ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

ESTENSIONE E POTENZIAMENTO DI UN SISTEMA PER IL TRACCIAMENTO IN AMBITO SANITARIO: IL PROGETTO TRACKING FOR CARE COME CASO DI STUDIO

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
CHIARA TARANTINO

Correlatore
Dott. Ing. ANGELO CROATTI
Dott. MARCO BENNI

Anno Accademico 2019 – 2020

Alla mia famiglia,
al mio fidanzato Alessandro,
e ai miei amici che mi hanno
sempre sostenuto in questo percorso.

Indice

Introduzione

L'enorme progresso tecnologico avuto negli ultimi anni ha portato grandi rivoluzioni nella vita quotidiana di miliardi di persone che hanno visto migliorare significativamente la loro qualità di vita e le condizioni nell'ambiente di lavoro. Un ruolo centrale in questa quarta rivoluzione industriale è ricoperto dall'*Internet of Things (IoT)*, è infatti la prima volta nella storia che vengono introdotti degli oggetti tecnologici in grado di svolgere compiti senza l'intervento dell'uomo con lo scopo di supportarlo.

Tra i vari ambiti dove l'applicazione di tali tecnologie potrebbe migliorare drasticamente la qualità del servizio erogato, ma purtroppo ancora oggi rimane un processo nella maggior parte dei casi ignorato o arretrato, c'è quello sanitario. Tale impiego introduce il concetto di *Internet of Healthcare Things (IoHT)* che indica un sistema sanitario nel quale i dispositivi IoT sono connessi tra loro e collaborano con il personale medico per prevenire malattie, fornire un servizio di assistenza ottimale per il paziente, ridurre i tempi di attesa, gli sprechi e soprattutto la probabilità di errore. Come conseguenza dell'IoHT nasce un nuovo modo di vedere l'ospedale che diventa una struttura smart dove tutto il possibile viene reingegnerizzato e automatizzato con l'intento di fornire dei servizi che si adeguano completamente alle esigenze dei pazienti e di facilitare anche il lavoro dei medici. Per il raggiungimento di questi obiettivi il monitoraggio, non solo dei pazienti, ma anche del personale e delle scorte di materiale medico, diventa una questione basilare, infatti le informazioni raccolte vengono utilizzate per il coordinamento tra gli operatori sanitari, come fondamento per le decisioni da compiere durante lo svolgimento delle loro attività ed anche a posteriori per valutare la prestazione del team e l'organizzazione ospedaliera svelandone eventuali criticità. Per questo motivo è importante che l'attività di tracciamento sia il più possibile completa, corretta e accurata ed è necessario che venga affidata maggiormente alle tecnologie IoHT in modo da ridurre l'onere cognitivo per il personale sanitario, permettendo loro di focalizzarsi pienamente sull'attività svolta.

È in questo ambito che si colloca il caso di studio esposto in questa tesi, il progetto *Tracking For Care (T4C)*, che si occupa di effettuare il tracciamento delle patologie tempo-dipendenti (trauma, ictus, patologie cardiache) sia in

fase ospedaliera che in quella preospedaliera. Questo elaborato si concentrerà in particolare su un sistema appartenente al T4C nato dalla collaborazione con il Trauma Team dell'Ospedale M.Bufalini di Cesena: *Trauma Tracker.* Esso è un sistema di tracciamento che permette di aiutare i medici a gestire specifiche situazioni di emergenza che coinvolgono pazienti traumatizzati, automatizzando il più possibile la raccolta in real-time delle informazioni rilevanti. Poiché Trauma Tracker attualmente gestisce solo la parte intraospedaliera, da un confronto con il *Trauma Team* è emersa la necessità di sviluppare un ulteriore sistema, denominato *PreH*, che si integrasse con il primo e che si occupasse del tracciamento del team e del paziente in tutta la fase preospedaliera, generalizzando però dalla sola gestione trauma verso tutte le altre patologie al fine di realizzare un monitoraggio pervasivo e completo. È stato dunque sviluppato un prototipo iniziale però ancora non del tutto adatto ad essere operativo.

Il lavoro di questa tesi consiste nel potenziare ed estendere il sistema PreH allo scopo di avvicinarsi sempre più a una versione che possa essere utilizzata sul campo. Dopo una breve introduzione sull'applicazione dell'IoT in ambito sanitario e in particolare nell'Ospedale 4.0, verrà descritto cosa significa fare tracciamento e la sua importanza portando Trauma Tracker come progetto esemplificativo. Quindi si entrerà nel merito di PreH, descrivendo dettagliatamente i vantaggi che apporta e gli obiettivi preposti, si effettuerà un'approfondita analisi del dominio e dei requisiti e dopo aver indicato come si è progettata l'architettura delle varie parti del sistema e in che modo esse comunicano tra loro, verrà descritto minuziosamente come l'applicativo è stato esteso e potenziato. Infine, l'ultima parte della tesi è dedicata alla validazione funzionale effettuata durante la fase di sviluppo e a quella svolta da un esperto clinico che ha permesso di valutare l'effettiva efficacia del lavoro svolto e in quale direzione proseguire.

Dato l'attuale restio all'introduzione delle tecnologie nell'ambito medico per i rischi connessi alla sicurezza e alla privacy, si cerca di dimostrare con questo prototipo la concreta validità della sua applicazione.

Capitolo 1

Applicazioni di tecnologie IoT in ambito Healthcare

Grazie alla *quarta rivoluzione industriale* si sono affermate nuove tecnologie, tra le quali **l'Internet of Things**, che hanno portato a nuovi approcci per la risoluzione dei problemi, impensabili fino a qualche anno fa.

In questo capitolo si introdurrà il contesto generale di riferimento per il progetto presentato con questa tesi di laurea partendo proprio da un'introduzione relativa all'Internet delle Cose. Quindi, si scenderà maggiormente nello specifico descrivendo la sua applicazione nell'ambito medico attraverso una panoramica generale **sull'Internet of Healthcare Things** che, dopo aver descritto come gli oggetti intelligenti si inseriscono nel mondo sanitario, si concentrerà principalmente sugli obiettivi, sui vantaggi che apporta e sulle sfide da affrontare. Verranno inoltre descritti anche alcuni esempi già operativi nella realtà attuale. Infine, verrà presentato **l'Ospedale 4.0**: l'ambito specifico nel quale il prototipo discusso in questo elaborato si colloca.

1.1 IoT e i principali ambiti di diffusione

Prima di poter parlare dell'Internet of Healthcare Things, si vuole introdurre il concetto alla sua base: **l'Internet of Things (IoT)**. Il termine IoT viene coniato per la prima volta da Kevin Ashton, ricercatore presso il MIT[1], nel 1999 che lo definisce come dei sistemi di oggetti fisici che sono in grado di raccogliere e scambiarsi dati utilizzando la rete Internet senza che l'uomo intervenga.

L'Internet of Things, in particolare nell'ultimo decennio, ha avuto applicazione nei più svariati ambiti. Se ne citano di seguito alcuni esempi rilevanti:

[1]**MIT:** Massachussets Institute of Technology.

- Settore delle industrie manifatturiere con **l'Industria 4.0 o Smart Factory**: è la definizione simbolica della quarta rivoluzione industriale. Consiste nella crescente integrazione di "sistemi cyber-fisici" (CPS), macchine intelligenti e connesse a Internet, nei lavori svolti dall'uomo all'interno delle aziende.

- Settore urbanistico con le **Smart City**: area urbana in cui, grazie all'utilizzo delle tecnologie digitali e più in generale dell'innovazione tecnologica, è possibile ottimizzare e migliorare le infrastrutture e i servizi per i cittadini rendendoli più efficienti.

- Settore domotico con le **Smart Home**: abitazioni nelle quali è possibile gestire in maniera automatica o da remoto impianti e dispositivi, al fine di risparmiare energia, semplificare la vita domestica e/o garantire la sicurezza delle persone all'interno.

- Settore medico con **l'Internet of Healthcare Things**.

1.2 Internet of Healthcare Things

1.2.1 Descrizione e obiettivi

Definizione L'Internet of Healthcare Things (IoHT in breve), detto anche Internet of Medical Things o Internet of Things in Healthcare, viene definito negli *Atti della IEEE International Conference Communications* [8] come un sistema sanitario basato sull'IoT che collega tutte le risorse disponibili in una rete, per svolgere attività sanitarie quali diagnosi, monitoraggio e interventi chirurgici a distanza su Internet. Può essere visto come un sottosistema delle Smart City.

Obiettivi L'obiettivo principale di usare la tecnologia IoT in ambito Healthcare è quello di *raccogliere dati dal corpo umano di un individuo permettendo così di monitorare la sua salute e di assisterlo digitalmente.* Il tracciamento dei parametri relativi al benessere di una persona, svolto in questa modalità "smart", permette sia di prevedere e quindi prevenire lo sviluppo di una malattia anticipando così le situazioni critiche molto prima che si verifichino ma anche di seguire l'evoluzione di una patologia già esistente per pazienti cronici, in modo da ridurre eventi acuti e ricoveri urgenti costosi per la collettività (basti infatti pensare che secondo lo studio svolto da *Lissemore* [7] circa l'80% della spesa sanitaria di ciascun Paese viene impiegata per le malattie croniche).

Un altro proposito che ha portato l'introduzione dell'IoHT nel sistema sanitario è stato quello di *snellire la richiesta sempre più crescente delle risorse*

mediche, sia in termini di personale ospedaliero, che di farmaci e strumenti, causata dal problema dell'invecchiamento della popolazione dovuto a sua volta dall'aumento dell'aspettativa di vita media. Le applicazioni IoHT, infatti, contribuiscono alla rivoluzione digitale nel settore sanitario e al miglioramento della qualità del servizio medico offerto.

Funzionamento

1. I sensori collegati ai dispositivi IoT, che solitamente sono oggetti wearable indossati dall'utente, raccolgono dati relativi alla salute sotto forma di *valori analogici*.

2. I dispositivi IoT pre-processano tali dati e trasformano quelli utili in *valori digitali* pronti per essere inviati.

3. I dati in formato digitale vengono trasferiti nel *cloud* o nei *data center*[2].

4. Medici, professionisti del settore sanitario o intelligenze artificiali accedendo a tali informazioni possono analizzarle e prendere le decisioni opportune informando i pazienti direttamente o tramite notifica ai dispositivi wearable o mobile.

