di Android. Proiettato su scala globale, questo dato significa un mercato potenziale di centinaia di milioni di persone e soprattutto che, verso la fine del 2018, il fatturato delle applicazioni per Android supererà quello generato dal principale concorrente, iOS di Apple.

2.4 Differenze tra Android e iOS

Le cause che stanno alla base di tale successo sono molteplici e chiariscono anche le principali differenze tra i due SO:

- Semplicità d'utilizzo: all'origine dei due sistemi operativi non vi era dubbio che iOS fosse molto più "user friendly" di Android, che ha dovuto arrivare almeno alla versione 4.0 (denominata "Ice Cream Sandwich") per essere paragonato al suo concorrente per quanto riguarda l'esperienza d'uso. Oggi non ci sono molte differenze tra i due principali sistemi operativi mobili. Entrambi si presentano con un'interfaccia molto semplice che evidenzia l'utilità di ogni singola applicazione. Se si considerano l'aspetto e le impostazioni della schermata home, gli smartphone Android, offrono un maggiore controllo rispetto a iOS sul sistema, sulle applicazioni e la schermata iniziale. Il livello della personalizzazione utente è praticamente illimitato ed è uno dei cardini di una filosofia completamente opposta rispetto a quella di iOS. In Android l'utente è padrone del sistema e può fare praticamente ciò che vuole, anche se, talvolta, per utenti meno esperti, questo può portare ad una sensazione di disorientamento, di non sapere cosa stia succedendo. Al contrario, in iOS "il sistema è padrone di se stesso": l'utente viene infatti guidato nella quasi totalità delle azioni che si possono svolgere, e questo da un lato è positivo perché è praticamente impossibile perdersi al suo interno, ma dall'altro a volte non permette di esprimere tutte le potenzialità dei devices
- Frammentazione: è la principale differenza tra i due sistemi. Android è infatti un SO open source pensato per poter essere installato sulle più disparate categorie

di dispositivi (oltre a smartphone e tablet esempi ne sono gli smartwatch, Android Car, l'IoT e le smartTv) perché riesce ad adattarsi alle varie caratteristiche dei dispositivi, a dimensioni e risoluzioni diverse degli schermi ed essendo appunto aperto, le varie ditte possono personalizzare la loro versione del sistema per adattarlo ai propri devices. Tutto questo da l'opportunità di scegliere tra una vasta gamma di prodotti, da fasce di prezzo molto basse a molto alte. Il grande svantaggio risiede però nella scarsa ottimizzazione (tra hardware e software) e nelle lunghe attese per il rilascio degli aggiornamenti del sistema, in quanto quando Google rilascia una nuova release di Android questa non può essere immediatamente installata su ogni device, ma le rispettive ditte produttrici devono lavorare all'adattamento della loro versione del SO aggiornato. Ovviamente in Apple non esiste frammentazione, in quanto il sistema è proprietario, ed anzi, probabilmente l'aspetto dove l'azienda di Cupertino è inarrivabile è proprio l'ottimizzazione tra hw e sw, di produzione propria.

- Sicurezza: il problema di Android non è la sua sicurezza intrinseca, ma il fatto che Google è meno rigida di Apple su quali applicazioni accettare o meno nel suo App Store. Il modo migliore per proteggere il proprio dispositivo Android è scaricare applicazioni solo dal Google Play Store; ma anche così, Google segnala che lo 0,16% di tutte le applicazioni contiene malware. Anche per iOS esistono applicazioni che contengono malware, ma i devices Apple sono intrinsecamente più sicuri, questo grazie anche al rilascio costante di patch di sicurezza che con certezza saranno installate nei vari prodotti, non essendoci il problema della frammentazione.

- Salvaguardia delle risorse: essendo progettato per sistemi embedded (sitemi progettati appositamente per una determinata applicazione), storicamente dotati di poche risorse di memoria, Android ha avuto sin da subito uno spirito parsimonioso. Senza far perdere fluidità alla user-experience, il sistema è particolarmente bravo nel distruggere e ricreare parti dell'applicazione in maniera del tutto impercettibile all'utente. Chi dovrà fronteggiare questo atteggiamento sarà il programmatore.

2.5 Elementi fondamentali di programmazione Android

Android come Sistema Operativo è figlio di Linux, ed accoglie in sé tutto il meglio di quanto è stato ideato per supportare lo sviluppo del web, desktop e mobile sia in termini di pattern progettuali che di librerie software. A livello di programmazione, Android si basa su Java (per le funzioni applicative) e XML (per realizzare la parte grafica), quindi per iniziare a sviluppare applicazione sono fondamentali i seguenti strumenti software:

- Un JDK, il kit di sviluppo per la tradizionale programmazione Java;
- Un IDE (ambiente di sviluppo integrato) che includa possibilmente tutti gli strumenti necessari al programmatore. La mia scelta è ricaduta su Android Studio, la soluzione ufficiale e quindi preferibile;
- Android SDK, il vero pacchetto di strumenti che, usati tramite Android Studio, permettono di realizzare applicazioni.

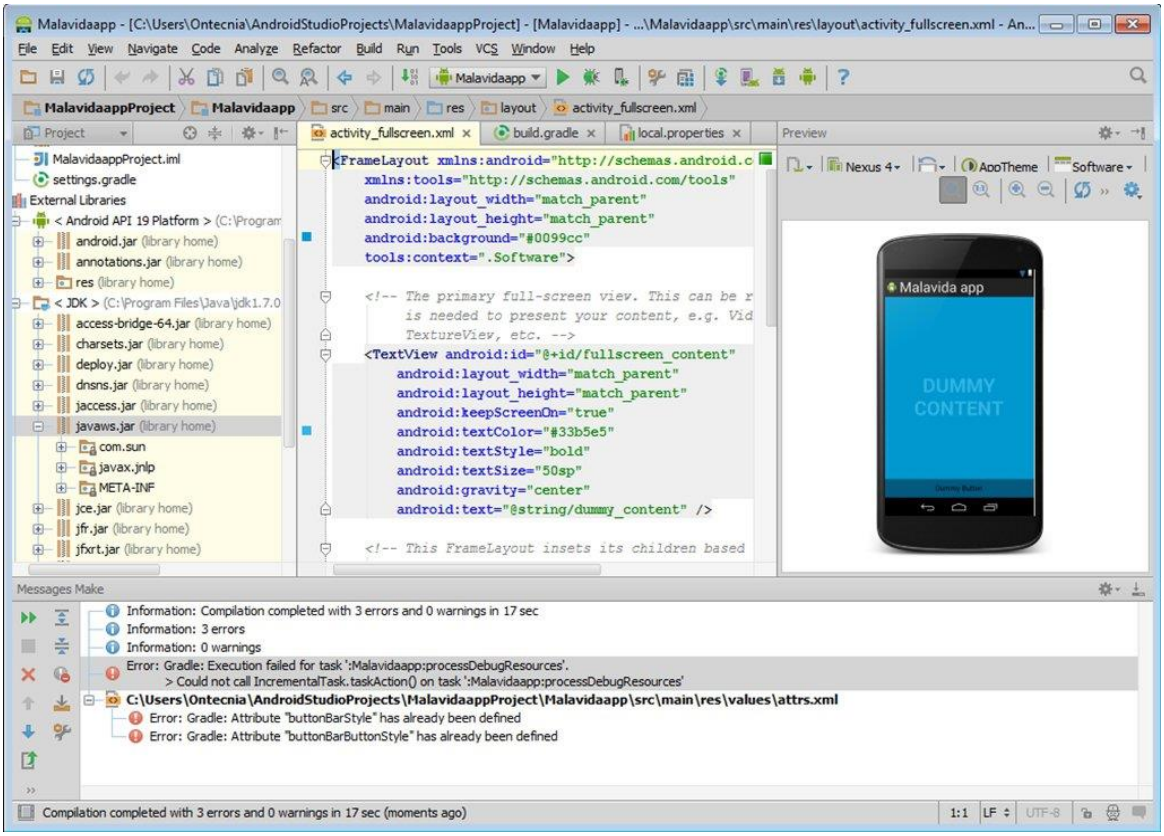


Figura 7: Schermata di un progetto di lavoro su Android Studio

All'inizio di ogni nuovo progetto, l'IDE propone diversi template che rappresentano i tipi più in voga di applicazioni e la configurazione è affidata ai file build di Gradle. Quest'ultimo è uno strumento di build automation; un contributo utilissimo che offre Gradle è la gestione delle dipendenze in stile Maven. Quando ci sarà bisogno di integrare librerie di sviluppo prodotte da Android o da sviluppatori di terze parti si potrà inserire direttive che permetteranno di recuperarle direttamente in rete tramite "coordinate" costituite da group id, artifact id e versione. Funzioni utili che aiutano lo sviluppatore durante la fase di scrittura del codice sono:

- La presenza di un emulatore AVD (Android Virtual Device) Manager per verificare istantaneamente il funzionamento del codice realizzato;
- Inline debugging, funzionalità che rende più immediata l'ispezione del codice durante il debug, affiancando alle righe di linguaggio Java i valori ed i riferimenti collegati agli oggetti;
- Un robusto sistema di logging per tracciare ogni tipo di errore e comportamento anomalo dell'app durante le fasi di test.

A livello di struttura, ogni applicazione Android, indipendentemente dalla finalità che si prefigge, affida le sue funzionalità a quattro tipi di componenti. Si tratta di Activity, Service, Content Provider e BroadcastReceiver che esistono affinchè la nostra applicazione possa integrarsi alla perfezione nell'ecosistema Android. In un'applicazione completa possiamo trovare tutti e quattro questi componenti oppure solo alcuni (l'unico componente fondamentale è l'Activity):

- Un'Activity è un'interfaccia utente. Ogni volta che si usa un'app generalmente si interagisce con una o più "pagine" mediante le quali si consultano dati o si immettono input. Dal momento in cui un'Activity compare sullo schermo al momento in cui scompare essa passa attraverso una serie di stati, il cosiddetto ciclo di vita. L'ingresso o l'uscita da uno di questi stati viene notificato con l'invocazione di un metodo di callback da parte del sistema. Quando un'Activity va in esecuzione per interagire direttamente con l'utente vengono obbligatoriamente invocati tre metodi:

- onCreate(): l'Activity viene creata. Il programmatore deve assegnare le configurazioni di base e definire quale sarà il layout dell'interfaccia;
- onStart(): l'Activity diventa visibile. È il momento in cui si possono attivare funzionalità e servizi che devono offrire informazioni all'utente;
- onResume(): l'Activity diventa la destinataria di tutti gli input dell'utente.

Android pone a riposo l'Activity nel momento in cui l'utente sposta la sua attenzione su un'altra attività del sistema, ad esempio apre un'applicazione diversa, riceve una telefonata o semplicemente, anche nell'ambito della stessa applicazione, viene attivata un'altra Activity. Anche questo percorso, passa per tre metodi di callback:

- onPause() (l'inverso di onResume()): notifica la cessata interazione dell'utente con l'Activity;
- onStop() (contraltare di onStart()): segna la fine della visibilità dell'Activity;
- onDestroy() (contrapposto a onCreate()): segna la distruzione dell'Activity.