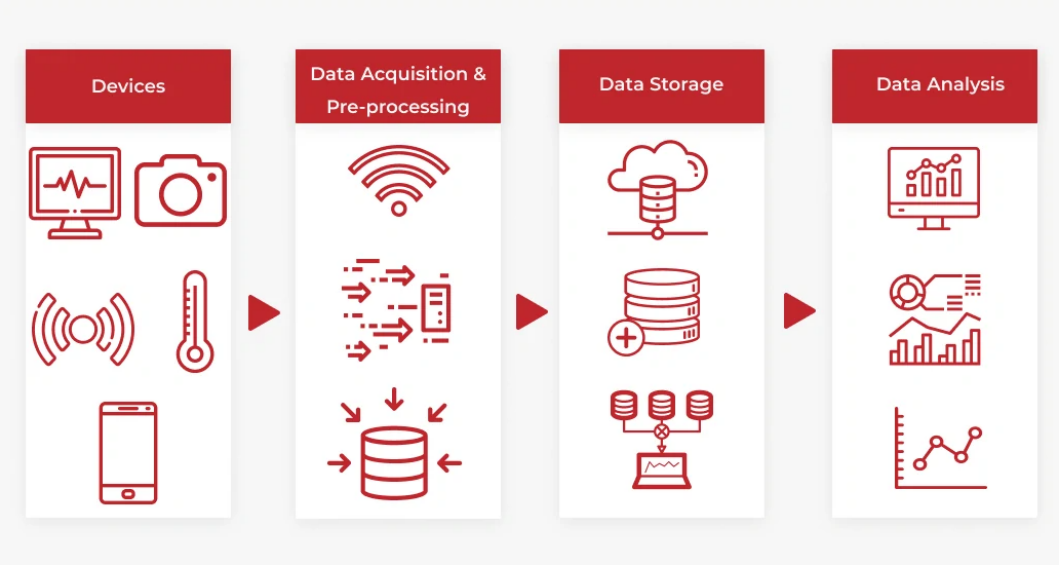


Figura 1.1: Stadi funzionamento dell'IoT nell'Healthcare

[2]**Data center:** unità organizzativa aziendale composta da server, storage, apparati di rete, cablaggi, armadi ecc.. che coordina e mantiene tutte le apparecchiature ed i servizi di gestione dei dati con lo scopo di supportare le attività aziendali.

1.2.2 Alcuni Esempi

Sistema di monitoraggio continuo del glucosio Eversense Il numero crescente di persone diabetiche ha portato a una domanda di innovazioni nei prodotti per il controllo del diabete maggiore di anno in anno. Secondo *L'International Diabetes Federation* [4] nel 2019 erano 463 milioni le persone che vivevano con questa malattia (1 adulto su 10) e questo numero è destinato ad aumentare a 700 milioni entro il 2045.

Il *sistema di monitoraggio continuo del glucosio Eversense* è una risposta a questa domanda poiché permette di monitorare continuamente fino a 180 giorni i livelli di glucosio nel sangue in tempo reale. È costituito da un sensore che viene inserito sotto la pelle, uno Smart Transmitter (rimovibile e ricaricabile) e un'app.

Il Transmitter, indossato sopra il sensore, oltre ad alimentarlo riceve i dati da quest'ultimo e li invia, tramite Bluetooth, all'app. Inoltre, in caso i livelli siano troppo bassi o alti, avvisa l'utente con una vibrazione. L'applicazione invece si occupa di mantenere lo storico di tutti i dati ricevuti e anche di mostrare all'utente una previsione dei valori futuri associando la tempistica di quanto velocemente verranno raggiunti, in modo che possa agire con sicurezza. Ovviamente notifica anche il soggetto prima che il glucosio raggiunga i livelli di allerta. [1]

Inalatori Intelligenti Anche l'asma purtroppo è una condizione che influisce sulla vita di tantissime persone in tutto il mondo. L'IoHT anche in questo caso cerca di facilitare la convivenza con questa malattia attraverso gli inalatori intelligenti. L'azienda *Propeller Health* ha creato un sensore che si installa negli inalatori e che tiene traccia di ogni inalazione. Tale sensore, grazie alla tecnologia Bluetooth, è in grado di connettersi a una specifica app mobile la quale aiuta le persone con asma e BPCO (malattia polmonare ostruttiva cronica) a capire cosa potrebbe causare i loro sintomi, tenere traccia degli usi dei farmaci di soccorso, notificare l'utente in caso di dimenticanza di questi ultimi e ricercare anche l'inalatore nel caso in cui sia stato smarrito. [3]

1.2.3 Vantaggi

L'applicazione dell'IoT in ambito medico porta vantaggi non solo ai pazienti ma anche al personale sanitario, ai manager e agli impiegati IT.

Vantaggi per i pazienti:

- Notifica rapida e semplice del problema al personale medico, da parte del paziente o di un dispositivo.

- Monitoraggio continuo delle condizioni dei pazienti e salvataggio dei loro parametri.
- Uso dell'assistenza medica a distanza che comporta maggior sicurezza per il paziente (e meno costi per il sistema sanitario).
- Maggior facilità nella gestione di complesse procedure di somministrazione dei farmaci tramite promemoria automatici che riducono così la probabilità del verificarsi di episodi acuti dovuti alla dimenticanza dell'assunzione dei medicinali.
- Consulti medici a distanza che rendono più efficiente il servizio sanitario riducendo i tempi di attesa negli ospedali.

Vantaggi per i medici:

- Accesso sempre disponibile alla storia medica completa dei pazienti, inclusi i dati più recenti.
- Aiuto da parte di algoritmi in grado di rilevare anomalie nei dati.
- Monitoraggio dei parametri vitali dei pazienti dimessi di recente che aiuta a cogliere eventuali complicazioni in anticipo, riducendo i rischi e i ricoveri ospedalieri.
- Facile localizzazione dei pazienti con difficoltà di orientamento (es. quelli affetti da demenza).
- Riduzione di errori grazie alla fornitura di dati precisi dai sistemi IoT rispetto alle informazioni sommarie riferite oralmente dal paziente.

Vantaggi per i manager e personale IT:

- Automatizzazione della manutenzione e applicazione della manutenzione predittiva che consente di prevedere e prevenire i guasti grazie al costante controllo dello stato di salute dei macchinari e attrezzature.
- Analisi più semplice del consumo di risorse.
- Diminuzione dei costi.
- Maggior facilità nell'analisi sanitaria relativa alla tendenza della salute o agli effetti di vari farmaci su molti pazienti che fa conseguire così un miglioramento del servizio sanitario offerto.

1.2.4 Sfide

La gestione dei dati sanitari degli individui è un argomento complesso che va affrontato con serietà e competenza e che vede in gioco diversi attori: le strutture sanitarie, i medici, le tech companies e il governo.

La prima sfida da affrontare è che, proprio perché si tratta di dati preziosi, altamente sensibili e personali, va fatto il possibile per *assicurarne la privacy e scongiurare possibili attacchi informatici*. L'architettura IoHT infatti implica la dipendenza dal cloud store centralizzato, il che significa che i fornitori (generalmente le tech companies) devono riconoscere la massima priorità nel mantenimento corretto di tali dati e assicurarne anche la non perdita per errore.

Altre barriere da affrontare per poter implementare su vasta scala l'IoHT sono *l'input massiccio di dati generati* dai dispositivi connessi che fanno conseguire problematiche di sovraccarico del sistema e *l'infrastruttura software obsoleta* che incrementa ancora di più il costo già elevato dell'implementazione di tale innovazione. Prima di poterla installare e renderla operativa è infatti necessario svolgere un'attenta attività di progettazione, analisi e sviluppo che richiede molti investimenti da parte del governo.

Anche gli *utenti finali sono purtroppo ancora oggi prevenuti* sull'uso della tecnologia in ambito sanitario per i rischi connessi alla sicurezza e privacy. Le istituzioni mediche dovrebbero diffondere la voce sui vantaggi di tale utilizzo e creare consapevolezza che questo cambiamento è per il meglio.

1.3 Ospedale 4.0

Come conseguenza dell'IoHT, nasce un nuovo modo di concepire l'ospedale: da centro medico polifunzionale nel quale il paziente è oggetto di cure a centro di fornitura di servizi sanitari di cui gli individui con necessità di aiuto medico ne sono usufruenti. [5]

Nell'**Ospedale 4.0, o Smart Hospital** in inglese, quindi viene reingegnerizzato e automatizzato tutto il possibile per poter fornire dei servizi che si adeguano completamente alle esigenze dei pazienti e per facilitare anche il lavoro dei medici. Questo comporta una rivoluzione nell'erogazione dell'assistenza sanitaria, nel monitoraggio medico, nella gestione delle sale d'attesa e nell'uso delle sale operatorie.

Come descritto anche nel libro *The healthcare digital revolution* [6], in primis in un Ospedale 4.0 ogni singolo metro quadro è cablato e *fornisce accesso alla rete Internet* tramite un Wi-Fi con elevata disponibilità di banda, necessaria per rispondere all'uso ultra-intensivo da parte del personale sanitario, degli oggetti intelligenti, dei pazienti e dei loro accompagnatori. In sala operatoria *i*

robot "chirurghi" conquistano il loro posto affianco ai medici che vengono anche supportati da *occhiali e caschi per la realtà aumentata* durante gli interventi. Anche le *cartelle cliniche diventano elettroniche* e non solo raccolgono i dati inseriti dagli operatori sanitari ma anche tutti quelli provenienti dai dispositivi che sono connessi in rete e che automatizzano la misurazione di parametri vitali, le somministrazioni di sostanze e l'effettuazione di prelievi.

Infine, una caratteristica essenziale per avere uno Smart Hospital è che il *monitoraggio*, non solo dei pazienti, ma anche del personale e delle scorte di materiale medico, *sia automatizzato* il più possibile.

È in questo ambito che il progetto "padre" di quello discusso in questa tesi di laurea si colloca. Viene rimandato nel prossimo capitolo la descrizione di che cosa significa fare tracciamento e l'introduzione a tale sistema.

Ovviamente questa rivoluzione non può avvenire istantaneamente ma già si sta assistendo a un'incrementale digitalizzazione e automatizzazione degli ospedali in diverse parti del mondo.

Capitolo 2

Il tracciamento in ambito Healthcare

Dopo aver trattato nel precedente capitolo come l'applicazione delle tecnologie 4.0 può generare dei vantaggi in ambito sanitario, portando anche alla nascita di un Ospedale 4.0, si era evidenziato come punto focale di innovazione la necessità di un *tracciamento automatizzato*.

Ora verrà quindi introdotto il significato e l'importanza del *tracciamento in ambito sanitario* e sarà portato come progetto esemplificativo dell'applicazione di tecnologie intelligenti destinate al monitoraggio Trauma Tracker: sistema principale dal quale si estende il prototipo trattato con questa tesi. Dopo aver fatto una panoramica sullo stato attuale di tale progetto e sui limiti del tracciamento preospedaliero corrente, verrà presentato l'applicativo PreH come soluzione per il monitoraggio di questa fase.

Nell'ultima sezione saranno anche esplicitati gli obiettivi della tesi per mettere in chiaro l'apporto del mio lavoro.

2.1 Tracciamento in ambito sanitario

Per **tracciamento in ambito sanitario** si intende la raccolta, il mantenimento e l'analisi di tutti i dati utili relativi alla salute di un individuo (anagrafica, parametri vitali, terapia farmacologica attuale, patologie pregresse, farmaci somministrati ecc...) ed alla collezione delle informazioni inerenti alla sua posizione.

A *livello ospedaliero* il monitoraggio non si ferma al singolo individuo ma può comprendere anche le azioni e la posizione del personale medico (soccorritori, medici, infermieri, paramedici) e le movimentazioni delle giacenze del materiale sanitario.

2.1.1 Importanza del tracciamento

Una documentazione completa, corretta e accurata del tracciamento è cruciale per diverse ragioni.

La prima tra tutti è che è stato dimostrato nell'articolo *Relationship between nursing documentation and patients' mortality* [13], che la *scarsa qualità* nei report sanitari *comporta una maggiore mortalità* dei pazienti, poiché le decisioni sulle azioni da compiere e il coordinamento tra gli operatori sanitari si basano su queste pratiche.

In secondo luogo, sono un'importante fonte di dati che possono essere *utilizzati a posteriori per valutare la prestazione del team e l'organizzazione ospedaliera*, permettendo dunque di rilevare eventuali problematiche o miglioramenti necessari. Possono essere impiegati anche in medicina legale per un'analisi a fini legali o assicurativi.