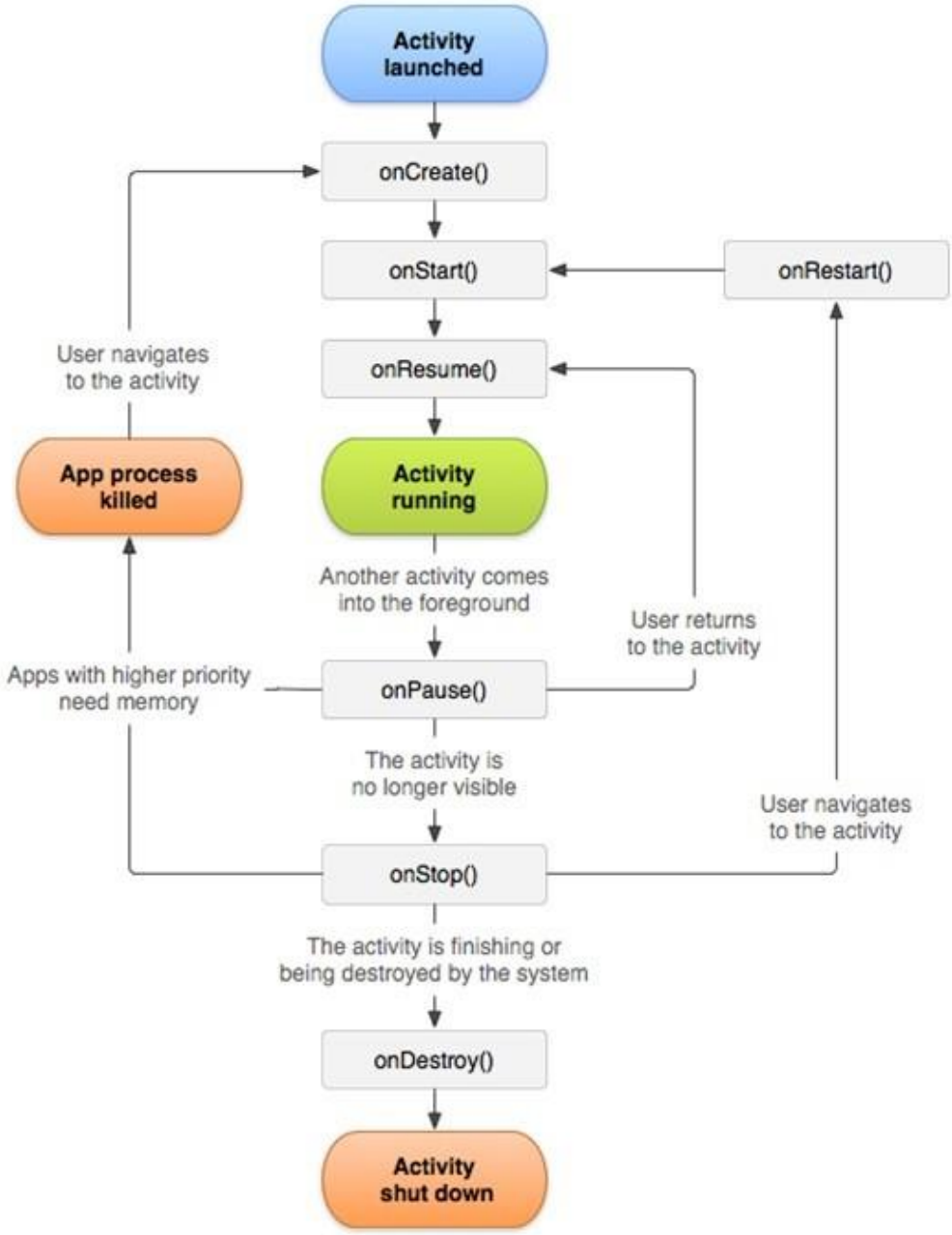


Figura 8: Diagramma che rappresenta il ciclo di vita di una Activity

- Un Service svolge un ruolo, se vogliamo, opposto all'Activity. Infatti rappresenta un lavoro, generalmente lungo e continuato, che viene svolto interamente in background senza bisogno di interazione diretta con l'utente. I Services hanno un'importanza basilare nella programmazione proprio perchè spesso preparano i

dati che le Activities devono mostrare all'utente permettendo una reattività maggiore nel momento della visualizzazione;

Figura 9: Diagramma che rappresenta il ciclo di vita di un Service

- Un Content Provider nasce con lo scopo della condivisione di dati tra applicazioni. Questi componenti permettono di condividere, nell'ambito del sistema, contenuti custoditi in un database, su file o reperibili mediante accessi in Rete. Tali contenuti potranno essere usati da altre applicazioni senza invadere lo spazio di memoria;
- Un Broadcast Receiver è un componente che reagisce ad un invio di messaggi a livello di sistema (broadcast) con cui Android notifica l'avvenimento di un determinato evento, ad esempio l'arrivo di un SMS o di una chiamata o sollecita l'esecuzione di azioni. Questi componenti sono particolarmente utili per la gestione istantanea di determinate circostanze speciali;
- Esiste poi un altro componente codificato come una normale classe Java che permette ad un componente qualsiasi di attivarne un'altro mediante apposite

invocazioni di sistema. Un Intent sottintende un potente strumento di comunicazione di Android per passare dati ad esempio da un'Activity ad un'altra oppure per chiamare dalla stessa Activity l'Activity di un'altra applicazione (uno dei casi più utilizzati è avere il link di un sito web all'interno di un'Activity e fare in modo che al tocco venga richiamato il browser di sistema con il quale aprire il suddetto link).

Per realizzare il porting di Hermione sono stati usati in gran parte Activities, Broadcast Receiver e Intent.

2.6 Gestione della sicurezza informatica nelle notifiche di Android

Essendo Hermione sostanzialmente un'app basata sulla ricezione di notifiche (sia push che pull) si è dimostrato necessario svolgere un lavoro di approfondimento sul funzionamento del sistema di notifiche in Android, soprattutto dal punto di vista della sicurezza. La notifica push è una tipologia di messaggistica istantanea con la quale il messaggio perviene al destinatario senza che questo debba effettuare un'operazione di scaricamento (modalità pull). Il sistema di notifiche di Google utilizza un servizio esterno chiamato Firebase Cloud Messaging (che ha sostituito da poco il Google Cloud Messaging) per la gestione delle notifiche push. FCM viene integrato nel sistema operativo dei vari devices tramite Google Play Services (si tratta di un'applicazione di sistema che controlla tutti i servizi legati a Google, ed è molto importante perché permette di tenere i devices aggiornati e sicuri). gli attori principali del sistema di notifiche sono tre:

- L’applicazione installata sul device;
- L’applicazione esterna, denominata provider, che si occupa di creare e mandare le notifiche;
- FCM.

Il funzionamento è semplice e si dispiega in questi passaggi:

1. Alla prima apertura dell’applicazione (non viene chiesto il permesso per ricevere notifiche push come su iOS ma va in automatico), il device invia il suo sender_id a FCM, che consiste nel project number del progetto Google legato all’applicazione (progetto creato tramite Google Developers Console, strumento per sviluppatori);
2. FCM genera e invia il device token del device (un codice alfanumerico che identifica in maniera univoca la coppia app + device);
3. A questo punto l’applicazione invia al provider il suo device token;
4. Il provider appena ricevuto il device token (dovrà essere tenuta una lista coi vari token) potrà chiedere in qualsiasi momento a FCM di inviare una notifica al device inviando una POST request contenente il device token , la sua server_key e la notifica;
5. FCM invia la notifica al device.

I tipi di token usati e scambiati sono i seguenti:

- sender_id: conservato nell’applicazione e inviato in fase di registrazione a FCM, consiste nel project number del progetto Google legato all’applicazione;

- device token: codice che FCM restituisce all'app e che identifica univocamente la coppia app + device;
- server_key: chiave di autenticazione che possiede il provider.

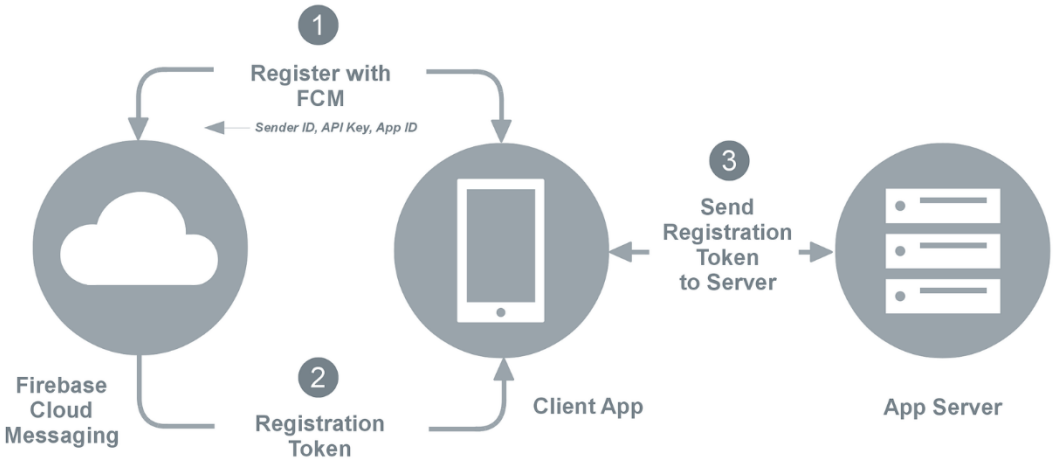


Figura 10: FCM, servizio di notifiche push di Google

Quando viene inviata una notifica ad un device Android essa viene intercettata da un Broadcast Receiver e visualizzata. Il provider invia a FCM le notifiche da inviare ai devices tramite una POST request indirizzata all'URL https://fcm.googleapis.com/fcm/send . La richiesta deve contenere:

- Header:
    - Uno di tipo authorization e di valore key uguale alla server_key del Provider;
    - L'altro che specifica il tipo di contenuto;
- Payload: deve essere al massimo di 4kb e contiene un oggetto JSON con due proprietà:

- device token del device al quale inviare la notifica;
- data: contenente le informazioni da trasmettere all'app.

FCM permette di scegliere che protocollo utilizzare per l'invio delle notifiche: oltre al più famoso https è possibile utilizzare anche xmpp. Quando il Provider invia un messaggio a FCM e riceve un response da esso, non è detto che il messaggio sia già stato consegnato, ma solo accettato per la consegna. Se il dispositivo a cui è indirizzata la notifica è acceso e "online" il messaggio viene consegnato; se è in modalità "doze" (sistema di gestione dei consumi che mette in deep sleep il terminale ibernando le applicazioni in uso e che si attiva automaticamente non appena il dispositivo viene scollegato dall'alimentazione esterna, in condizioni di non utilizzo e a schermo spento) la notifica diventa di bassa priorità (almeno che la priorità non sia settata ad alta) la notifica viene tenuta in FCM finchè il dispositivo non esce dalla suddetta modalità (tipicamente sbloccandolo); se il device è spento, la notifica resterà sui server di FCM finchè non tornerà online e verrà consegnata. La notifica rimane sui server per un massimo di quattro settimane, a meno che non sia settata la condizione time_to_live che permette di decidere la durata dello storage.

2.7 Realizzazione di Hermione su sistema operativo Android

Il porting di Hermione realizzato risulta compatibile su tutti gli smartphone che montano versioni di Android superiori alla 4.0. I linguaggi utilizzati sono stati Java per la parte applicativa e XML per quella puramente grafica. Dal punto di vista dei contenuti,

l'applicazione è basata su quattro Activities principali che vanno poi a richiamare tutta una serie di Activities secondarie, classi e servizi utili a svolgere le funzioni per cui l'applicazione è stata creata. A supporto di tutte queste Activities e classi sono state incluse nel progetto alcune librerie esterne dalle quali sono stati presi o sovrascritti i metodi che costituiscono il cuore pulsante dell'app.

2.7.1 Librerie utilizzate

Per realizzare tutte le funzioni di Hermione è stato necessario importare nel progetto delle particolari librerie esterne, quali:

- com.google.firebase.messaging: libreria creata da Google per mettere a disposizione degli sviluppatori classi e metodi adibiti alla ricezione e invio di notifiche push. Di particolare rilevanza le classi Java FirebaseInstanceId e FirebaseMessagingService. La prima mediante i metodi getInstance() e getToken() genera il device token (come visto prima) che serve per riconoscere il device a cui mandare le notifiche; la seconda è una classe che tramite i metodi onMessageReceived() e handleDataMessage() permette di gestire il contenuto in JSON della notifica arrivata (viene fatta una distinzione tra quando la notifica arriva mentre l'applicazione è in foreground, cioè in primo piano, e quando invece è in background, nel qual caso viene utilizzato il metodo interno sendNotification());
- Android Asynchronous Http Client: libreria open source basata sulle librerie HttpClient di Apache scaricabile dal sito http://loopj.com/android-async-http/ per

la gestione dei servizi REST del protocollo https; Hermione infatti, per dialogare col Provider (in questo caso un modulo denominato ATG) utilizza delle classiche chiamate GET (per il recupero di informazioni), POST e PUT (per la modifica di dati e informazioni). La caratteristica di questa libreria è che fa in modo che tutte le richieste vengano eseguite al di fuori del thread dell'interfaccia principale dell'app, ma qualsiasi logica di callback viene eseguita sullo stesso thread responsabile della chiamata tramite la classe Handler di Android, garantendo flessibilità e ottimizzazione dei processi. Questa libreria inoltre permette di gestire in maniera ottimale i response delle chiamate in formato JSON;

- cz.msebera.android:httpclient:4.3.6 : libreria anch'essa basata su librerie HttpClient di Apache che aggiunge funzionalità mancanti alla libreria precedente, soprattutto per quanto riguarda l'intestazione da associare alle varie chiamate; ce n'è infatti qualcuna che necessita di allegare alla chiamata nell'intestazione un oggetto JSON, pratica resa possibile dalla classe StringEntity.

2.7.2 Activities principali

Questa versione di Hermione ruota attorno a quattro Activities principali, tutte le altre Activities e classi sono di supporto:

- LoginActivity: rappresenta la schermata iniziale dell'applicazione che permette di connettersi al sistema. Al primo accesso verrà richiesto di settare l'indirizzo del Provider a cui l'app si connetterà per inviare le richieste; per svolgere questa operazione cliccando sull'icona in alto a destra si verrà rimandati tramite Intent ad

una delle Activities secondarie in cui si potrà inserire e salvare l'indirizzo corretto: LoginSettingsActivity. Una volta inserito l'indirizzo si ritornerà all'Activity di login che contiene due EditText per poter inserire username e password, una Textview che visualizza il nome del device e un Button per eseguire il login. Una volta premuto il bottone verrà eseguita la parte di codice che caratterizza questa Activity:

- Viene fatto un controllo per verificare se esiste una connessione di rete (tramite le classi ConnectivityManager e NetworkInfo);
- In caso affermativo si esegue un altro controllo per verificare che l'indirizzo a cui si cercherà di connettersi sia corretto; successivamente viene istanziato un oggetto della classe AsyncHttpClient che grazie al metodo post() invierà una POST request contenente gli elementi utili ad effettuare l'autenticazione nel sistema (in questo caso tipicamente username, password, nome device e device token);
- Al metodo post() deve essere passato come argomento un nuovo oggetto della classe JsonHttpResponseHandler che gestisce la risposta del sistema: se è negativa verrà eseguito il metodo onFailure() che riporta un messaggio di errore con la spiegazione del perché la richiesta non è andata a buon fine; se la risposta è positiva verrà eseguito il metodo osSuccess() che gestisce l'oggetto JSON ritornato come response dal Provider;
- Queste parti di codice sono incastonate in costrutti try {…} catch{…} utilizzati per catturare le varie eccezioni ed errori dovuti principalmente

alla gestione degli oggetti JSON ed evitare malfunzionamenti e chiusure inaspettate dell'app.

```
//setto il listener sul bottone login
bottoneLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {


        ConnectivityManager connectivityManager = (ConnectivityManager)
getApplicationContext().getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();

        Log.e(TAG, "conn : " + activeNetworkInfo);


        //viene fatto un controllo sulla connessione internet: se non c'è si avvisa con un toast se
no
        //si va avanti
        if (activeNetworkInfo == null) {
            Toast.makeText(getApplicationContext(), "Nessuna connessione di rete",
Toast.LENGTH_LONG).show();
        } else {


            //è scritta tutta la request perchè col metodo login della classe Authentication sembra
non
            //funzionare

            //controllo: se l'indirizzo è giusto parto con la procedura di login, se è sbagliato
spara un toast
            if (settings.getString("indirizzo-atg", "indirizzo").equals("https://atg-
dev.medarchiver.com") || settings.getString("indirizzo-atg",
"indirizzo").equals("https://atg.medarchiver.com")) {

                //aggiungo all'indirizzo la sezione /login
                String url = settings.getString("indirizzo-atg", "indirizzo") + "/login";

                //dichiaro un nuovo oggetto della classe asynchttpclient, che fa parte di una
libreria esterna
                //che mi permette di gestire connessioni con la rete per fare POST request ecc...
                AsyncHttpClient client = new AsyncHttpClient();

                //creo la progress dialog che verrà visualizzata una volta premuto login
                final ProgressDialog pDialog = new ProgressDialog(LoginActivity.this);


                //creo l'oggetto json che conterrà i dati da passare tramite POST request
                final JSONObject json = new JSONObject();
                try {
                    json.put("dev-id", deviceId);
                    Log.e(TAG, "TOKEN : " + deviceId);
                    json.put("dev-name", Build.MODEL.toString());
                    json.put("username", username.getText().toString());
                    json.put("password", password.getText().toString());

                    //inizializzo i dati che andranno passati tramite POST, cioè gli headers e il
payload
                    String contentType = getResources().getString(R.string.content_type);
                    String jsonString = json.toString();
                    //la stringentity permette di passare l'oggetto json convertito in stringa
                    StringEntity entity = new StringEntity(jsonString, "UTF-8");
                    //molto importante è la conversione in caratteri che http legge
                    entity.setContentType(contentType);
                    entity.setContentEncoding(new BasicHeader(HTTP.CONTENT_ENCODING,
                            contentType));

                    //invia la richiesta
                    client.post(getApplicationContext(), url, entity, "application/json", new
JsonHttpResponseHandler() {

                        //ora viene fatto l'override di tutti i metodi del gestore della richiesta
jsonhttpresponsehandler
                        @Override
                        public void onStart() {

                            //quando parte la richiesta viene visualizzata la process dialog che
resta in attesa finchè
                            //non c'è responso
                            pDialog.setMessage("Loading...");
                            pDialog.setIndeterminate(false);
```

```
pDialog.setCancelable(true);
                pDialog.show();
            }

            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONObject obj) {
                try {

                    //la process dialog scompare
                    pDialog.dismiss();

                    //se la richiesta va a buon fine riceveremo un response in json che verrà
convertito
                    //in stringhe

                    String response = obj.toString();
                    Log.e(TAG, "response: " + response);
                    String access_token = obj.get("access_token").toString();

                    //da lasciare se no non fa il login
                    String expires_in = obj.get("expires_in").toString();
                    String token_type = obj.get("token_type").toString();
                    String scope = obj.get("scope").toString();

                    String refresh_token = obj.get("refresh_token").toString();

                    //viene aggiornato il file preferenze
                    editor.putString("access_token", access_token);
                    editor.putString("refresh_token", refresh_token);
                    editor.putString("status", "LOGGED");
                    editor.apply();


                    Intent intent = new Intent(getApplicationContext(), MainActivityMod.class);
                    startActivity(intent);
                    finish();


                } catch (JSONException e) {
                    e.printStackTrace();
                }

            }

            @Override
            public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject
errorResponse) {


                try {
                    //se la richiesta non va a buon fine viene visualizzato un toast che avverte che
                    //username o password sono errati

                    pDialog.dismiss();
                   // String codice = errorResponse.get("error-code").toString();
                   String error = errorResponse.toString();
                    // String error = errorResponse.get("error").toString();
                    Toast.makeText(getApplicationContext(), error, Toast.LENGTH_LONG).show();

              } catch (NullPointerException e) {
                    e.printStackTrace();
                }
            }
        });
    } catch (JSONException e) {
        e.printStackTrace();
    }

} else {

    Toast.makeText(getApplicationContext(), "Indirizzo non corretto", Toast.LENGTH_LONG).show();

}

     }
    });
}
```

Figura 11: parte di codice di LoginActivity, l'ascoltatore sul bottone di Login

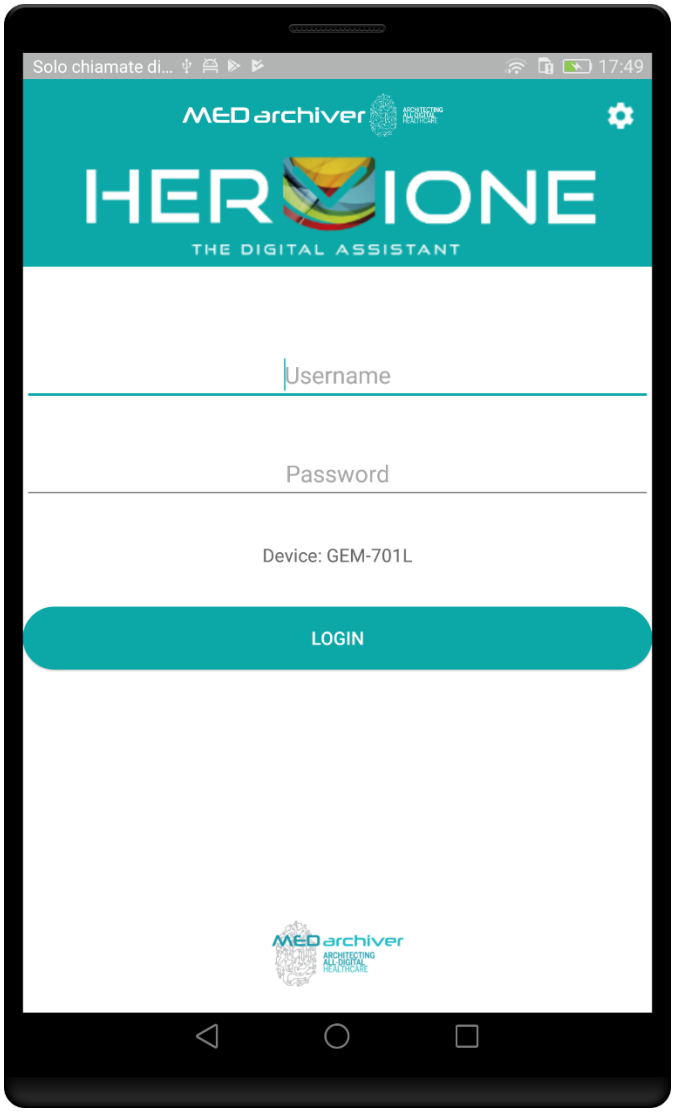


Figura 12: Screenshot relativo a LoginActivity

Una volta eseguita con successo la procedura di autenticazione tramite Intent l'utente viene reindirizzato alla seconda Activity principale;

- MainActivityMod: rappresenta la schermata dell'applicazione che contiene la lista (realizzata con una ListView riempita da un Adapter personalizzato) degli eventi/notifiche inviati al device (o profilo operatore); il layout grafico associato a questa Activity è particolare perché si tratta di uno SwipeRefreshLayout, elemento che consente di svolgere un'operazione (in questo caso una GET request per il

refresh della lista) eseguendo uno swipe verso il basso col dito sul touchscreen. Gli elementi fondamentali che costituiscono questa Activity sono:

- Un oggetto della classe Authentication che viene istanziato per usare il metodo getAccessToken() per ricevere ad ogni richiesta le nuove credenziali di accesso al sistema;
- Una serie di oggetti che vanno a creare e riempire la lista (se non ci sono eventi in lista viene visualizzato un avviso);
- Un oggetto della classe BroadcastReceiver che è sempre "in ascolto" e quando arriva una notifica gestita dai metodi della classe MyFirebaseMessagingService ne capta il contenuto andando ad aggiornare in tempo reale la lista;
- Un oggetto della classe NotificationAll che viene istanziato per poter utilizzare il metodo getAll(); questo metodo fa una chiamata GET al provider (sempre tramite un oggetto della classe AsyncHttpClient) per ricevere i dati di tutti gli eventi/notifiche associate al corrispondente device, che poi verranno utilizzati per creare la lista;
- La definizione della classe MessageAdapter; un Adapter è quell'elemento di codice che lega gli oggetti della lista definiti dal codice con i rispettivi elementi grafici; va a gestire la costruzione della lista e il suo popolamento con la possibilità di applicarvi dei filtri;
- Elementi quali SearchView, SearchManager e Filter che vanno a realizzare due funzioni molto importanti: la ricerca tra gli elementi della lista (accessibile tramite un'icona in alto a destra dello schermo) e la possibilità

di impostare dei filtri che permettono di visualizzare solo alcuni elementi della lista (anche questa funzione accessibile da un menù in alto a destra dello schermo).

```
//setto quello che bisogna fare quando viene fatto uno swipe dall'alto verso il basso sulla lista
mySwipeRefreshLayout.setOnRefreshListener(
        new SwipeRefreshLayout.OnRefreshListener() {
            @Override
            public void onRefresh() {

                // in questo metodo metterò getAccessToken e getEvents
                myUpdateOperation();
                Log.e(TAG, "access token : " + settings.getString("access_token", "token"));
                Log.e(TAG, "refresh token : " + settings.getString("refresh_token", "token"));

            }


            private void myUpdateOperation() {

                ConnectivityManager connectivityManager = (ConnectivityManager)
getApplicationContext().getSystemService(Context.CONNECTIVITY_SERVICE);
                NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();

                Log.e(TAG, "conn : " + activeNetworkInfo);


                //viene fatto un controllo sulla connessione internet: se non c'è si avvisa con un
toast se no
                //si va avanti
                if (activeNetworkInfo == null) {
                    Toast.makeText(getApplicationContext(), "Nessuna connessione di rete",
Toast.LENGTH_LONG).show();
                } else {

                    //ad ogni swipe la lista viene cancellata e rifatta
                    listMessages.clear();


                    auth.getAccessToken(new Authentication.JSONResponseCallback() {
                        @Override
                        public void onJSONResponse(boolean success, String access_token, String
refresh_token) {

                            if (success) {
                                editor.putString("access_token", access_token);
                                editor.putString("refresh_token", refresh_token);
                                editor.apply();
                                Log.e(TAG, "token token : " + access_token);
                            } else {
                                Toast.makeText(getApplicationContext(), access_token,
Toast.LENGTH_LONG).show();

                            }

                        }

                    }, settings.getString("indirizzo-atg", "indirizzo"),
FirebaseInstanceId.getInstance().getToken(), settings.getString("access_token", "access"),
settings.getString("refresh_token", "refresh"));


                    NotificationAll getEvents = new NotificationAll(MainActivityMod.this);
                    getEvents.getAll(settings.getString("indirizzo-atg", "indirizzo"),
settings.getString("access_token", "token"), listMessages, adapter, layoutText, listView,
getIntent());


                }
                    //con questo comando faccio in modo che quando lo swiperefreshlayout ha gestito
quello che
                    //deve fare smetta di caricare e visualizzi i risultati
                    mySwipeRefreshLayout.setRefreshing(false);



            }
        }
);
```

Figura 13: metodo setOnRefreshLister relativo a mySwipeRefreshLayout

Cliccando su un elemento della lista il metodo della classe AdapterView setOnItemClickListener() tramite Intent aprirà l'Activity ActivityFragment.

```
//setto la gestione del tocco sul singolo elemento della lista
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapter, View view, int position, long id) {

        ConnectivityManager connectivityManager = (ConnectivityManager)
getApplicationContext().getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();

        Log.e(TAG, "conn : " + activeNetworkInfo);


        //viene fatto un controllo sulla connessione internet: se non c'è si avvisa con un toast se
no
        //si va avanti
        if (activeNetworkInfo == null) {
            Toast.makeText(getApplicationContext(), "Nessuna connessione di rete",
Toast.LENGTH_LONG).show();
        } else {

            //mi tiro fuori il parametro id_notifica che mi servirà per fare la get per tirarmi
fuori i
            //dettagli del corrispondente evento
            //la posizione degli elementi della lista me la da listmessages.get(position) perchè
listmessages è un
            //array che tiene in memoria tutti i dati delle notifiche che sono arrivate perchè gli
oggetti message
            //vengono man mano popolati con i dati delle notifiche in arrivo
            final Message rigaLista = (Message) listMessages.get(position);
            int id_notifica = rigaLista.getId();

            //lancio l'activity che conterrà i fragment..... al posto di scrivere this come in
un'activity normale
            //bisogna mettere view.getContext()
            Intent templateIntent = new Intent(view.getContext(), ActivityFragment.class);
            templateIntent.putExtra("id", id_notifica);
            Log.e(TAG, "id : " + id_notifica);
            startActivity(templateIntent);
            //recreate ricrea l'activity, non è male solo che perdo tutte le notifiche precedenti
            finish();

        }


    }
});
```

Figura 14: parte di codice relativa alla gestione della pressione su un elemento della lista

Figura 15: Schermata esemplicativa di MainActivityMod

- ActivityFragment: questa Activity è fondamentale perché funge da contenitore per il Fragment TemplateFragment. Un Fragment è un componente indipendente che può essere collegato ad una Activity dell'applicazione in Android. Tipicamente definisce una parte dell'interfaccia utente (oppure anche tutta) riferita a quell'Activity e incapsula al suo interno le funzionalità in modo da rendere più facile e veloce il riutilizzo all'interno di Activities e layouts. Tale tipo di componente

viene eseguito sempre nel contesto di una Activity ma rimane indipendente per quanto riguarda il ciclo di vita e l'interfaccia utente; infatti i Fragments hanno un proprio ciclo di vita e una propria interfaccia. ActivityFragment andrà a contenere sempre e solo TemplateFragment, che si va ad aggiornare in maniera dinamica alla pressione dei vari bottoni che contiene. È importante notare che questa Activity è in costante comunicazione col Fragment che ospita in quanto i dati ricevuti che devono essere passati tra le varie "sovrascritture" di TemplateFragment devono essere prima trasferiti tramite l'interface DataPassListener all'Activity;

```
//Activity che contiene il fragment che visualizzerà i vari template

public class ActivityFragment extends AppCompatActivity implements TemplateFragment.DataPassListener
{


    //dichiaro la stringa che farà da titolo al file Shared Preferences
    public static final String PREFS_NAME = "Preferenze";


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

        //istanzio un oggetto actionbar che mi permette di modificarla a piacimento
        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayOptions(ActionBar.DISPLAY_SHOW_CUSTOM);
        View viewActionBar = getLayoutInflater().inflate(R.layout.action_bar_layout, null);

        actionBar.setHomeButtonEnabled(true);

        ActionBar.LayoutParams actionBarParams = new ActionBar.LayoutParams(//Center the textview in
the ActionBar !
                ActionBar.LayoutParams.WRAP_CONTENT,
                ActionBar.LayoutParams.MATCH_PARENT,
                Gravity.CENTER);
        final TextView titoloActivity = (TextView) viewActionBar.findViewById(R.id.titoloActivity);
        titoloActivity.setText("Gestione Evento");
        actionBar.setCustomView(viewActionBar, actionBarParams);

        //istanzio un nuovo oggetto template fragment faccio in modo che venga visualizzato
       TemplateFragment templateFragment = new TemplateFragment ();
        templateFragment.setArguments(getIntent().getExtras());
        getSupportFragmentManager().beginTransaction()
                .replace(R.id.container, templateFragment )
                .commit();
    }
    //questa interfaccia è direttamente collegata alla classe TemplateFragment da cui prende il
valore
    //numeropagina; questo valore serve perchè quando verrà fatto il reload del fragment per
visualizzare una
    //nuova pagina del template, l'id pagina verrà appunto preso da numeropagina
    @Override
    public void passData(int numeroPagina) {
        TemplateFragment templateFragment = new TemplateFragment ();
        Bundle args = new Bundle();
        args.putInt("idPagina", numeroPagina);
        args.putInt("id", getIntent().getIntExtra("id", 0));
        templateFragment .setArguments(args);
        getSupportFragmentManager().beginTransaction()
                .replace(R.id.container, templateFragment )
                .commit();

    }
    //metodo con cui decido cosa voglio che venga fatto quando premo il tasto back
    @Override
    public void onBackPressed() {

        super.onBackPressed();


        //recupera l'indirizzo impostato dall'utente in loginsettings
        SharedPreferences settings = this.getSharedPreferences(PREFS_NAME, 0);

        //istanzio la classe NotificationResponse per poter poi utilizzare il suo metodo chiudi
        //che mi permette di chiudere il template aperto (fa partire una richiesta /cancel verso
atg)
        NotificationResponse response = new NotificationResponse(this);

        response.chiudi(settings.getString("indirizzo-atg", "indirizzo"),
getIntent().getIntExtra("id", 0), 0, settings.getString("access_token", "token"), null);

        Intent main = new Intent(getApplicationContext(), MainActivityMod.class);
        startActivity(main);
        finish();
    }
}
```

Figura 16: codice relativo ad ActivityFragment

- TemplateFragment: va a rappresentare le schermate che mostrano il contenuto dei singoli eventi. È stato scelto di utilizzare un Fragment in quanto serviva un elemento che potesse facilmente aggiornarsi e cambiare aspetto, in quanto ogni evento viene gestito mostrando dei template dinamici con più pagine che cambiano elementi a seconda della tipologia di evento. Con questa soluzione si ha una sola Activity che contiene un elemento che si autoaggiorna a seconda delle scelte dell'utente, scelta rivolta all'ottimizzazione delle risorse, rispetto a creare un Activity per ogni tipo di evento dovendo gestire un numero elevato di cicli di vita. Elementi fondamentali di questo Fragment sono:
    - L'interface DataPassListener per passare i dati (in questo caso le scelte operate dall'utente e il percorso tra le pagine dei templates) tra Fragment, Activity e di nuovo Fragment;
    - L'oggetto della classe Authentication istanziato per utilizzare il metodo getAccessToken();
    - Un'istanza della classe NotificationLock che chiama il metodo lock(), utilizzato per fare in modo che se un utente apre un evento, questo non sia più disponibile nelle liste di altri devices, nel caso in cui la notifica sia stata inviata a più utenti;
    - Un oggetto della classe NotificationResponse, istanziato per usare i metodi getPagina() (per mandare una PUT request contenente le scelte digitate dall'utente nelle varie pagine dei template; verrà ritornato un intero corrispondente al numero di pagina successivo del template al quale si deve accedere), chiudi() (per mandare una PUT request per chiudere la

gestione dell'evento; in questo caso eventuali selezioni nelle pagine del template verranno cancellate) e conferma() (per mandare una PUT request che conferma la gestione dell'evento; chiudendo l'evento con questo metodo non sarà più visualizzabile nella lista degli eventi);