In ambito sanitario, in particolare nel contesto delle patologie tempo dipendenti (trauma, ictus e patologie cardiache), è possibile suddividere il tracciamento in due grandi fasi: quella interna all'ospedale definita **intraospedaliera** e quella che precede l'entrata in ospedale del paziente detta invece **preospedaliera.** Verranno ora approfonditi questi concetti, in primis attraverso la presentazione del sistema *Trauma Tracker* che si inserisce nell'ambito del tracciamento della patologia tempo dipendente del trauma e successivamente con l'introduzione della sua estensione per la gestione della fase preospedaliera: *il progetto PreH.*

2.2 Trauma Tracker

2.2.1 Descrizione e motivazioni del progetto

Il progetto **Trauma Tracker** è stato sviluppato grazie alla collaborazione tra il *gruppo di ricerca del Dipartimento di Informatica - Ingegneria E Scienze Informatiche (DISI)* e il *Trauma Team* dell'ospedale Bufalini di Cesena.

È un *sistema di tracciamento*, che sfruttando le tecnologie IoT, permette di aiutare i medici a gestire specifiche situazioni di emergenza che coinvolgono pazienti traumatizzati. Solitamente, in molti ospedali italiani tra cui il Bufalini di Cesena (ma anche in quelli Europei), quando si verificano queste circostanze viene designato un **Trauma Team**[1] che si deve occupare delle prime cure e

[1]**Trauma Team:** di base è composto da un Team leader (Anestesista-Rianimatore, chirurgo, medico dell'emergenza), un medico specialista, un infermiere solitamente specialista in Area Critica, un medico radiologo e dal personale di supporto (Operatore Tecnico addetto all'Assistenza -OTA-, Operatore Socio Sanitario -OSS-).

della stabilizzazione del paziente critico giunto presso il dipartimento di emergenza ospedaliero. All'interno del team è il **Trauma Leader** che ha il compito di compilare la documentazione oltre a coordinare la rianimazione, supervisionare il lavoro e spesso aiutare anche gli altri membri della squadra. Da questo, si può ben capire, che spesso il leader non riesce a riempire i moduli in real-time ma solo al termine della situazione di emergenza poiché è impegnato in altri compiti primari per mantenere in vita il paziente. Però, dato che si è una situazione di urgenza, sono molti gli eventi che accadono in poco tempo ed è facile che egli non riesca a tenere traccia nella memoria di tutto, non potendosi solo dedicare alla raccolta di informazioni. Oltre a ciò, tali dati vengono poi convertiti da cartacei a digitali da un altro operatore e questo aumenta ancora di più la probabilità di errore.

2.2.2 Obiettivi e funzionalità

L'obiettivo principale di Trauma Tracker è proprio cercare di evitare il problema descritto nella sottosezione precedente promuovendo la generazione di documentazione digitale completa (senza dati mancanti), corretta e accurata, riducendo così anche il carico di lavoro per il Trauma Leader. Le due principali funzionalità del sistema infatti sono:

- **automatizzare per quanto più possibile la memorizzazione in real-time di informazioni rilevanti** come cambiamenti dei parametri vitali, identità e posizione del Trauma Leader, luogo e ora di un evento e permettere un veloce inserimento di quest'ultimo nell'applicazione.

  Per *eventi* si intendono i cambiamenti nello stato[2] del paziente e le somministrazioni, i trattamenti, i test diagnostici e le manovre effettuate.

  Attualmente per poter tracciare in maniera automatizzata la posizione del paziente e del Trauma Leader è utilizzato il cosiddetto *Trauma Tracker Location Service*. È un servizio basato su un'infrastruttura costituita da **BLE**[3] **beacon** disposti in ogni stanza del percorso trauma. Ciascuno di essi è connesso alla rete dell'ospedale e monitora continuamente la presenza o meno di un *tag ricevitore* che viene indossato dal leader dell'equipe che gestisce l'emergenza.

  Per il monitoraggio automatico dei parametri vitali, invece, ci si affida al *Trauma Tracker Gateway Service* che permette di recuperare i parametri vitali dai **sensori IoT** connessi al paziente.

[2]**Stato del paziente:** stato del battito cardiaco (bradicardia, tachicardia), autonomia o meno nella respirazione, presenza o meno di un'emorragia esterna, tre punteggi della Glascow Coma Scale (risposta all'apertura degli occhi, risposta verbale, risposta motoria) ecc..

[3]**BLE:** Bluetooth Low Energy.

- **generare automaticamente report digitali** sulla base dei dati raccolti, evitando così la conversione dal cartaceo e riducendo notevolmente anche il tempo e il lavoro necessario.

2.2.3 Risultati ottenuti

Trauma Tracker è operativo da più di un anno ed i risultati della sua applicazione sono soddisfacenti. Infatti, come indicato nell'articolo *Real-time tracking and documentation in trauma management* [12], dopo la validazione del prototipo tutti i medici sono d'accordo che grazie al sistema:

- si può effettuare una *ricostruzione retrospettiva precisa dei fatti* grazie anche alla memorizzazione del luogo e dell'ora di ogni evento che con la documentazione cartacea spesso non venivano raccolti.
- *la completezza delle informazioni* raccolte *è incrementata* notevolmente, in particolare per quelle relative allo stato del paziente.
- *l'accuratezza dei dati è migliorata* soprattutto per quelli inerenti ai segni vitali e agli esami del sangue.
- *il tempo* necessario per produrre la documentazione *è diminuito.*

2.2.4 Scopo finale

In realtà, Trauma Tracker, che ad oggi gestisce principalmente la parte del trauma dentro l'ospedale, ha come *obiettivo finale quello di estendersi* per coprire il tracciamento in diversi ambiti che possono essere classificati in due macro-categorie: **preospedaliera e intraospedaliera**. Il monitoraggio inizia con la fase preospedaliera la quale viene gestita dal dipartimento di emergenza, poi, all'interno dell'ospedale segue e si completa con un percorso specifico a seconda della patologia tempo dipendente del paziente: trauma, ictus o infarto. Per questo progetto così pervasivo nell'articolo *Pervasive tracking for time-dependent acute patient flow: a case study in trauma management* [11] si propone un'architettura orientata ai servizi[4] che comprende vari sottosistemi che interagiscono tra loro.

Si esprime quindi la necessità di avere un applicativo indipendente che gestisca tutta la parte preospedaliera e che vada poi a trasmettere le informazioni utili raccolte, ai sottosistemi che gestiscono il tracciamento internamente all'ospedale: nasce così **il progetto PreH**.

[4]**Architettura orientata ai servizi:** è un pattern architetturale per la costruzione di sistemi e applicazioni basato sui servizi. Ogni servizio incapsula una funzionalità specifica che può essere anche riusata.

2.3 Il tracciamento preospedaliero e il progetto PRE-H

Per **tracciamento in ambito preospedaliero** si intende la raccolta, il mantenimento e l'analisi delle informazioni relative alla parte che precede l'entrata in ospedale di un soggetto per il quale è stato chiamato il soccorso ospedaliero. Il monitoraggio quindi inizia con la chiamata al 118 e termina nel momento nel quale il paziente viene lasciato in ospedale.

I dati che vengono collezionati sono relativi sia alla salute dell'individuo che alle azioni e alla posizione del personale sanitario (medici, paramedici e infermieri) che effettuano il soccorso.

2.3.1 Attuazione del tracciamento preospedaliero corrente

Dal confronto con i medici dell'ospedale Bufalini di Cesena si è evinto che, quando si riceve una chiamata di soccorso, viene abilitato un **team PreH** che si deve occupare di raggiungere il paziente (con ambulanza o elisoccorso), stabilizzarlo e poi portarlo in ospedale il più velocemente possibile. È il leader di tale squadra che deve compilare durante il soccorso un report cartaceo con una struttura predefinita per ogni soggetto che verrà trasportato in ospedale. Questo documento sarà poi consegnato al medico prestabilito (in caso di paziente traumatizzato sarà il Trauma Leader) per consentirgli di avere un quadro generale delle condizioni dell'individuo assistito.

Nella Figura 2.1 è riportata la relazione di soccorso utilizzata negli ospedali dell'Emilia Romagna per raccogliere i dati della fase preospedaliera.

In altri casi, o sono i soccorritori che contattano telefonicamente la centrale operativa del 118 informandoli, oppure il leader del team PreH comunica le informazioni utili durante una breve conversazione con lo specialista sanitario che è stato incaricato di occuparsi del paziente al momento del suo arrivo in ospedale. Se il soggetto ha subito un trauma, ovviamente tale specialista sarà il Trauma Leader che si occuperà, nel frattempo, di inserire manualmente le informazioni all'interno dell'applicazione Trauma Tracker, nella specifica sezione relativa ai dati preospedalieri.

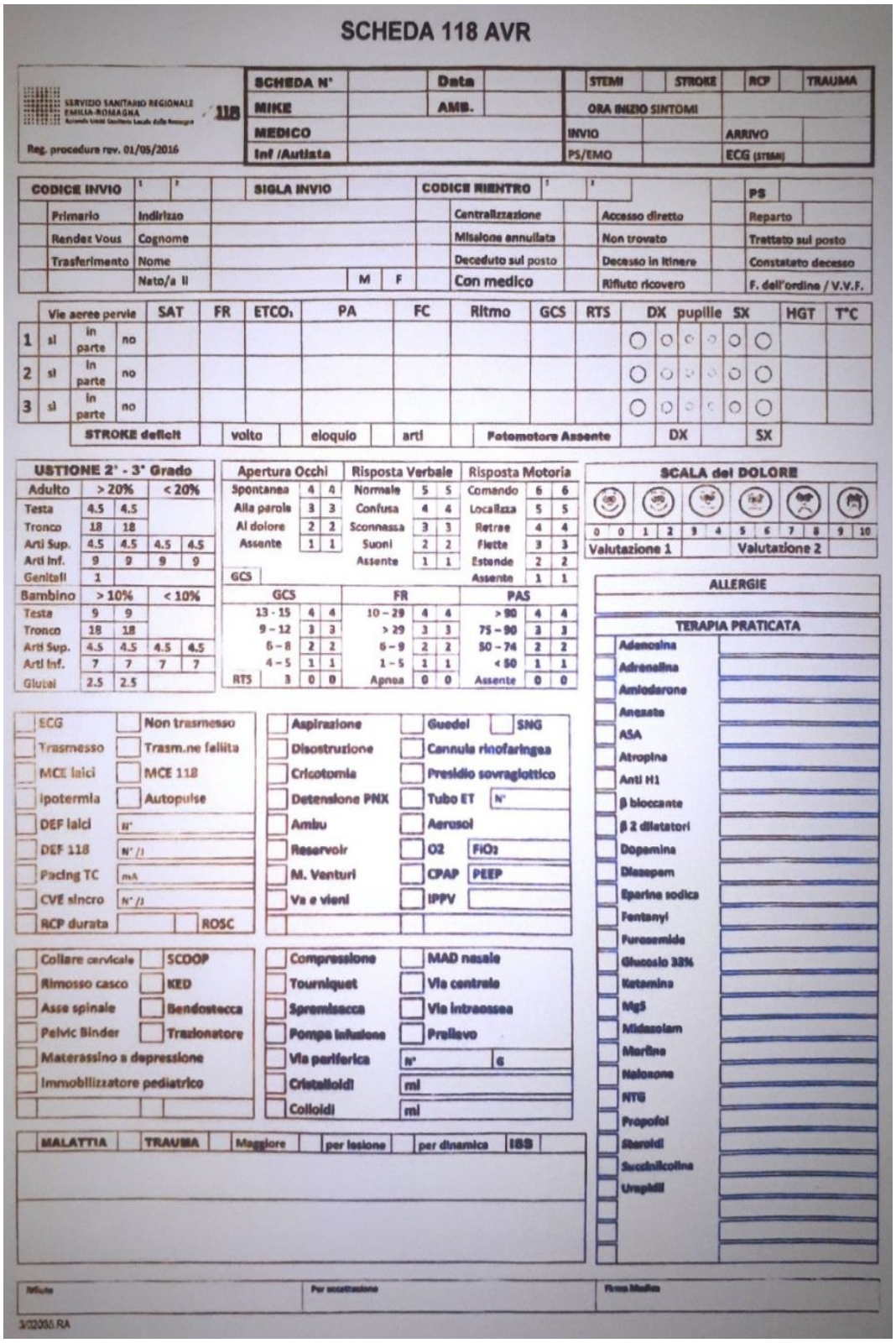


Figura 2.1: Scheda cartacea utilizzata attualmente nel soccorso preospedaliero

Problemi Ci sono diversi problemi che derivano da questo modus operandi:

1. Nella maggior parte dei casi, i medici in ospedale *non hanno nessuna informazione sul paziente che dovranno assistere fino a quando egli non arriva effettivamente al pronto soccorso.* Questo significa che non possono prepararsi anticipatamente ad eventuali interventi (ad esempio aprendo e predisponendo una sala operatoria), non possono chiamare preventivamente un medico specializzato e sono anche costretti a decidere come agire sull'individuo soccorso solo al momento della sua entrata in ospedale. In tal modo si perdono minuti preziosi che, soprattutto nel caso delle patologie tempo-dipendenti, possono anche significare la sopravvivenza o meno del paziente.

2. Nelle poche situazioni, nelle quali gli operatori sanitari in ospedale hanno a disposizione alcuni dati, questi sono stati comunicati loro dalla centrale operativa che a sua volta li ha ricevuti dal personale sanitario impiegato nel soccorso. *Le informazioni ottenute in questo modo sono spesso sommarie e con un'alta probabilità d'errore* poiché vengono trasmesse oralmente e perché tale comunicazione coinvolge più soggetti.

   Suddetta problematica si può ricondurre ugualmente quando, al momento dell'arrivo in ospedale, il leader PreH riassume velocemente a voce gli avvenimenti al medico che si occuperà della fase intraospedaliera. Se poi quest'ultimo dovrà anche inserire le informazioni manualmente dentro un'applicazione come Trauma Tracker, *la possibilità del verificarsi di un'inesattezza è ancor più alta.*

3. I dati inseriti nella documentazione preospedaliera dal leader del team PreH sono ovviamente *soggetti ai suoi possibili errori.* Nella Sottosezione 2.2.1 abbiamo visto le problematiche che causano la produzione di dossier inaffidabili nella fase intraospedaliera, le quali possono essere ovviamente riportate anche per questo stadio di soccorso: multitasking del leader, molti eventi che accadono in poco tempo, documentazione compilata non in real-time ma successivamente con i dati ricordati a mente e conversione da cartaceo a digitale dei report. Oltre a questi ostacoli i soccorritori devono anche spesso agire in ambienti e condizioni ostili per conformazione del territorio o condizioni climatiche.

2.3.2 Il progetto PreH

Come indicato precedentemente nella Sottosezione 2.2.4 **il progetto PreH** nasce come estensione di Trauma Tracker, specializzandosi nel *tracciamento della fase preospedaliera.* Si occupa cioè di gestire il periodo che inizia con la

chiamata al 118 effettuata per conto o dalla persona da soccorrere e termina con l'arrivo di quest'ultima in ospedale; dopo di che, se il paziente ha subito un trauma grave, il controllo del tracciamento viene passato a Trauma Tracker. PreH è un sistema comunque indipendente nel funzionamento, infatti si occupa di gestire **tutte le patologie** e non solo i pazienti traumatizzati rappresentando in questo modo un passo aggiuntivo per il raggiungimento di un monitoraggio pervasivo e completo.

Descrizione e Obiettivi Nella Sottosezione 2.1.1 si è evidenziato quanto sia fondamentale la documentazione per fornire un servizio sanitario sicuro e migliore. Il progetto PreH persegue, similmente a Trauma Tracker, *l'obiettivo di aiutare il leader della squadra di soccorso nella redazione di una documentazione digitale corretta, completa e accurata* che comporta come conseguenza l'eliminazione di schede cartacee e anche la risoluzione del problema (3) indicato nella Sottosezione 2.3.1.

Il sistema cercherà infatti di automatizzare il più possibile la raccolta delle informazioni sia relative al paziente che al team, e, per quelle che non è possibile reperire autonomamente, di renderne facile e veloce l'inserimento per il PreH leader, che potrà così compilare i report in real-time durante il soccorso senza perdere troppo tempo. Anche in questo caso, come per Trauma Tracker, grazie a PreH si potranno raccogliere in maniera automatica alcune informazioni fondamentali come il luogo e l'ora per ogni evento, che in forma cartacea spesso vengono omesse.

Un altro obiettivo che si cerca di raggiungere è quello di *poter informare preventivamente i medici all'interno dell'ospedale.* PreH prevede infatti l'invio in tempo reale di ogni informazione immessa o raccolta durante il soccorso.

Questo, oltre a risolvere i problemi (1) e (2), comporta anche una conseguenza positiva ulteriore: poiché come detto sopra il tracciamento svolto dal sistema non è solo specifico per il paziente, i medici all'interno dell'ospedale potranno sapere in ogni momento la posizione del team preospedaliero e che cosa stanno facendo. Il monitoraggio della squadra PreH è utile anche per effettuare un'analisi a posteriori delle difficoltà e dei problemi che si sono riscontrati, permettendo così un miglioramento nel servizio offerto (ad esempio si potrebbe constatare un ritardo di partenza dell'equipaggio perché l'elisoccorso non era pronto).

Infine, un traguardo molto importante che consente di raggiungere il sistema è quello di *generare report automaticamente che possono essere facilmente integrati in Trauma Tracker*, evitando pertanto al Trauma Leader l'inserimento manuale di tutte le informazioni preospedaliere (e se presente anche l'anagrafica del paziente). In questo modo viene appianata anche la seconda complicazione citata nel punto (2) della Sottosezione 2.3.1.

2.3.3 Lavori Correlati

ViTTS A sostegno del progetto PreH presentato con questa tesi si porta il sistema sviluppato e validato dall'Università di medicina di Utrecht (Paesi Bassi) denominato **online Victim Tracking and Tracing System**, o in breve **ViTTS** [10].

Tale progetto si colloca in un dominio applicativo molto critico e caotico da gestire ovvero quello degli incidenti su larga scala (crolli di grattacieli, forti terremoti, importanti alluvioni ecc..). In questi casi, facilmente la domanda di cure sanitarie può eccedere la disponibilità delle risorse e per questo motivo *occorre ottimizzarne la dislocazione*. In particolare, due decisioni molto importanti da affrontare sono quali e quanti mezzi di soccorso impiegare e come assegnare le vittime agli ospedali. In tali situazioni è essenziale anche garantire *la tracciabilità e rintracciabilità dei pazienti* per evitare confusione e di commettere errori. Oltre a ciò deve essere presente *una risposta immediata e una coordinazione perfetta tra le varie organizzazioni* che intervengono in soccorso: dipartimento di polizia, quello dei vigili del fuoco, quello medico e quello governativo.

Queste condizioni vengono però di frequente a mancare a causa di *problemi di comunicazione* dovuti al fatto che ogni ente ha un proprio sistema il quale è scarsamente interoperabile con gli altri. Ovviamente ciò comporta conseguenze negative nella gestione generale della situazione: spesso il numero di vittime, il loro stato di salute e il luogo nel quale si trovano, sono sconosciuti. Tale circostanza, oltre a poter causare una risposta inefficace all'emergenza, porta anche angoscia e frustrazione ai familiari delle vittime che cercano di rintracciarli.

ViTTS nasce per offrire un servizio adeguato di tracciamento e rintracciamento delle vittime di incidenti su larga scala sia nella parte preospedaliera che quella intraospedaliera e rendere tali informazioni facilmente accessibili per tutte le organizzazioni autorizzate. Il sistema si compone di un *bus dati centrale* collegato da una parte ai database esistenti delle varie organizzazioni (polizia, ospedali, enti governativi ecc..) e dall'altra a un database principale protetto dove vengono salvati tutti i dati inseriti durante l'emergenza. Tali informazioni vengono poi distribuite nel bus e ricevute solo dagli utenti autorizzati. In questo modo ogni organizzazione ottiene i dati aggiornati per cui compete in tempo reale.

Sono stati condotti numerosi test per valutare la fattibilità, l'usabilità e la robustezza del sistema tra cui una simulazione in uno scenario realistico. La circostanza riproduceva un incidente causato da un camion cisterna, contenente acido solforico, che aveva sfondato le mura di un edificio ed aveva provocato

40 feriti. Con solo un'ora di istruzione sul funzionamento di ViTTS per gli operatori impiegati nel soccorso, ci sono stati risultati molto positivi.

Per tale ragione ci si aspetta che l'impiego di PreH nel contesto reale e la sua integrazione con il sistema già operativo per l'ambito intraospedaliero, comporti esiti altrettanto proficui.

2.4 Obiettivi della tesi

La tesi si pone come obiettivo principale quello di effettuare un potenziamento e un'estensione del prototipo PreH, appartenente all'ambito del sistema Tracking For Care, in parte già esistente [9].

Da un lato ci si concentrerà sul *migliorare tutti gli aspetti che ad oggi non rendono funzionale* l'uso del sistema nella realtà per i medici soccorritori, attraverso la **riprogettazione, rifattorizzazione** e conseguentemente **reimplementazione** di alcune parti. Dall'altro verranno *implementate nuove funzionalità* per perfezionare l'applicativo offerto.

Un altro scopo che ci si impegna a raggiungere è quello di *ottimizzare tutte le caratteristiche* relative a come il sistema funziona e a come è implementato:

- Si cercherà di rendere più intuitiva e semplice l'interfaccia grafica.
- Si renderanno adattabili le schermate dall'applicativo a dispositivi con schermi di grandezze differenti.
- Si cercherà di ottimizzare l'uso della memoria occupata.

Infine, verrà svolta una *validazione molto approfondita* del sistema verificando il suo corretto funzionamento in diversi scenari e analizzando i possibili successivi miglioramenti necessari. Saranno anche richiesti dei feedback sul nuovo prototipo ad un medico dell'ospedale Bufalini di Cesena, scelto come referente, per comprendere se le richieste degli utenti finali sono state completamente soddisfatte e in quale direzione proseguire.

Nei prossimi capitoli verranno illustrate, in accordo con gli standard definiti dall'Ingegneria del Software, le principali fasi che hanno portato alla realizzazione del nuovo prototipo PreH.