- Abbiamo poi una parte di codice adibita a creare dinamicamente la pagina in base all'oggetto JSON ritornato da getPagina(); il sistema funziona infatti in questo modo: vengono inviate delle notifiche che rappresentano degli eventi (es. somministrazione terapia al letto di un certo paziente), questi eventi quando vengono aperti dalla lista iniziano ad essere gestiti, e verranno quindi visualizzate delle pagine con una serie di elementi in grado di portare l'utente a completare la gestione dell'evento (es. domande a scelta multipla, a scelta esclusiva, parti in cui si possono aggiungere delle note); la struttura della pagina è intrinsecamente sempre la stessa (titolo evento, possibilità di aprire un sito web, domanda con possibili risposte, possibilità di inserimento note e bottoni per muoverci tra le pagine, chiudere o confermare la gestione dell'evento), ma verranno modificati di volta in volta i componenti interattivi che la compongono, scegliendo tra: TextView, EditText, RadioButton, Checkbox, Button;
- Tramite Intent in certi tipi di template si può accedere ad una particolare Activity, chiamata BrowserActivity, che svolge la funzione di browser interno all'applicazione: è infatti possibile visitare particolari link (ad esempio che riportano al tracciato ECG del paziente) senza dover uscire dall'app ed aprire il browser nativo del device.

```
case "checkbox":

    //se optionType è checkbox la procedura è uguale a prima solo che verrà
    //visualizzato il layout contenente i checkbox

    titolo.setText(testo);

    //verifico se il template richiede che venga visualizzato l'edittext per inserire commenti
    if (pagina.has("options") && pagina.has("useComment") &&
pagina.get("useComment").toString().equals("true")) {

        //array che contiene le opzioni
        final JSONArray opzioni = pagina.getJSONArray("options");
        final ArrayList<String> opt = new ArrayList<String>(opzioni.length());
        for (int i = 0; i < opzioni.length(); i++) {
            opt.add(opzioni.getJSONObject(i).get("caption").toString());
        }

        final ArrayList<String> id = new ArrayList<String>(opzioni.length());
        for (int i = 0; i < opzioni.length(); i++) {
            id.add(opzioni.getJSONObject(i).get("id").toString());
        }

        radioGroup.setVisibility(View.GONE);
        layoutCheckBox.setVisibility(View.VISIBLE);
        layoutInformazioni.setVisibility(View.GONE);
        layoutInformazioni.setVisibility(View.VISIBLE);
        layoutEdit.setVisibility(View.VISIBLE);
        layoutBottoni.setVisibility(View.GONE);
        layoutBottoni.setVisibility(View.VISIBLE);

        final EditText editText;
        editText = new EditText(getContext());


        editText.setHintTextColor(getResources().getColor(R.color.gray));
        editText.setTextColor(getResources().getColor(R.color.black));
        Display display = ((WindowManager)
getContext().getSystemService(Context.WINDOW_SERVICE)).getDefaultDisplay();
        int width = display.getWidth();
        LinearLayout.LayoutParams layoutEditParams = new LinearLayout.LayoutParams(width,
ViewGroup.LayoutParams.WRAP_CONTENT);
        editText.setLayoutParams(layoutEditParams);
        layoutEdit.addView(editText);


        final CheckBox[] chArray = new CheckBox[opt.size()];
        final ArrayList<Button> bList1 = new ArrayList<Button>(etichetta.size());

        if (web.has("label")) {
            Button informazioni1;
            informazioni1 = new Button(getContext());
            final String label = web.get("label").toString();
            informazioni1.setText(label);
            informazioni1.setTextColor(getResources().getColor(R.color.white));
            informazioni1.setBackground(getResources().getDrawable(R.drawable.buttonshape));
            LinearLayout.LayoutParams layoutInfoParams = new
LinearLayout.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.LayoutParams.WRAP_CONTENT);
            layoutInfoParams.setMargins(5, 3, 0, 0); // left, top, right, bottom
            informazioni1.setLayoutParams(layoutInfoParams);
            layoutInformazioni.addView(informazioni1);


            //setto l'ascoltatore sul bottone informazioni e gestisco il suo tocco
            informazioni1.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {

                    //se l'etichetta equivale a Chiama l'app apre il tastierino per
                    //eseguire una chiamata col numero inserito in indirizzo, se no fa
                    //partire l'activity per visualizzare pagine internet
                    if (label.equals("Chiama")) {
                        Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse(indirizzo));
                        startActivity(intent);
                    } else {

                        //viene lanciata l'activity che contiene la webview
                        Intent intent = new Intent(getContext(), BrowserActivity.class);
```

Figura 17: parte di codice di TemplateFragment che mostra come viene composto il template nel caso siano presenti dei checkboxes

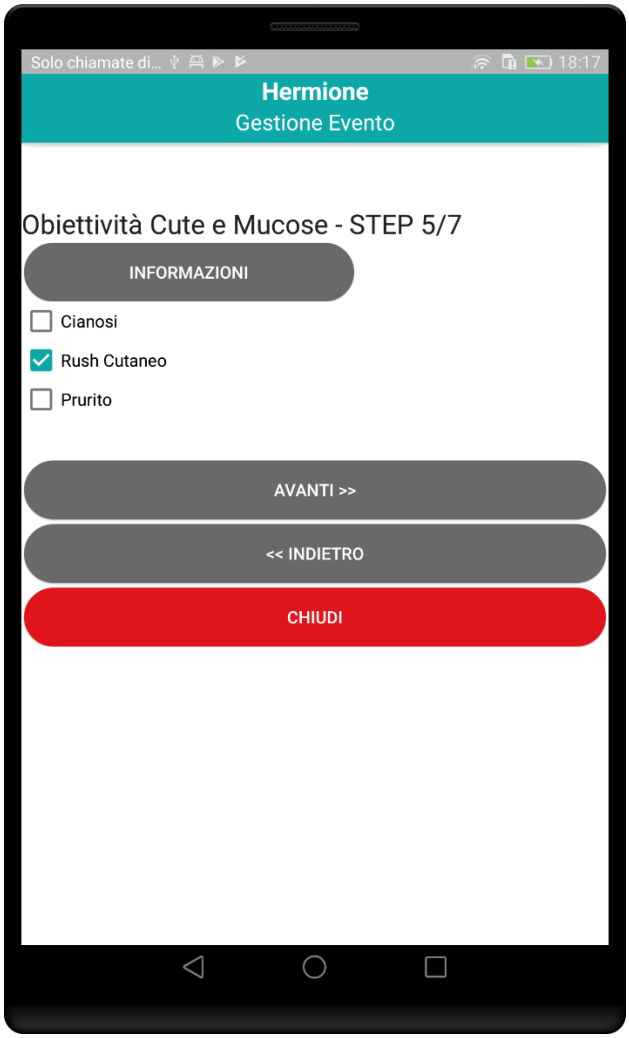


Figura 18: esempio template contenente checkboxes

2.7.3 File Manifest

Il file manifest è un importante file XML che contiene informazioni dettagliate riguardo all'applicazione, più precisamente definisce:

- Il nome del package;
- Il codice della versione;
- Il nome della versione;

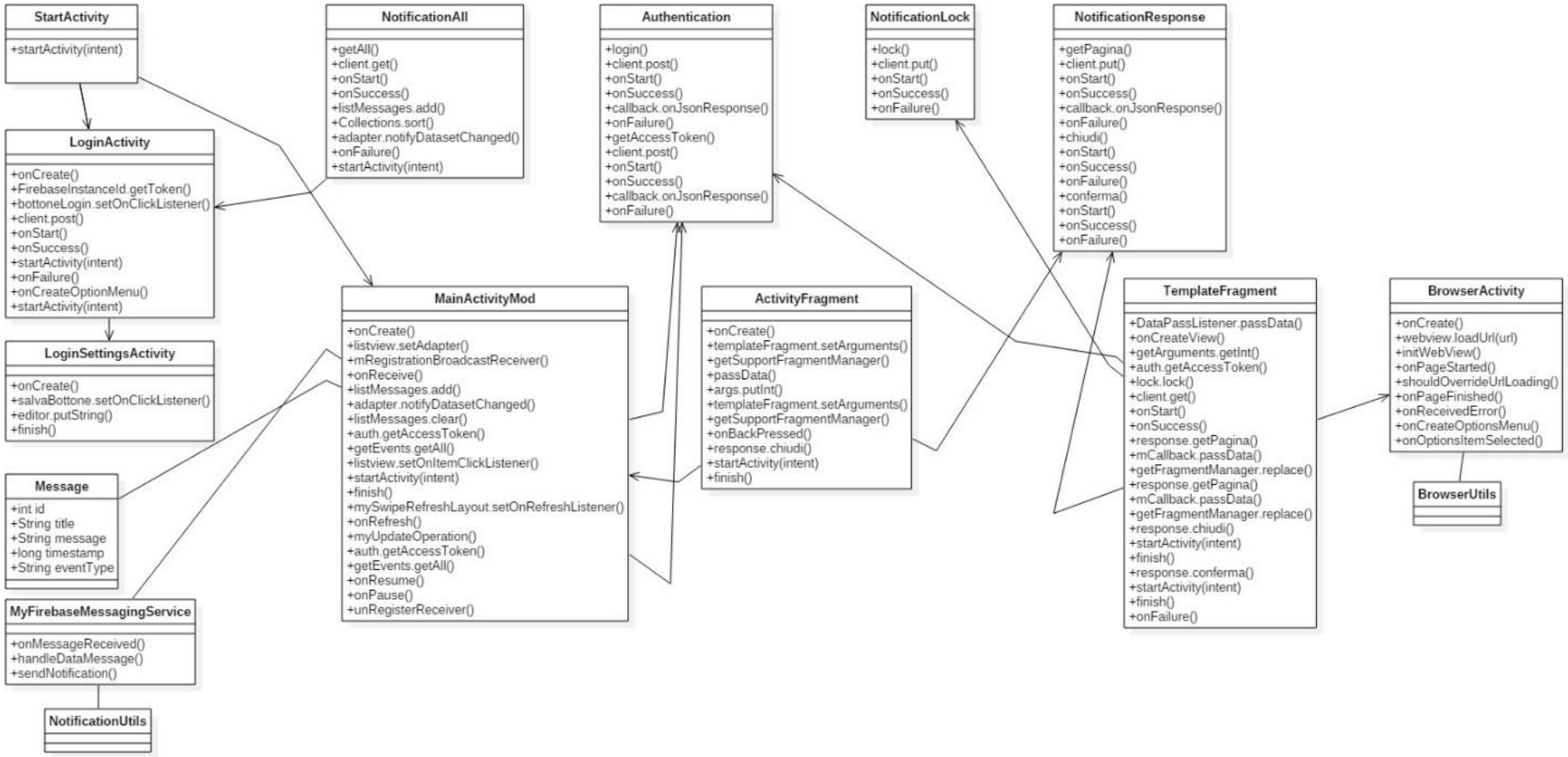
- La versione del SDK minima per l'utilizzo dell'applicazione (minSdkVersion);
- L'immagine che rappresenta l'applicazione sul dispositivo (icon);
- Il nome dell'applicazione (label);
- Il nome delle activity presenti nell'applicazione.

È un file fondamentale perché va a definire soprattutto i permessi di cui l'applicazione ha bisogno per funzionare. Nel caso di Hermione è servito ottenere il permesso ad utilizzare la connessione internet, a leggere lo stato del device e lo stato della connessione di rete e ad accedere al comparto telefonico del device

3 Risultati

3.1 Class Diagram completo di Hermione

A seguire viene proposto un particolare tipo di class diagram che racchiude in se tutte le classi Java dell'applicazione nella sua versione finale coi loro metodi più importanti; le relazioni/associazioni presenti tra le classi non sono quelle di un normale class diagram bensì si è voluto evidenziare quando una classe deve ricorrere ad un particolare metodo definito in un'altra. Nel diagramma si dovrebbe vedere come tra i metodi che una classe chiama ci sia una struttura gerarchica: ad esempio quasi in ogni classe il primo metodo chiamato è onCreate() che al suo interno va a chiamare altri metodi o metodi di altre classi istanziate che a loro volta vanno a chiamarne altri, come da paradigma fondamentale della programmazione ad oggetti.


StartActivity
+startActivity(intent)
NotificationAll
+getAll()
+client.get()
+onStart()
+onSuccess()
+listMessages.add()
+Collections.sort()
+adapter.notifyDatasetChanged()
+onFailure()
+startActivity(intent)
Authentication
+login()
+client.post()
+onStart()
+onSuccess()
+callback.onJsonResponse()
+onFailure()
+getAccessToken()
+client.post()
+onStart()
+onSuccess()
+callback.onJsonResponse()
+onFailure()
NotificationLock
+lock()
+client.put()
+onStart()
+onSuccess()
+onFailure()
NotificationResponse
+getPagina()
+client.put()
+onStart()
+onSuccess()
+callback.onJsonResponse()
+onFailure()
+chiudi()
+onStart()
+onSuccess()
+onFailure()
+conferma()
+onStart()
+onSuccess()
+onFailure()
LoginActivity
+onCreate()
+FirebaseInstanceId.getToken()
+bottoneLogin.setOnClickListener()
+client.post()
+onStart()
+onSuccess()
+startActivity(intent)
+onFailure()
+onCreateOptionMenu()
+startActivity(intent)
LoginSettingsActivity
+onCreate()
+salvaBottone.setOnClickListener()
+editor.putString()
+finish()
MainActivityMod
+onCreate()
+listview.setAdapter()
+mRegistrationBroadcastReceiver()
+onReceive()
+listMessages.add()
+adapter.notifyDatasetChanged()
+listMessages.clear()
+auth.getAccessToken()
+getEvents.getAll()
+listview.setOnItemClickListener()
+startActivity(intent)
+finish()
+mySwipeRefreshLayout.setOnRefreshListener()
+onRefresh()
+myUpdateOperation()
+auth.getAccessToken()
+getEvents.getAll()
+onResume()
+onPause()
+unRegisterReceiver()
ActivityFragment
+onCreate()
+templateFragment.setArguments()
+getSupportFragmentManager()
+passData()
+args.putInt()
+templateFragment.setArguments()
+getSupportFragmentManager()
+onBackPressed()
+response.chiudi()
+startActivity(intent)
+finish()
TemplateFragment
+DataPassListener.passData()
+onCreateView()
+getArguments.getInt()
+auth.getAccessToken()
+lock.lock()
+client.get()
+onStart()
+onSuccess()
+response.getPagina()
+mCallback.passData()
+getFragmentManager.replace()
+response.getPagina()
+mCallback.passData()
+getFragmentManager.replace()
+response.chiudi()
+startActivity(intent)
+finish()
+response.conferma()
+startActivity(intent)
+finish()
+onFailure()
BrowserActivity
+onCreate()
+webview.loadUrl(url)
+initWebView()
+onPageStarted()
+shouldOverrideUrlLoading()
+onPageFinished()
+onReceivedError()
+onCreateOptionsMenu()
+onOptionsItemSelected()
BrowserUtils
Message
+int id
+String title
+String message
+long timestamp
+String eventType
MyFirebaseMessagingService
+onMessageReceived()
+handleDataMessage()
+sendNotification()
NotificationUtils