Capitolo 3

Analisi del dominio e dei requisiti

In questo capitolo verrà presentata la prima fase del ciclo di vita del software: **l'analisi**. Essa ha lo scopo generale di chiarire, dettagliare e documentare le funzioni, i servizi e le prestazioni che devono essere offerti da un sistema software. Tali informazioni, che rappresentano il punto di accordo tra l'utente finale e il progettista, vengono raccolte all'interno di un documento chiamato **specifica dei requisiti** che viene prodotto al termine di questo step e che sarà il punto di partenza per le successive fasi di progettazione, sviluppo e validazione.

Il progetto PreH era già stato avviato in precedenza [9] ed in tale studio era stata effettuata una prima analisi del dominio e dei requisiti e sviluppata un'iniziale versione prototipale del sistema solo relativa al trauma. L'analisi che verrà eseguita sarà un refactoring che includerà i punti focali di quella svolta precedentemente ma sarà integrata con i nuovi fattori emersi dal mio studio che hanno l'obiettivo di migliorare ed estendere il prototipo esistente.

3.1 Modello del dominio

Quando si riceve una chiamata di soccorso, il leader del team PreH dovrà poter avviare una nuova missione inserendo il suo nominativo, il tipo di mezzo utilizzato ed eventualmente il codice della missione all'interno di un'**applicazione** precedentemente installata sul tablet rugged[1] che gli viene

[1]**Rugged:** uno smartphone o tablet rugged è un dispositivo capace di sopravvivere a condizioni estreme (cadute da grandi altezze, immersioni in acqua, schiacciamento) e di resistere anche alla polvere e ad altissime temperature, grazie alle caratteristiche fisiche con cui viene progettato.

fornito dall'ospedale. Una volta fatto ciò, è necessario che possa immettere ogni spostamento rilevante della squadra di soccorso, dal momento nel quale parte dall'ospedale al momento della consegna dell'individuo assistito al dipartimento di emergenza ospedaliero.

Quando viene raggiunto il paziente, il leader PreH, dopo aver effettuato su di lui un'attenta analisi, dovrà avere la possibilità di registrare nell'applicazione i farmaci somministrati, le manovre e i trattamenti effettuati e le complicanze eventualmente sopravvenute. Oltre a questo, è opportuno che il programma applicativo permetta di inserire lo stato ed i parametri vitali del paziente e, se è possibile identificarlo, anche l'anagrafica. Ci dovranno anche essere delle apposite schermate per la visualizzazione e gestione dei criteri di identificazione di trauma maggiore.

Durante lo svolgimento del soccorso il leader PreH dovrà poter visionare in ogni momento lo storico delle operazioni effettuate e complicazioni avvenute relative al paziente. È opportuno anche che l'applicazione metta a disposizione una schermata per inserire maggiori dati relativi all'evento che ha generato la missione come l'indirizzo, la dinamica, se è un intervento secondario o primario, il numero di missioni attive su tale evento, il numero di pazienti coinvolti ed il codice di invio.

Nel caso ideale in cui il tablet sia sempre connesso a Internet, ogni dato inserito dovrà essere inviato immediatamente ad un **servizio di back-end**, il quale provvederà ad immagazzinarlo e renderlo disponibile in real-time **nella dashboard** alla quale hanno accesso i medici all'interno dell'ospedale. Da quest'ultima sarà possibile generare un report pdf contenente tutte le informazioni registrate relative al soccorso, che potrà così essere incluso nella cartella clinica e/o archiviato alla centrale operativa.

Dovrà, infine, essere possibile terminare il tracciamento in ogni momento inserendo il codice di rientro, l'ospedale di arrivo ed opzionalmente il luogo di rilascio (pronto soccorso o sala operatoria). Per ogni missione che si intende concludere verrà data la possibilità di scegliere se eliminarla, se inviarla al servizio di back-end o se salvarla localmente sul dispositivo. Un soccorso si intende completamente terminato solo se viene inviato con successo al back-end. Il salvataggio, invece, è utile in caso la rete Internet non sia disponibile, infatti, permette di non perdere i dati e di iniziare una nuova missione senza rimanere bloccati a causa dell'attesa dell'invio di quella precedente. Le missioni memorizzate ovviamente dovranno poter essere inoltrate (e quindi terminate) successivamente quando la connessione sarà ripristinata.

3.1.1 Casi d'uso

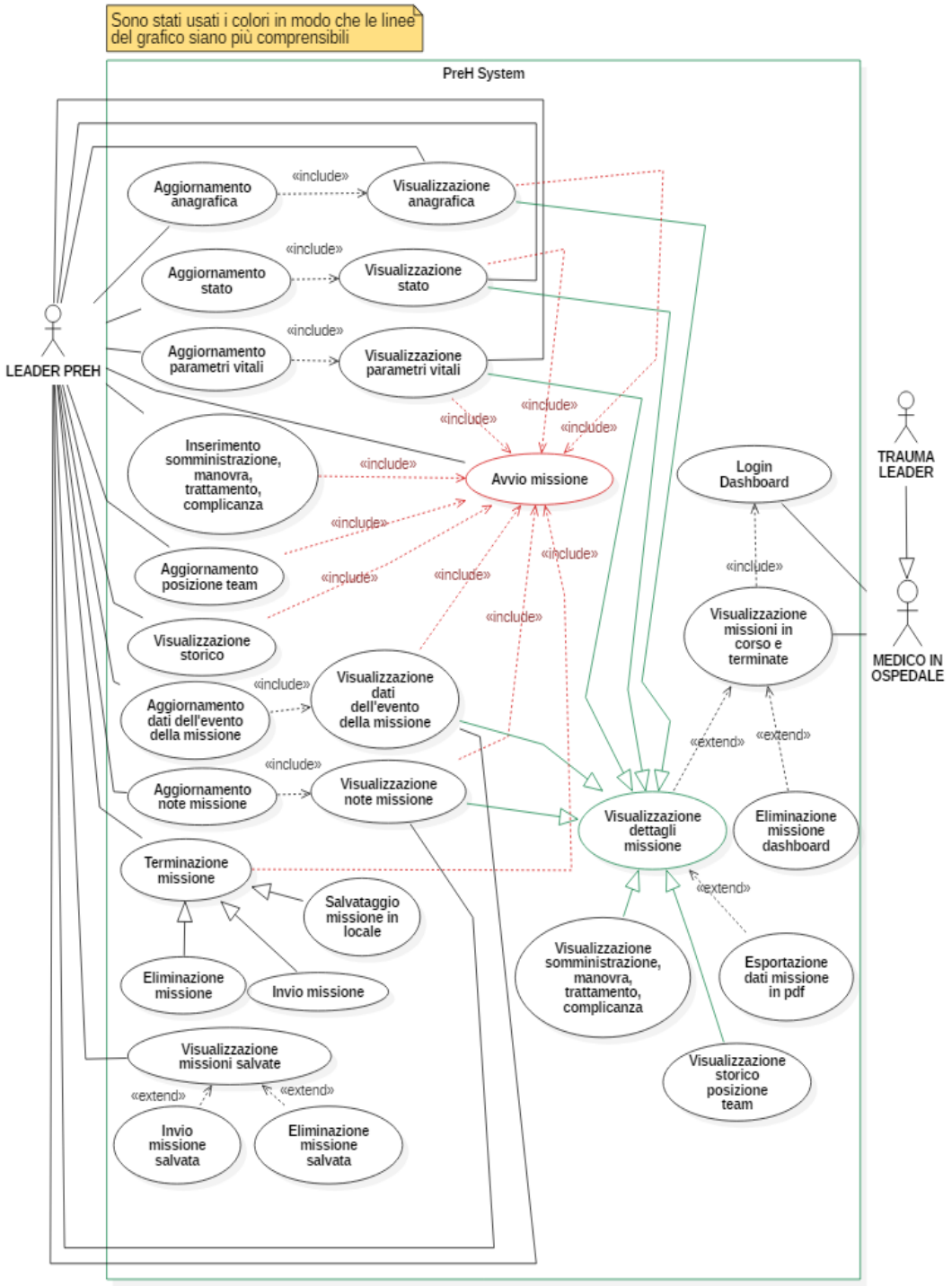


Figura 3.1: Diagramma casi d'uso sistema PreH

I **casi d'uso** è un diagramma UML di tipo dinamico[2] che descrive le funzionalità del sistema come vengono percepite dagli utenti e dagli analisti. Non specifica la struttura del software cioè "come" i task devono essere realizzati.

È molto importante poiché nella *fase di analisi* permette di effettuare in maniera esaustiva e non ambigua la raccolta dei requisiti. Ragionare sui casi d'uso con il committente consente infatti di scoprire e definire chiaramente che cosa il sistema dovrà fare.

Inoltre, essi guidano anche *l'intero progetto di sviluppo*: sono il punto di partenza per la progettazione del sistema ed anche il riferimento primario per la definizione e l'esecuzione dei test di verifica di quanto prodotto.

Nella sezione successiva verranno elencati i possibili **scenari**: ciascuno di essi è un'istanza di un caso d'uso cioè rappresenta una sua specifica esecuzione. L'obiettivo è quello di riuscire a ricavare per ogni caso d'uso *i prerequisiti* ed anche le eventuali *varianti* prodotte da scenari di insuccesso o diversi da quello base.

3.1.2 Scenari d'utilizzo

UC1 Avvio Missione

Scenario Base	1. Il medico avvia l'applicazione. 2. Il medico inserisce il mezzo usato per il soccorso ed il suo nominativo. 3. Il medico preme sul pulsante per avviare una nuova missione.
Varianti	• 2(a) Il medico inserisce anche il codice della missione opzionale.

UC2 Aggiornamento posizione team

Prerequisiti	• La missione deve essere avviata. • Se lo step non è il primo occorre che quello precedente sia stato già raggiunto.
Scenario Base	1. Il medico seleziona nell'app lo step appena raggiunto. 2. Il medico inserisce le informazioni richieste e conferma. 3. L'applicazione memorizza anche l'ora dell'inserimento.

[2]**Diagrammi UML dinamici:** descrivono le interazioni tra gli oggetti del sistema.

Varianti	• 1(a) Se il medico erroneamente seleziona uno step successivo rispetto a quello raggiunto non si deve dare la possibilità di impostarlo come step corrente. • 2(a) Nel caso in cui lo step sia il caricamento del paziente non occorre inserire il luogo perché è uguale a quello dello step precedente (l'arrivo sul paziente). • 2(b) Nel caso in cui lo step sia l'arrivo in pronto soccorso non occorre inserire il luogo perché è, per logica, il pronto soccorso.

UC3 Visualizzazione dati evento della missione

Prerequisiti	• La missione deve essere avviata. • Se la visualizzazione avviene tramite dashboard, il medico deve aver prima fatto il login e aperto la schermata riepilogativa dei dettagli della missione.
Scenario Base	1. Il medico visualizza le informazioni relative all'evento della missione accedendo nella schermata specifica dell'app o osservando la sezione apposita nella dashboard.
Varianti	• 1(a) Se nessun dato relativo all'evento della missione è stato ancora inserito, ovviamente i campi della schermata e della sezione saranno vuoti.

UC4 Aggiornamento dati evento della missione

Prerequisiti	• La missione deve essere avviata.
Scenario Base	1. Il medico accede nell'app, tramite apposito pulsante, alla schermata relativa all'evento della missione. 2. Il medico aggiorna le informazioni relative all'evento della missione.