3.2 Integrazione di Hermione col Database centralizzato e il modulo ATG

L'applicazione può essere integrata con i diversi moduli che compongono MEDArchiver® (CLIO, MEDREV, MEDTERAPIE solo per citarne alcuni) tramite collegamento client-server con un Database centralizzato, che avviene mediante il modulo ATG, un set di servizi web REST che interagiscono con l'app e col Database. Le chiamate REST a cui ci si riferiva durante l'analisi della fase di sviluppo rappresentano proprio le chiamate di ATG (allo stesso modo quando prima si parlava di Provider si intendeva il modulo ATG). Si avranno quindi le seguenti modalità di interazione:

- Login: prima di ogni altra chiamata è necessario fare quella relativa al login. Questo metodo riceve in ingresso username, password, nome device e device token tramite POST request e restituisce un access token e un refresh token; l'access token serve per eseguire qualsiasi altra chiamata;
- Notification-all: questa chiamata viene fatta per ricevere da ATG la lista di tutte le notifiche/eventi per uno specifico device. Per inviare la richiesta bisognerà inserire nel campo "Header" l'access token ottenuto durante la procedura di login. Per richiedere invece una notifica in particolare bisogna aggiungere in coda all'indirizzo il suo id numerico. Nel campo event_type vengono ritornati dei valori che corrispondono a:
    - W: attenzione;
    - I: info;
    - A: allarme;

    - T: task;
    - E: valutazioni.
- Notification-response: chiamata che il device fa per inviare le selezioni dell'utente; viene ritornato il numero della pagina del template successiva;
- Notification-lock: con questa chiamata, che viene eseguita sempre prima della chiamata che ritorna una singola notifica/evento, è possibile fare in modo che l'evento non sia più visibile nelle liste di altri device/utenti (se la notifica è stata inviata a più device/utenti) finchè non viene chiuso;
- Notification-cancel: permette di chiudere la gestione di un evento, rendendolo visibile di nuovo a tutti e cancellando le eventuali selezioni fatte dall'utente nelle varie pagine del template che visualizza l'evento.

3.3 Gestione notifiche ed eventi

Hermione è in grado di ricevere e visualizzare notifiche legate ad eventi clinici di ogni genere: reminder per somministrazioni terapia, avvisi di assistenza, warning per malfunzionamenti di dispositivi medici collegati ai vari moduli di MEDArchiver® e tanti altri. L'utente oltre a visualizzare gli eventi è in grado di gestirli; è infatti possibile interagire col modulo che ha inviato la notifica:

- Rispondendo a domande sotto forma di questionario con risposte a scelta multipla oppure mutuamente esclusiva;
- Inserendo note testuali per documentare la risoluzione di un problema oppure la gestione dell'evento;

- Accedendo alla parte telefonica del device per chiamare con la massima rapidità un numero utile suggerito dall'applicazione;
- Visualizzando referti del paziente come ad esempio quello dell'elettrocardiogramma oppure i suoi esami del sangue.

Dal punto di vista architetturale ad ogni notifica corrisponde un evento, e ogni evento viene visualizzato tramite un template dinamico, le cui pagine hanno sempre la stessa struttura di base, ma cambiano a livello di elementi visualizzati in base alle risposte date dall'utente o alla tipologia dell'evento. Gli elementi che devono andare a comporre il template vengono inviati all'interno di un oggetto JSON all'applicazione come response della GET request inviata per recuperare la notifica corrispondente all'evento. Come visto nel capitolo due, nel codice dell'applicazione (in particolare nella classe TemplateFragment) è presente una serie di metodi e componenti che associano agli elementi testuali dell'oggetto JSON gli elementi grafici e applicativi di un'applicazione Android.

Di seguito viene mostrato l'oggetto JSON contenente la struttura base di una pagina di un template:

```
{
  "pageId" : int,

  "text" : "String",

  "optionType" : "radio" || "checkbox" || "none",

  (se l'opzione inserita è none il jsonarray "options" non viene messo)

  "options" : [ { "id" : "int", "caption" : "String" },
                { "id" : "int", "caption" : "String" },
                …
              ],

  (opzionale se si vuole inserire una nota) "useComment": true,

  (opzionale se si vuole inserire una nota) "requireComment": true,

  (opzionale se si vuole inserire una nota) "commentLabel": "String",

  "buttons": [ { "label": "String", "action": "cancel" || "confirm" ||
                 "next" || "back" },
               …
             ],

  "webview": { "label" : "String", "address" : "url" },
}
```

I campi più interessanti sono:

- optionType: con questo campo si esplicita che tipo di risposta si vuole: radio se si può scegliere solo un responso tra quelli visualizzati, checkbox se è possibile selezionare più opzioni e none nel caso in cui non ci sia da rispondere o scegliere, bensì si debba scrivere qualcosa (valutazione medica, nota);
- options: è un array che contiene gli id e le possibili opzioni che servono per andare a generare il template (se optionType è none questo campo non è presente);
- buttons: è un array che contiene gli id e le etichette che servono per andare a generare i bottoni del template.

L'applicazione sviluppata elabora i dati dell'oggetto JSON ricevuto andando a creare dinamicamente i vari template.

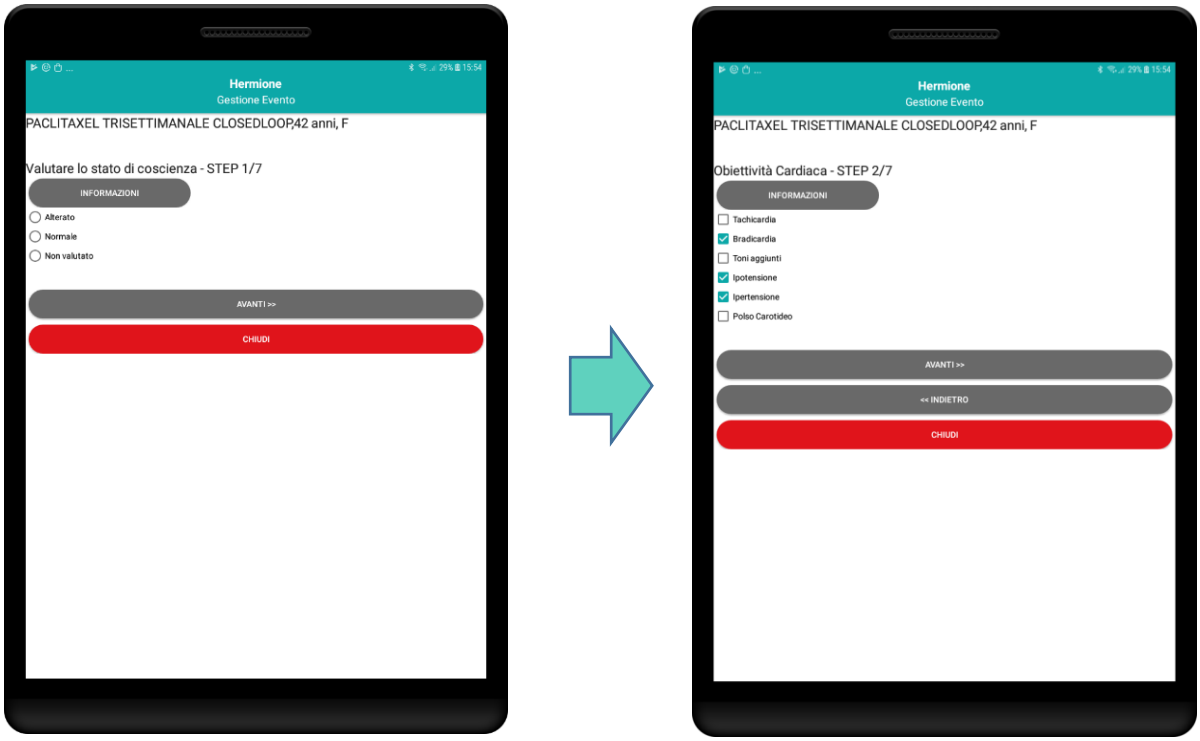


Figura 19: costruzione dinamica del template svolta dall'app partendo dall'oggetto JSON

3.4 Sicurezza e accesso

La procedura di login al sistema segue lo standard oAuth 2.0, che rappresenta una soluzione sicura perché garantisce l'accesso da parte di terzi ai dati proteggendo contemporaneamente le loro credenziali. L'idea di base è quella di autorizzare terze parti a gestire documenti privati senza condividere la password (operazione che presenta molti limiti di sicurezza). OAuth nasce quindi con il presupposto di garantire l'accesso delegato ad un client specifico per determinate risorse sul server per un tempo limitato, con possibilità di revoca. Gli attori principali sono:

- Server: contenitore delle risorse (Database + ATG);
- Client: parte che richiede l'accesso alle risorse protette (Hermione);
- Utente: persona fisica che vuole accedere alle risorse presenti sul server.

Per accedere alle risorse, l'utente inserisce le sue credenziali, il client fa una richiesta inserendo le credenziali utente e interagisce col server ottenendo delle proprie credenziali temporanee (access token). Quando le credenziali temporanee scadono, dovrà essere inviata una richiesta per ottenere un nuovo access token che l'app utilizza per accedere al sistema (se si tenta di accedere con un token scaduto verrà restituito il messaggio "Expired token"). La procedura di reautorizzazione esula comunque da un nuovo inserimento delle credenziali, in quanto quando viene rilasciato un access token, questo viene accompagnato da un refresh token da utilizzare per richiedere un nuovo access token. La scadenza del token è fissata a dieci minuti di default ma è modificabile.

Questo metodo è utilizzato da grandi aziende come Facebook, Twitter, Apple e Amazon. I suoi principali vantaggi sono i seguenti:

- Evita di controllare l'autorizzazione di ogni dispositivo in quanto il server rilascia un qualcosa (access token) per certificare l'autorizzazione all'accesso alle risorse;
- Disconnessione remota dall'utente dopo un certo intervallo di tempo rifiutando l'access token scaduto;
- L'utente può accedere solamente utilizzando le credenziali corrette;
- Lega i token a un determinato device;
- Evita la necessità di salvare localmente le credenziali utente;
- È di facile implementazione utilizzando librerie pronte all'uso.

Ogni singola chiamata ad ATG deve essere autorizzata inserendo nell'intestazione l'access token ricevuto.

3.5 Caso d'uso: funzionamento di Hermione e integrazione con i moduli OMM e CLIO

3.5.1 CLIO: un sistema ad anello chiuso

Il progetto CLIO nasce con lo scopo di utilizzare algoritmi basati sul machine learning per affrontare la sfida dell'analisi dei big data nel settore dell'assistenza sanitaria. Sfruttare questa enorme quantità di dati non è fattibile con i metodi convenzionali. Pensiamo ad un sistema informatico sofisticato, in grado di registrare continuamente i dati di infusione dei farmaci chemioterapici, i parametri vitali del paziente, le osservazioni e le valutazioni cliniche di medici e infermieri. CLIO è un sistema closed loop, nel quale il paziente viene associato al sistema infusionale e al monitor per la rilevazione dei parametri e questi a MEDArchiver® in modo da ricevere costantemente informazioni sullo stato del paziente e sull'andamento della terapia. Il sistema CLIO si basa su un motore a stati finiti, il quale valuta tutti gli input e genera una risposta di output che potrebbe essere un segnale d'allarme, un avviso o un'indicazione di come procedere con la terapia.

Un esempio di applicazione del sistema è in quei protocolli in cui le dinamiche dell'infusione mutano nel corso del trattamento, ovvero protocolli in cui il farmaco comincia l'infusione ad una certa velocità per poi essere incrementata successivamente, se le condizioni del paziente lo consentono. All'interno del sistema questa tipologia di protocolli viene adeguatamente configurata, vengono definiti i limiti di guardia per i parametri del paziente e una volta avviata l'infusione sarà il sistema ad avvisare l'operatore di come proseguire seguendo le linee guida del protocollo configurato.

CLIO è anche somministrazione sicura del farmaco in quanto integra tutti i controlli per il patient match (processo di comparazione dei dati da differenti sistemi informativi sanitari per vedere se appartengono tutti allo stesso paziente ottenendo così un report completo della storia clinica del paziente con relative indicazioni terapeutiche).

Oltre a questo CLIO è il controllore dello stato del paziente durante la terapia, con generazione di allarmi qualora le condizioni del paziente si avvicinino o superino i livelli di guardia impostati e infine è raccolta di dati clinici, in quanto i parametri rilevati vengono costantemente archiviati a sistema e possono essere utilizzati in qualsiasi momento.

Poiché la quantità di dati disponibile nei luoghi di cura del paziente sta crescendo grazie a apparecchiature portatili di monitoraggio, i sistemi di telemedicina, gli operatori necessitano di strumenti di supporto per interpretare facilmente tali dati e adottare le opportune azioni. Per esempio, l'infusione endovenosa di farmaci antineoplastici espone i pazienti critici alle reazioni avverse, ed esistono pochi strumenti che aiutano gli operatori a prevenire efficacemente tali situazioni. L'avvento degli anticorpi monoclonali ha recentemente rivoluzionato il trattamento dei tumori che si traduce in un minore numero di reazioni avverse rispetto alla chemioterapia convenzionale e in una maggiore tollerabilità dei pazienti affetti. Tuttavia, rimane una vasta gamma di eventi avversi, richiedendo un ulteriore sforzo da parte degli operatori per identificare, gestire e, se possibile, prevenire le reazioni.

Ed è qui che entra in gioco il Multi Device Closed Loop System: un sistema di sicurezza del paziente che mira a prevenire e minimizzare l'impatto delle reazioni avverse ai farmaci (ADR Adverse Drug Reaction) durante l'infusione endovenosa di farmaci antineoplastici, affrontando:

- L'insorgenza rapida delle reazioni avverse;
- La complessità del trattamento dei pazienti oncologici;
- L'impatto del monitoraggio delle condizioni del paziente;
- La gestione delle ADR in termini di risorse umane professionali e dei costi associati.

Il motore di CLIO:

1. Recepisce in ingresso i vari input:
    - Parametri Vitali e Dati di Infusione: provenienti dal sistema OMM che dialoga con la centrale di monitoraggio e le pompe di infusione;
    - Azioni utente: operazioni randomiche effettuate dagli operatori (stop infusione, cambi di velocità, riavvii, cambio farmaco ecc..)
    - Valutazioni mediche e infermieristiche: dettate dai protocolli standard.
2. Elabora i dati secondo i protocolli integrati e le logiche precedentemente impostate a sistema e in base alla terapia impostata per il paziente;
3. Per mezzo della app dedicata Hermione, fornisce gli output in tempo reale agli operatori con le istruzioni da eseguire e le operazioni corrette da intraprendere.

3.5.2 Use Case Diagram e Activity Diagram

Di seguito sono riportati i diagrammi UML che esemplificano alcune delle attività che possono essere realizzate con Hermione nell'ambito della sua integrazione con OMM e CLIO.

Il caso d'uso scelto si sviluppa nei successivi passi:

1. Il paziente si trova a letto in reparto (ad esempio in Oncologia);
2. Un sistema di monitoraggio viene associato al paziente: i suoi dati vengono passati a OMM (pressione, temperatura, saturazione, frequenza respiro);
3. Il paziente viene associato al sistema di infusione: i dati dalla pompa vengono passati a OMM (informazioni sul farmaco, volume infuso e velocità di infusione);
4. Tutti i dati acquisiti in precedenza vengono trasmessi a CLIO, che li elabora e suggerisce le azioni da intraprendere: START SOMMINISTRAZIONE;
5. L'operatore a questo punto compie l'azione AVVIO POMPA;
6. Dopo un intervallo di tempo preimpostato sulla base del corretto protocollo dovrà essere modificata la velocità di infusione del farmaco;
7. Viene inviato un alert su Hermione: MODIFICARE VELOCITA';
8. Viene rilevata una saturazione dell'ossigeno nel sangue (SpO2) del paziente fuori norma;
9. Un alert è generato su Hermione: STOP INFUSIONE + RICHIESTA VALUTAZIONE MEDICA;
10. L'operatore dopo aver controllato il paziente compilerà direttamente da Hermione la valutazione medica;

11. Viene inviato un alert su Hermione: STOP DEFINITIVO INFUSIONE;

12. L'operatore ferma definitivamente l'infusione del farmaco.

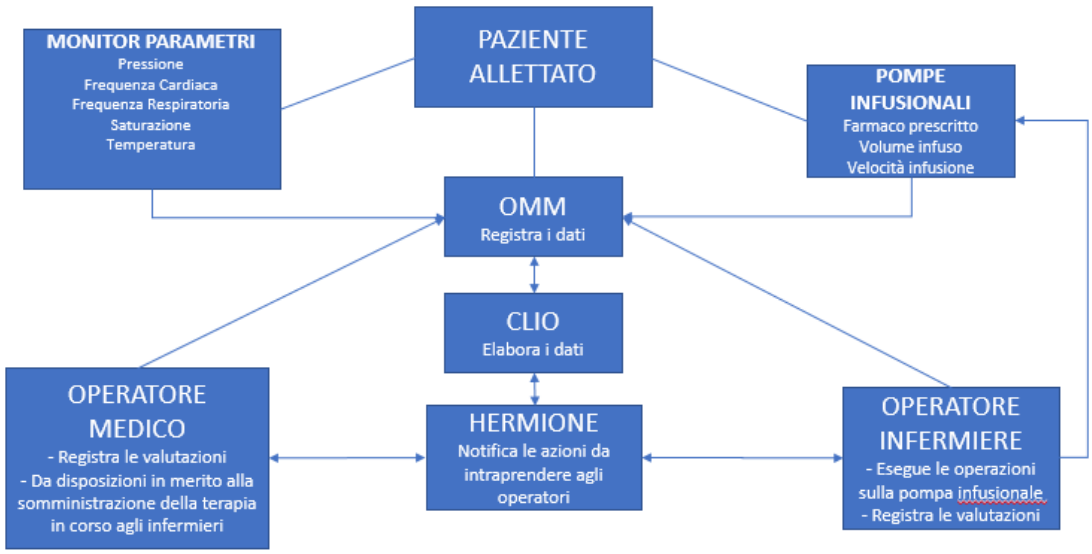


Figura 20: Use Case Diagram

In figura 20 si possono notare tutti gli attori coinvolti nel workflow e le interazioni tra di essi: al paziente allettato vengono collegati i due sistemi di monitoraggio e infusione che inviano i dati raccolti al modulo OMM; quest'ultimo registra i vari record e trasmette al modulo CLIO che li elabora andando successivamente ad inviare ai vari operatori suggerimenti sulle azioni da intraprendere in base all'analisi fatta sui dati ricevuti.

In figura 21 è rappresentato l'activity diagram del workflow che si scatena durante un episodio di somministrazione terapia in cui accade che un parametro paziente (saturazione dell'ossigeno nel sangue) vada fuori norma. Successivamente vengono

mostrate anche le varie schermate inerenti alle notifiche che compaiono sull'applicazione per la visualizzazione e gestione dell'evento.