UC5 Visualizzazione note missione

Prerequisiti	• La missione deve essere avviata. • Se la visualizzazione avviene tramite dashboard, il medico deve aver prima fatto il login e aperto la schermata riepilogativa dei dettagli della missione.
Scenario Base	1. Il medico visualizza le note della missione aprendo la finestra di dialogo specifica nell'applicazione o osservando la sezione apposita nella dashboard.
Varianti	• 1(a) Se nessuna nota relativa alla missione è stata ancora inserita, ovviamente la finestra di dialogo e la sezione saranno vuote.

UC6 Aggiornamento note missione

Prerequisiti	• La missione deve essere avviata.
Scenario Base	1. Il medico visualizza nell'app le note della missione. 2. Il medico aggiorna le note della missione. 3. Il medico conferma la modifica.
Varianti	• 3(a) Se il medico non conferma la modifica le note non vengono aggiornate.

UC7 Visualizzazione storico

Prerequisiti	• La missione deve essere avviata.
Scenario Base	1. Il medico accede nell'app, tramite apposito pulsante, alla schermata relativa allo storico. 2. Il medico visualizza lo storico.
Varianti	• 2(a) Se ancora non è stata registrata nessuna manovra, trattamento, somministrazione o complicanza, lo storico sarà vuoto.

UC8 Visualizzazione anagrafica

Prerequisiti	• La missione deve essere avviata. • Se la visualizzazione avviene tramite dashboard, il medico deve aver prima fatto il login e aperto la schermata riepilogativa dei dettagli della missione.
Scenario Base	1. Il medico visualizza l'anagrafica del paziente accedendo nella schermata specifica dell'app o osservando la sezione apposita nella dashboard.
Varianti	• 1(a) Se nessun dato relativo all'anagrafica è stato ancora inserito, ovviamente i campi della schermata e della sezione risulteranno vuoti.

UC9 Aggiornamento anagrafica

Prerequisiti	• La missione deve essere avviata.
Scenario Base	1. Il medico accede nell'app, tramite apposito pulsante, alla schermata dell'anagrafica del paziente. 2. Il medico aggiorna l'anagrafica.

UC10 Visualizzazione stato

Prerequisiti	• La missione deve essere avviata. • Se la visualizzazione avviene tramite dashboard, il medico deve aver prima fatto il login e aperto la schermata riepilogativa dei dettagli della missione.
Scenario Base	1. Il medico visualizza lo stato del paziente accedendo nella specifica schermata dell'app o osservando l'apposita sezione nella dashboard.
Varianti	• 1(a) Se nessun dato relativo allo stato del paziente è stato ancora inserito, ovviamente i campi della schermata e della sezione saranno vuoti.

UC11 Aggiornamento stato

Prerequisiti	• La missione deve essere avviata.
Scenario Base	1. Il medico accede nell'app, tramite apposito pulsante, alla schermata dello stato del paziente. 2. Il medico aggiorna lo stato.

UC12 Visualizzazione parametri vitali

Prerequisiti	• La missione deve essere avviata. • Se la visualizzazione avviene tramite dashboard, il medico deve aver prima fatto il login e aperto la schermata riepilogativa dei dettagli della missione.
Scenario Base	1. Il medico visualizza i parametri vitali del paziente accedendo nella specifica schermata dell'app o osservando la tabella apposita nella dashboard.
Varianti	• 1(a) Se nessun parametro vitale relativo al paziente è stato ancora inserito, ovviamente i campi della schermata e la tabella saranno vuoti.

UC13 Aggiornamento parametri vitali

Prerequisiti	• La missione deve essere avviata.
Scenario Base	1. Il medico accede nell'app, tramite apposito pulsante, alla schermata dei parametri vitali del paziente. 2. Il medico aggiorna i parametri vitali.

UC14 Inserimento somministrazione

Prerequisiti	• La missione deve essere avviata. • L'equipaggio deve essere arrivato sul posto.

Scenario Base	1. Il medico accede nell'app alla schermata delle somministrazioni. 2. Il medico seleziona il farmaco da somministrare. 3. Il medico inserisce il dosaggio. 4. Il medico conferma la somministrazione.
Varianti	• 4(a) Se il medico non conferma la somministrazione il farmaco non viene registrato come somministrato.

UC15 Inserimento manovra

Prerequisiti	• La missione deve essere avviata. • L'equipaggio deve essere arrivato sul posto.
Scenario Base	1. Il medico accede nell'app alla schermata delle manovre. 2. Il medico attiva la manovra desiderata.
Varianti	• 2(a) Se il medico seleziona una manovra già attiva questa viene disattivata. • 2(b) Se la manovra desiderata è il pacing transcutaneo non si parla di attivazione ma di inserimento. Occorre, in tal caso, immettere anche i valori richiesti.

UC16 Inserimento trattamento

Prerequisiti	• La missione deve essere avviata. • L'equipaggio deve essere arrivato sul posto.
Scenario Base	1. Il medico accede nell'app alla schermata dei trattamenti. 2. Il medico seleziona il trattamento desiderato. 3. Il medico inserisce i dati richiesti. 4. Il medico conferma l'esecuzione del trattamento.

Varianti	• 2(a) Se il trattamento desiderato è relativo alla sezione delle vie aeree o alla minitoracotomia allora per inserirlo basta attivare lo switch corrispondente. Ovviamente, in tal caso, se il trattamento era già stato attivato, questo viene disattivato. • 2(b) Se il medico vuole effettuare un trattamento a tempo deve attivare sempre prima il timer. • 3(a) Per alcuni trattamenti non è necessario inserire dati. • 4(a) Se il medico non conferma il trattamento, questo non viene registrato come eseguito.

UC17 Inserimento complicanza

Prerequisiti	• La missione deve essere avviata. • L'equipaggio deve essere arrivato sul posto.
Scenario Base	1. Il medico accede nell'app alla schermata delle complicanze. 2. Il medico attiva la complicanza desiderata.
Varianti	• 2(a) Se il medico seleziona una complicanza già attiva questa viene disattivata.

UC18 Terminazione missione

Prerequisiti	• La missione deve essere avviata.
Scenario Base	1. Il medico accede nell'app alla schermata di terminazione missione. 2. Il medico inserisce il codice di rientro e l'ospedale di arrivo. 3. Il medico termina la missione inviandola.

Varianti	• 2(a) Se era già stato inserito precedentemente il luogo di atterraggio, l'ospedale di arrivo corrisponderà a quello di tale località. • 2(b) Il medico può opzionalmente inserire anche il luogo di rilascio del paziente. • 3(a) Se la rete non è presente o il server non è raggiungibile, l'invio non va a buon fine e la missione viene automaticamente salvata sul dispositivo. • 3(b) Il medico può decidere di terminare la missione anche salvandola direttamente sul dispositivo senza tentare l'invio. • 3(c) Il medico può scegliere come ulteriore opzione, oltre all'invio e al salvataggio, anche l'eliminazione della missione.

UC19 Visualizzazione missioni salvate

Scenario Base	1. Il medico avvia l'app. 2. Il medico accede alla schermata contenente la lista delle missioni salvate localmente sul dispositivo tramite apposito pulsante presente nella pagina iniziale.
Varianti	• 2(a) Se nessuna missione è stata ancora salvata sul dispositivo, l'elenco sarà vuoto.

UC20 Eliminazione missione salvata

Prerequisiti	• Almeno una missione deve essere salvata sul dispositivo.
Scenario Base	1. Il medico avvia l'app. 2. Il medico accede alla schermata contenente la lista delle missioni salvate localmente sul dispositivo tramite apposito pulsante presente nella pagina iniziale. 3. Il medico elimina la missione desiderata premendo sull'opportuno tasto.

UC21 Invio missione salvata

Prerequisiti	• Almeno una missione deve essere salvata sul dispositivo.
Scenario Base	1. Il medico avvia l'app. 2. Il medico accede alla schermata contenente la lista delle missioni salvate localmente sul dispositivo tramite apposito pulsante presente nella pagina iniziale. 3. Il medico sceglie la missione da trasmettere e preme sull'opportuno pulsante per iniziare l'invio. 4. L'invio va a buon fine e la missione viene eliminata dal dispositivo locale.
Varianti	• 4(a) Se la rete non è presente o il server non risponde, l'invio non va a buon fine e si dovrà riprovare successivamente.

UC22 Login Dashboard

Scenario Base	1. Il medico in ospedale accede alla pagina web della dashboard. 2. Il medico inserisce le sue credenziali. 3. Il medico effettua il login.
Varianti	• 3(a) Il login può non andare a buon fine se le credenziali sono errate.

UC23 Visualizzazione missioni in corso e terminate

Prerequisiti	• Login alla dashboard
Scenario Base	1. Il medico visualizza l'elenco delle missioni in corso e terminate in due tabelle distinte nella home della dashboard.

UC24 Visualizzazione dettagli missione

Prerequisiti	• Login alla dashboard

Scenario Base	1. Il medico sceglie dall'elenco delle missioni in corso e terminate visualizzate nella home della dashboard, una di esse della quale vederne maggiori informazioni. 2. Il medico apre la schermata contenente i dettagli della specifica missione premendo sull'apposito pulsante. 3. Il medico visualizza tutte le informazioni relative alle somministrazioni, manovre, trattamenti, complicanze, parametri vitali, stato e anagrafica del paziente che sono state registrate, le note ed i dati della missione che sono stati inseriti, le informazioni sull'evento immesse e lo storico delle varie posizioni raggiunte dal team.

UC25 Eliminazione missione dashboard

Prerequisiti	• Login alla dashboard
Scenario Base	1. Il medico visualizza l'elenco delle missioni in corso e terminate in due tabelle distinte nella home della dashboard. 2. Il medico elimina la missione desiderata premendo sull'apposito pulsante.

UC26 Visualizzazione somministrazione, manovra, trattamento, complicanza

Prerequisiti	• La missione deve essere avviata. • Il medico deve aver prima fatto il login alla dashboard e aperto la schermata riepilogativa dei dettagli della missione.
Scenario Base	1. Il medico accede alla scheda interessata scelta tra quella delle somministrazioni, delle manovre, dei trattamenti e delle complicanze. 2. Il medico visualizza in una tabella l'elenco dettagliato degli eventi della scheda scelta.
Varianti	• 2(a) Se nessun evento della scheda scelta è stato ancora inserito, ovviamente la tabella sarà vuota.

UC27 Visualizzazione storico posizione team

Prerequisiti	• La missione deve essere avviata. • Il medico deve aver prima fatto il login alla dashboard e aperto la schermata riepilogativa dei dettagli della missione.
Scenario Base	1. Il medico osserva nella sezione relativa allo storico dei vari spostamenti del team le informazioni comprensive di luogo e timestamp.
Varianti	• 1(a) Se nessun spostamento è stato ancora registrato, ovviamente la sezione sarà vuota.

UC28 Esportazione dati missione in pdf

Prerequisiti	• La missione deve essere avviata. • Il medico deve aver prima fatto il login alla dashboard e aperto la schermata riepilogativa dei dettagli della missione.
Scenario Base	1. Il medico scarica un pdf, tramite apposito pulsante, contenente un riepilogo di tutte le informazioni relative alla specifica missione che sono visualizzate anche sulla schermata della dashboard.
Varianti	• 1(a) Se nessun dato è stato ancora inserito, il pdf sarà ovviamente vuoto.

3.1.3 Glossario

- **Missione:** rappresenta un intervento di soccorso al quale è associato un medico e un mezzo. La missione comprende anche tutti i dati relativi al paziente, le azioni svolte su di lui dal personale medico, le eventuali complicanze verificatasi e anche le informazioni relative sia all'evento che ha generato la missione che al tracciamento della posizione del team di intervento.

- **Medico / leader PreH:** soggetto responsabile del soccorso e anche del tracciamento.

- **Team PreH / squadra di soccorso / equipaggio:** team di medici, infermieri e paramedici incaricati di effettuare il soccorso del paziente.
- **Paziente:** persona che necessita il soccorso.
- **Centrale operativa 118:** organo addetto alla ricezione delle richieste di soccorso, alla coordinazione e invio di medici e veicoli nei vari interventi.
- **Codice di invio:** codice alfanumerico composto da tre parti variabili; la prima è una lettera che denota il luogo dell'evento, la seconda è composta dalla lettera C, che sta per codice, seguita da un numero da 01 a 20 che indica la patologia prevalente e la terza è un carattere a scelta tra B,V,G,R che codifica la gravità dell'emergenza. Viene prodotto dal centro operativo a seguito di alcune domande poste telefonicamente. Nella Figura 3.2 vengono riassunti i significati dei vari codici che costituiscono le tre parti di quello di invio.

CODICE DEL LUOGO	
K	CASA
L	LAVORO
P	LUOGO PUBBLICO
Q	SCUOLA
S	STRADA
T	TRASFERIMENTO
Y	IMPIANTO SPORTIVO
Z	ALTRO LUOGO

CODICE GRAVITÀ	
R	Emergenza
G	Urgenza primaria
V	Urgenza differibile
B	Non urgenza

CODICE PATOLOGIA	
C01	TRAUMATOLOGICO
C02	CARDIOCIRCOLATORIO
C03	RESPIRATORIO
C04	NEUROLOGICO
C05	PSICHIATRICO
C06	NEOPLASTICO
C07	TOSSICOLOGICO
C08	METABOLICO
C09	GASTROENTEROLOGO
C10	UROLOGICO
C11	OCULISTICO
C12	OTORINO
C13	DERMATOLOGICO
C14	OSTETRICO
C15	INFETTIVO
C19	ALTRA PATOLOGIA
C20	NON IDENTIFICATO

Figura 3.2: Significato Codici

- **Step:** posizioni nelle quali si può trovare il team di soccorso. Gli step possibili sono: partenza equipaggio, arrivo sul posto, arrivo sul paziente, caricamento paziente, partenza dal posto, atterraggio e arrivo in PS.
- **Step corrente:** posizione attuale nella quale si trova il team di soccorso. Solo uno step può essere considerato come corrente.

- **Codice di rientro:** numero che viene fornito dal leader PreH al termine della missione e rappresenta la gravità del paziente. Il range di valori che può assumere è compreso tra 1 (non sussiste emergenza) e 4 (paziente deceduto).
- **Missione in corso:** missione che non è stata ancora completamente inviata al servizio di back-end.
- **Missione terminata:** missione che è stata inviata completamente con successo al servizio di back-end. Una missione può essere anche conclusa sul dispositivo tablet ma se non è stata ancora inoltrata, non può considerarsi terminata.
- **Trauma maggiore:** evento traumatico caratterizzato da lesioni in grado di determinare un rischio immediato o potenziale per la sopravvivenza del paziente.
- **Intervento primario:** missione svolta usando l'elisoccorso che ha come obiettivo primario quello di recuperare il paziente e trasportarlo fino all'ospedale più adeguato per le sue condizioni.
- **Intervento secondario:** missione svolta usando l'elisoccorso che ha come obiettivo primario il trasporto di un paziente critico da un ospedale all'altro.
- **Numero di missioni attive sull'evento interessato:** corrisponde al numero di mezzi di soccorso impiegati per uno stesso evento.

3.1.4 Diagrammi UML del modello

Dopo aver analizzato attentamente i casi d'uso ed elencato i relativi scenari, verrà ora raffigurato **il modello dei dati del dominio** utilizzando i *diagrammi delle classi UML*. In particolare, verranno mostrate le entità principali del sistema indicandone i loro campi e metodi più rilevanti e saranno anche definiti esplicitamente i vincoli e le interdipendenze che sussistono tra di esse.

Missione, Evento e Paziente

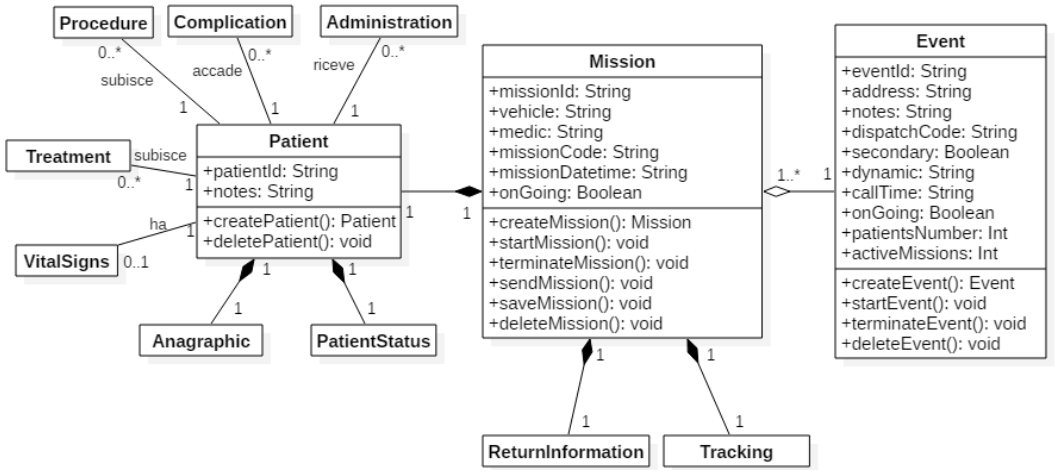


Figura 3.3: Diagramma classi UML delle entità missione-evento-paziente

La Figura 3.3 rappresenta le tre *entità base* del sistema e le relazioni che sussistono tra loro. Ogni evento può generare una o più missioni che includono le informazioni di ritorno e il tracciamento del team. Ciascuna missione è relativa ad uno specifico evento e paziente, di quest'ultimo si vuole memorizzare l'anagrafica, il suo stato e, se presenti, eventuali somministrazioni, complicazioni, manovre, trattamenti ed elenco di segni vitali. In seguito verranno mostrate nel dettaglio le *entità secondarie* più rilevanti.

Tracciamento Team

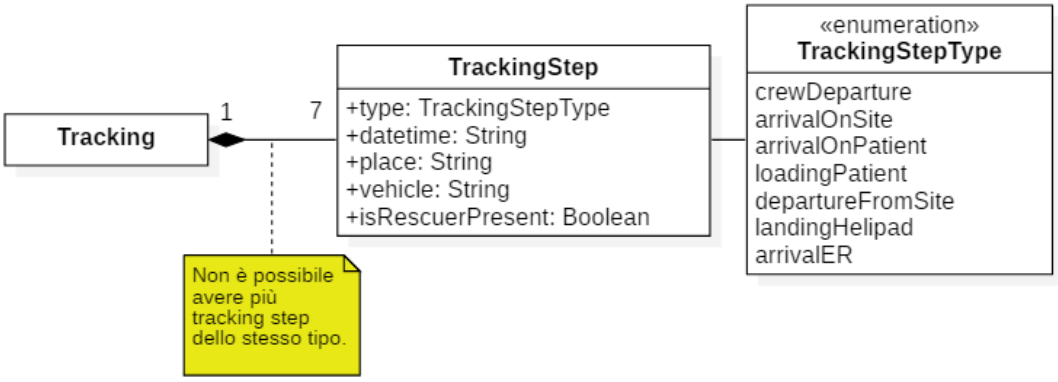


Figura 3.4: Diagramma classi UML del tracciamento del team

Il tracciamento del team di una specifica missione si compone di 7 step dei quali per ciascuno se ne vuole memorizzare il tipo, il tempo di inserimento, il luogo, il veicolo utilizzato per il soccorso e se il leader PreH è presente o meno. Come indicato nell'annotazione della Figura 3.4, il tracciamento non può contenere più step dello stesso tipo, quindi ognuno di essi avrà un genere diverso rispetto agli altri, scelto tra i membri del set dell'enumerazione *TrackingStepType*.

Segni Vitali

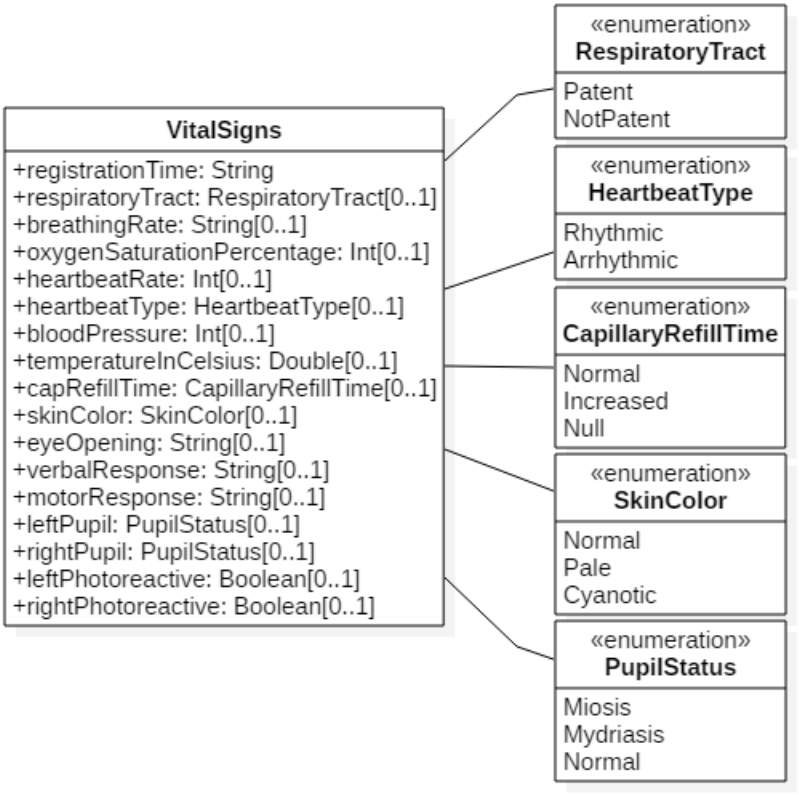


Figura 3.5: Diagramma classi UML dei segni vitali

Per ogni paziente soccorso si vuole mantenere un elenco di tutti i segni vitali più recenti, fondamentali per elaborare una diagnosi e le azioni da intraprendere. Poiché non sempre vengono effettuate tutte le misurazioni e i controlli, tutti i campi sono opzionali [0..1] tranne il tempo nel quale avviene la registrazione che è obbligatorio. Per alcuni parametri il valore deve essere scelto tra il set di nomi messo a disposizione nella corrispondente enumerazione come mostrato in Figura 3.5.