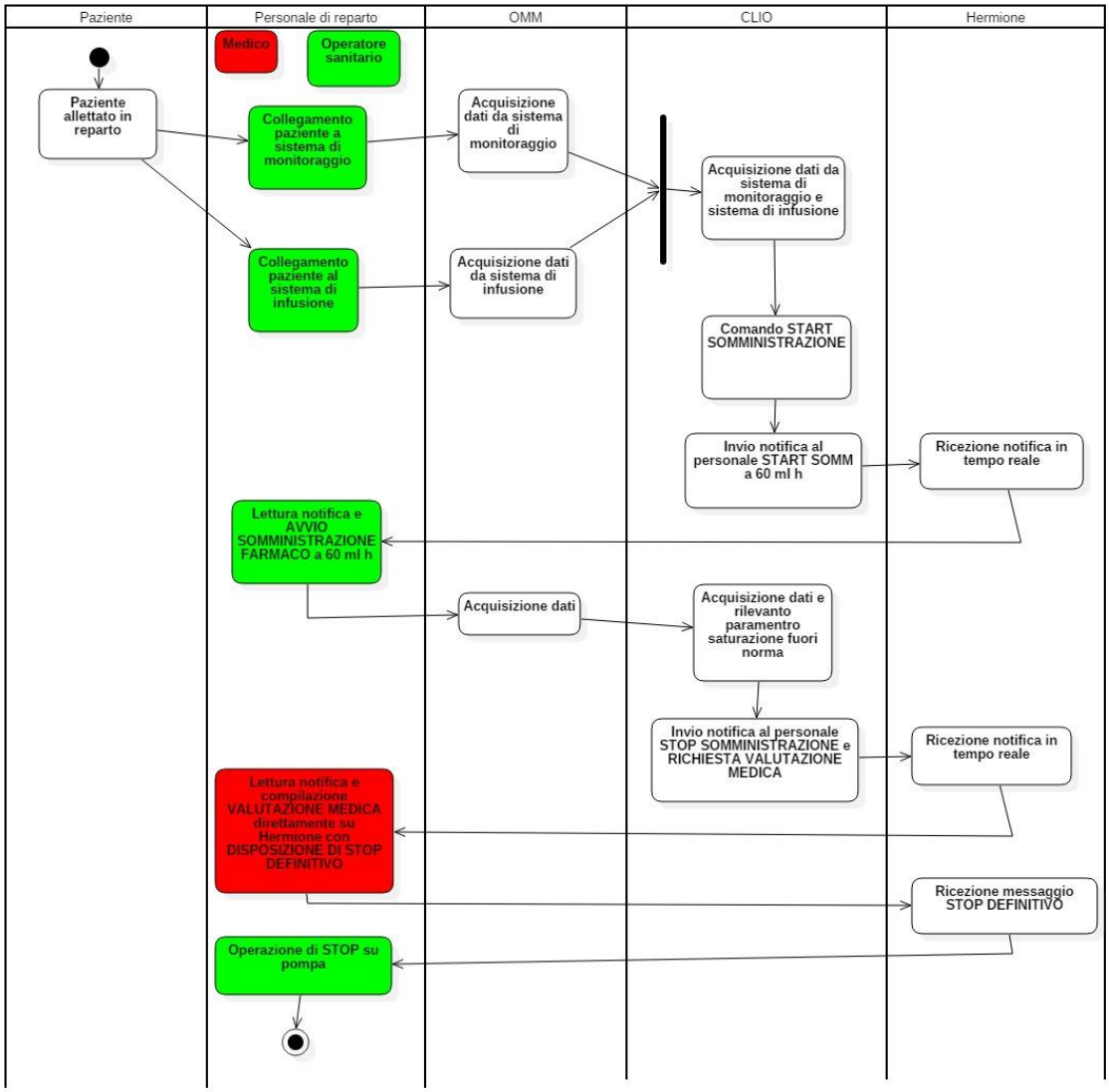


Figura 21: Activity Diagram inerente ad un esempio di workflow

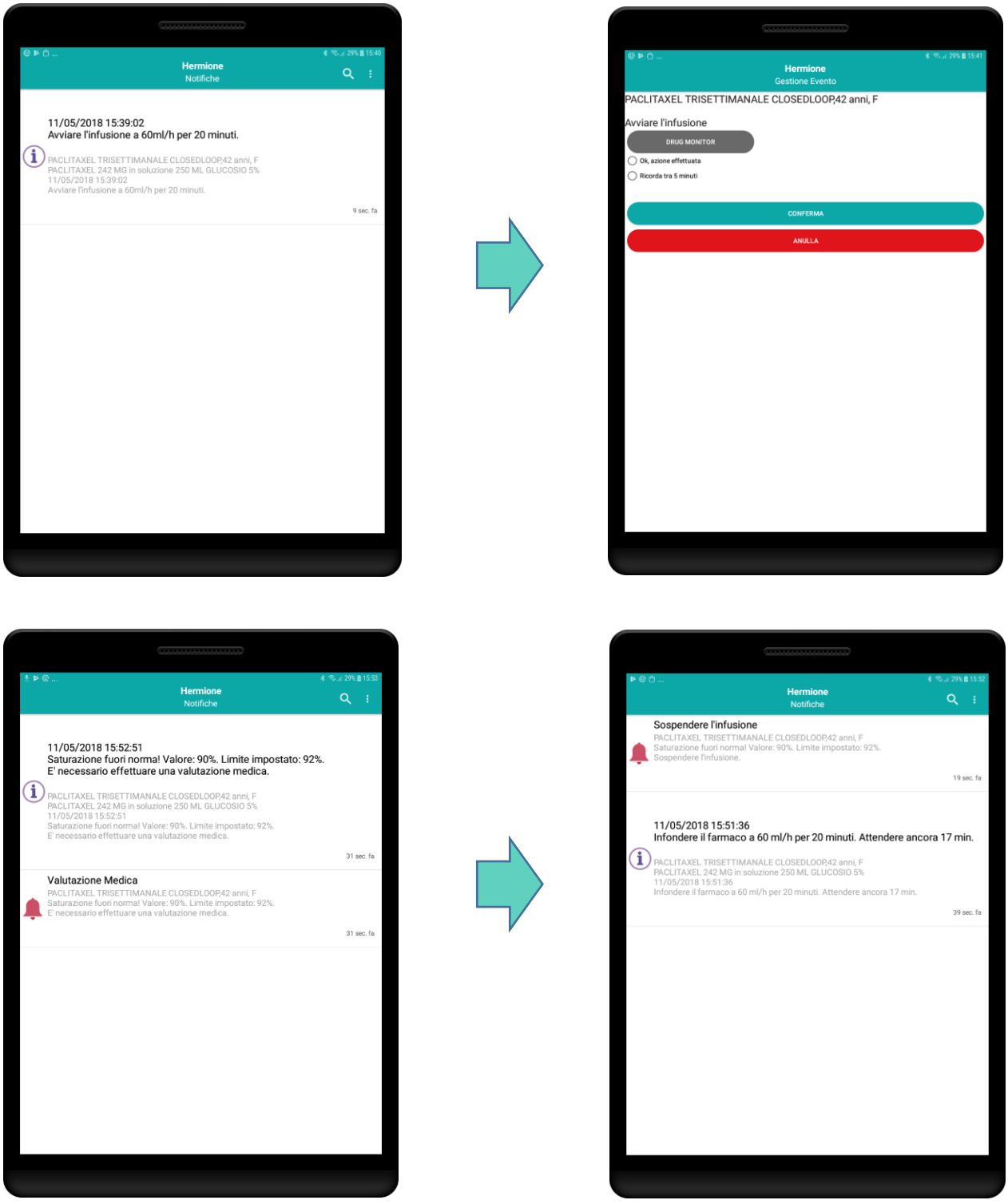


Figura 22: Schermate dell'applicazione che mostrano le varie notifiche e template inerenti al workflow preso in considerazione

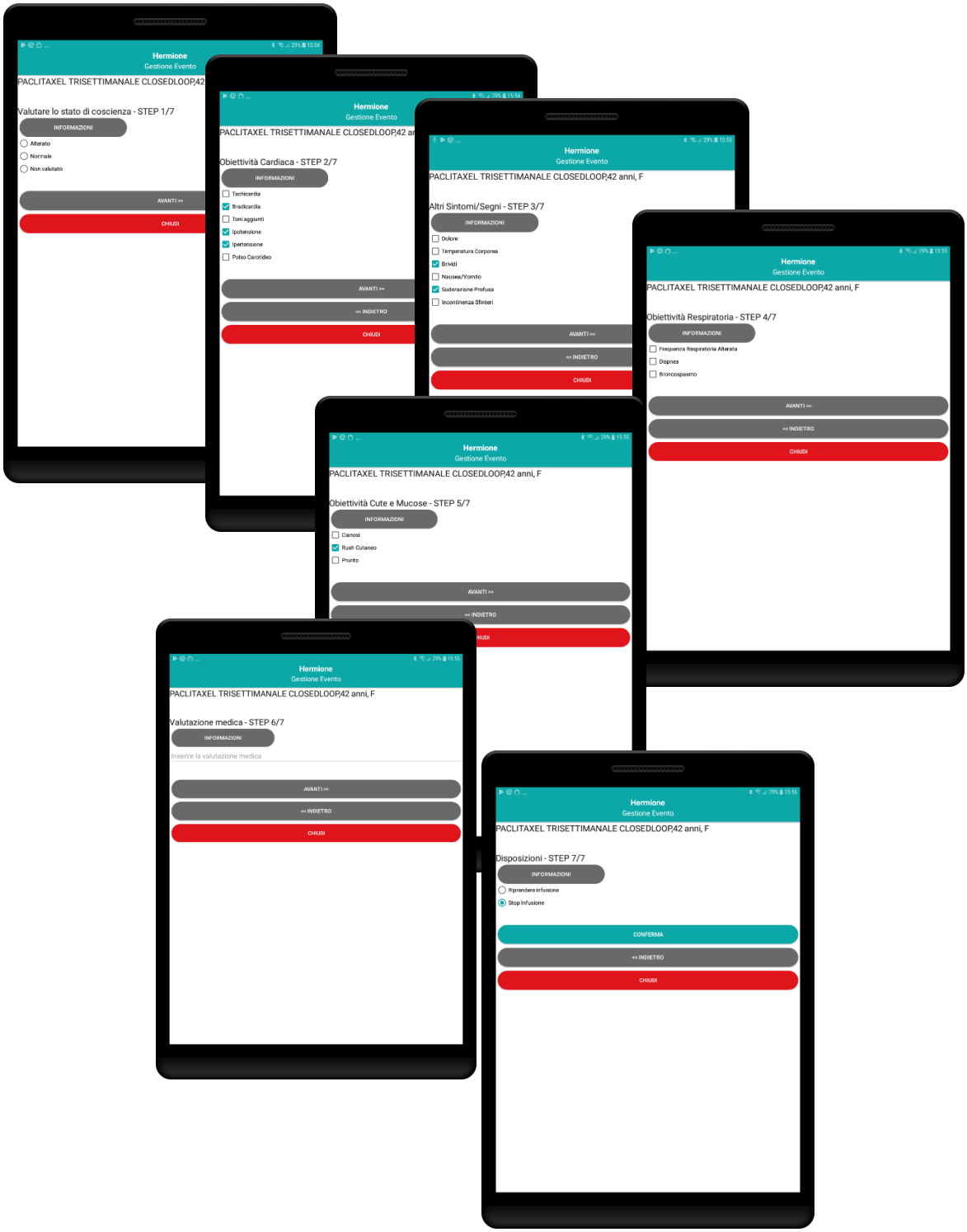


Figura 23: template che si riferisce alla gestione dell'evento "Valutazione medica"

4 Conclusioni

Lavorando ad un progetto come quello di Hermione, sono potuto venire a contatto con quello che con ogni probabilità rappresenterà uno dei temi principali nell'immediato futuro dell'applicazione della tecnologia in Sanità: la raccolta di una grande quantità di dati legati ai pazienti da qualsiasi dispositivo medico, in qualsiasi luogo, e come utilizzarli al meglio con le giuste tempistiche.

L'obiettivo che ci si pone col sistema OMM + CLIO + Hermione ad esempio è quello di avere sempre più la capacità di raccogliere dati di qualità, aprendo la strada al miglioramento continuo dei protocolli di cura. Questo per aiutare i clinici a:

- Prevedere se un trattamento farà più bene che male, e agire di conseguenza nella prescrizione;
- Ridurre errori durante la terapia;
- Ridurre gli eventi avversi;
- Ottimizzare costi e processi.

Il valore aggiunto di tutto questo è (Big Data Analitics) l'elaborazione delle informazioni che stimoli nuovi spunti di riflessione a partire dall'analisi di grandi quantità di dati, attribuendo alle evidenze emerse un peso decisivo nell'influenzare le scelte del personale medico/infermieristico.

Si vuole sviluppare un atteggiamento critico nei confronti del dato e fornire le competenze necessarie alla sua gestione.

Una medicina quindi basata sulle prove (Evidence Based Medicine) che cerca di valutare la forza delle evidenze dei rischi e benefici dei trattamenti. Questo aiuta i medici a prevedere se un trattamento farà più bene che male, e agire di conseguenza nella prescrizione.

È per questo che Hermione, oltre agli esempi visti, può svolgere tutta una serie di attività mirate a supportare il lavoro del personale medico/infermieristico, tutte ampiamente configurabili e personalizzabili.

Smartphone, Tablet, Smartwatch possono certamente diventare dispositivi medici, strumenti con delle potenzialità assistenziali enormi, tutto sta nello sfruttarli nel giusto modo guidati dalle giuste idee.

4.1 Sviluppi futuri

Uno dei possibili sviluppi già pensati per il futuro di Hermione, è quello di estendere l'applicazione ai pazienti, nell'ambito di un progetto di telemedicina. Essendo un assistente digitale mobile in grado di raggiungere e accompagnare ovunque l'utente, cosa meglio di Hermione può permettere di ricevere sul proprio device notifiche push (alert e reminder) relative ad eventi che coinvolgono personalmente il paziente?

Sono stati per il momento identificati tre possibili scenari di utilizzo realizzabili in una Struttura Sanitaria che mira a migliorare la qualità dei servizi al paziente:

1. Reminder prestazioni ambulatoriali: 24 ore prima della data prevista per la prestazione, viene inviata una notifica al dispositivo del paziente che potrà consultare le note di preparazione all'esame. Qualora il paziente non potesse presentarsi all'esame, può richiedere di essere contattato dalla struttura per fissare una data alternativa per l'appuntamento;
2. Ritiro referti on-line: quando il referto viene registrato in cartella clinica, viene inviata una notifica al dispositivo del paziente che può accedere direttamente al sito della struttura in cui ha effettuato la prestazione per visualizzare/scaricare il referto;
3. Monitoraggio pressione: ai pazienti critici in cura, viene inviata con cadenza specifica, una notifica al dispositivo personale per l'inserimento dei valori pressori che verranno poi visualizzati direttamente dal medico curante.

Bibliografia e Sitografia

[1] Ecosistema MEDArchiver®:

- https://www.medarchiver.com/;
- "Digital Clinic Era", MEDArchiver srl.

[2] Metodi:

- "Perché Android", http://www.html.it/pag/19496/introduzione-perch-android/;
- "iOS o Android? 10 elementi per chi deve scegliere", https://www.cwi.it/mobile-wireless/android/106597-106597;
- "Android programmazione avanzata", Fabio Collini;
- "Firebase Cloud Messaging, https://en.wikipedia.org/wiki/Firebase_Cloud_Messaging;
- https://developer.android.com/
- "Tutorial Fragment", www.corsomobile.it/utility/download.php?id=17;
- https://github.com/

[3] Risultati:

- "ATG Specifications: documento esplicativo sulle funzionalità di ATG e l'integrazione con HERMIONE", MEDArchiver srl;
- "MEDArchiver ATG API v2.3", MEDArchiver srl;
- https://oauth.net;
- https://it.wikipedia.org/wiki/OAuth;
- "CLIO ed Hermione Generazione 4.0", MEDArchiver srl;