Stato Paziente

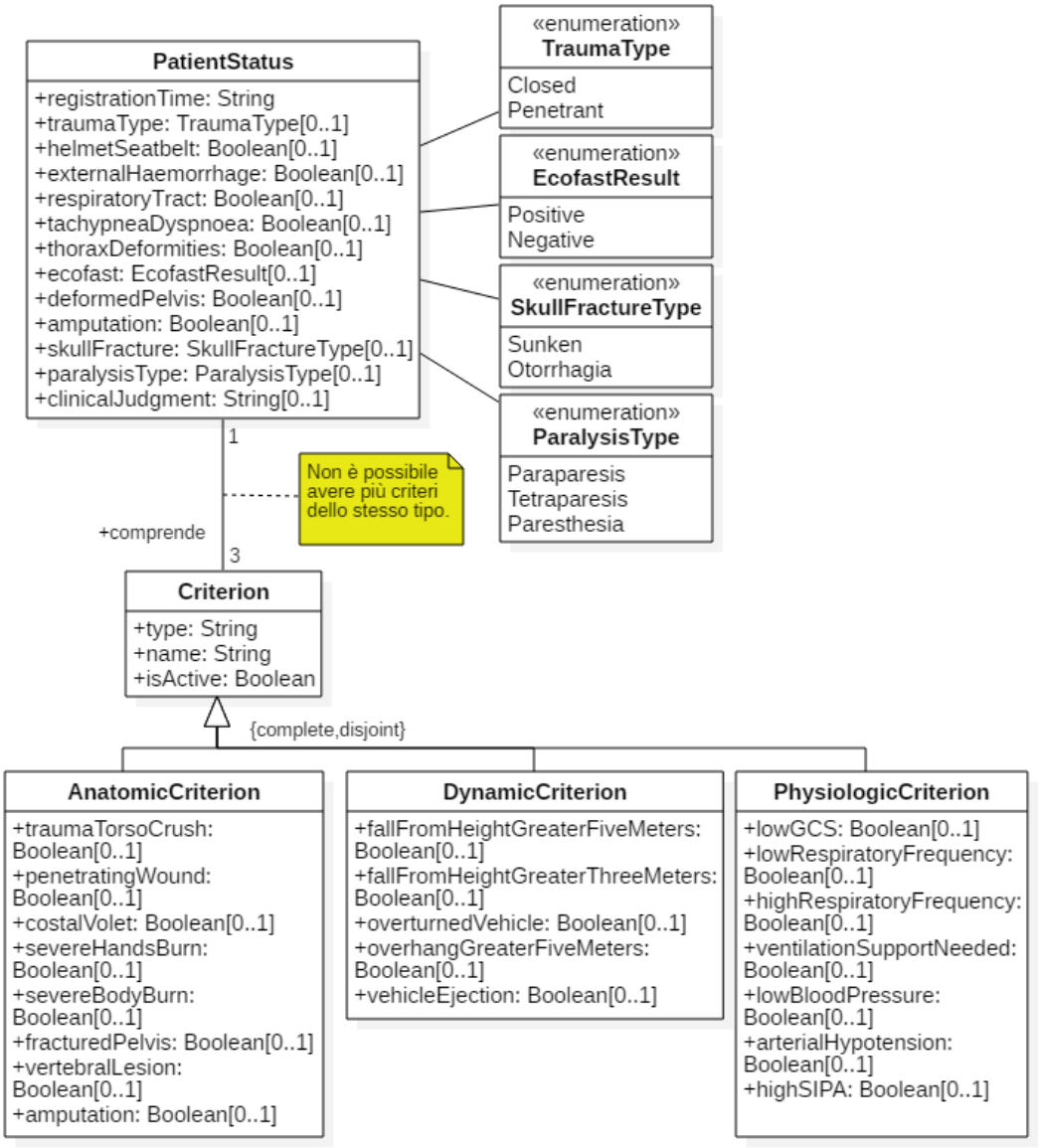


Figura 3.6: Diagramma classi UML dello stato del paziente

Per ogni paziente si richiede anche che sia mantenuto un riepilogo del suo stato poiché aiuta a definire chiaramente quanto sono gravi le sue condizioni. Esso si compone di vari parametri, tutti quanti opzionali tranne il tempo di registrazione, visto che alcuni di essi potrebbero essere assenti o non necessario inserirli. Il valore di alcuni campi deve essere obbligatoriamente scelto tra il set di nomi presenti nell'enumerazione specifica (vedi *traumaType, ecofast,*

skullFracture e paralysisType nella Figura 3.6). Nel caso in cui il medico voglia specificare indicazioni aggiuntive, non gestite dagli attributi presenti nell'entità *PatientStatus*, può farlo inserendole nel *clinicalJudgment*.

Lo stato del paziente comprende anche il riferimento a tre criteri che definiscono la presenza o meno di un trauma maggiore: *criterio anatomico, dinamico e fisiologico*. Per ciascuno di essi si vuole memorizzare il tipo, il nome e se è attivo o meno. Il criterio è attivo quando uno dei campi booleani di cui è composto ha valore *true*.

Trattamenti

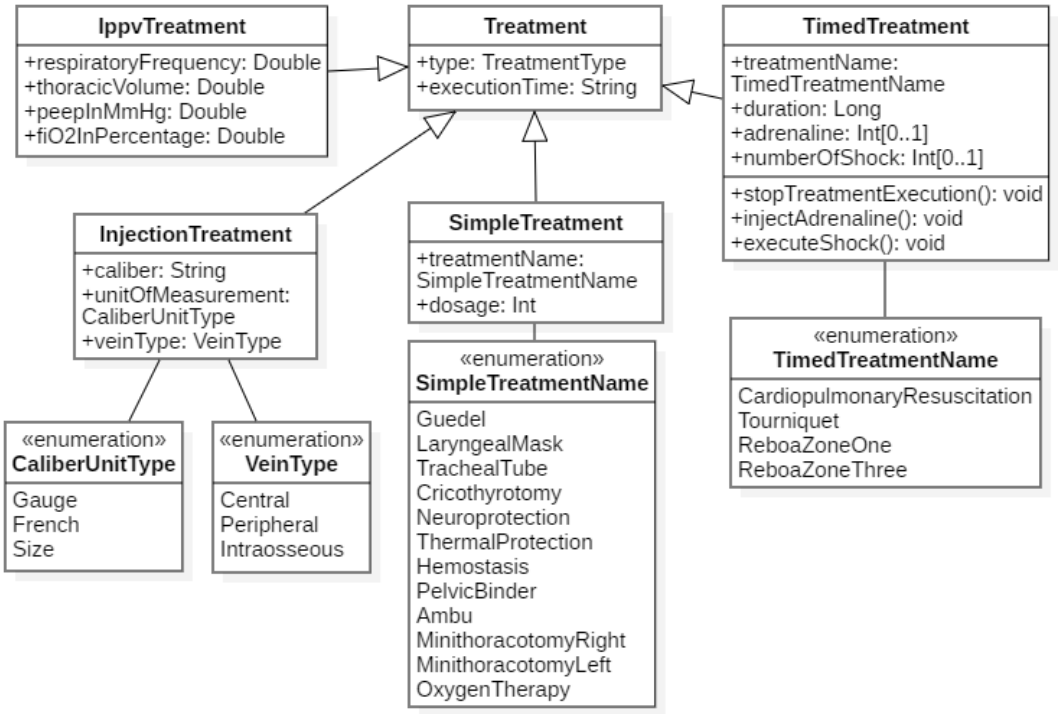


Figura 3.7: Diagramma classi UML dei trattamenti

Ci sono varie tipologie di trattamenti che possono essere eseguiti su uno specifico paziente durante la fase di soccorso: *semplici, IPPV, a tempo e ad iniezione*. Per tutti è di fondamentale importanza conoscerne il momento di esecuzione e la tipologia. I trattamenti semplici sono inoltre caratterizzati dal nome, scelto tra quelli disponibili nell'enumerazione corrispondente *SimpleTreatmentName*, e da un dosaggio. Quelli a tempo, ovviamente sono descritti dal nome (sempre a scelta tra i campi dell'enumerazione *TimedTreatmentName*) e da una durata. Se il trattamento a tempo scelto è la rianimazione

cardiopolmonare si ha anche la possibilità di registrare eventuali dosi di adrenalina somministrate e gli shock effettuati (per tale motivo questi attributi sono opzionali [0..1]). Anche i trattamenti a iniezione e quelli IPPV hanno campi specifici che vengono memorizzati solo per la loro tipologia: per quelli a iniezione è importante salvare in quale tipo di vena avviene (centrale, periferica, intraossea) e il calibro dell'ago utilizzato con associata l'unità di misura scelta tra Gauge, French e Size, mentre per quelli IPPV[3] viene descritta la frequenza respiratoria, il volume toracico, la PEEP (pressione positiva di fine espirazione) e la percentuale di fiO2 (frazione inspirata di ossigeno).

Manovre

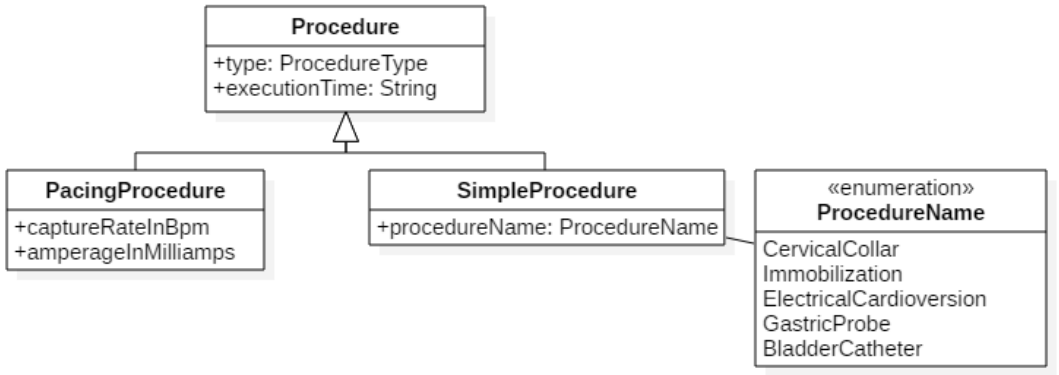


Figura 3.8: Diagramma classi UML delle manovre

È possibile che sia necessaria l'esecuzione di alcune manovre durante il soccorso di un certo paziente. Esse possono essere divise in due tipologie: *manovre semplici e di pacing.* In entrambi i casi si vuole memorizzare il tipo e il momento nel quale la manovra viene eseguita, per quelle di pacing occorre registrare anche la frequenza di stimolazione desiderata (bpm) e l'amperaggio (mA) mentre per quelle semplici si richiede di salvare solo il nome che deve essere scelto tra i campi disponibili nell'enumerazione *ProcedureName* mostrata nella Figura 3.8.

[3]**IPPV:** Intermittent Positive Pressure Ventilation.

Complicazioni

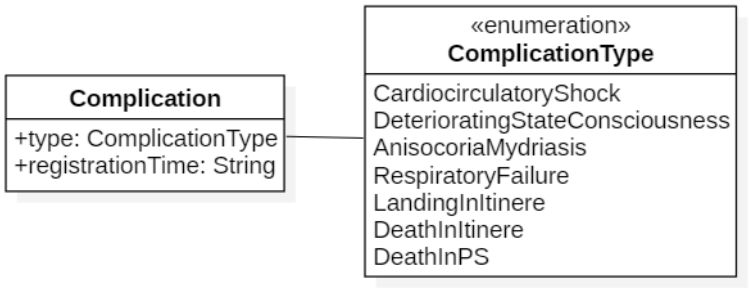


Figura 3.9: Diagramma classi UML delle complicazioni

Durante il soccorso di un paziente è possibile che si verifichino delle complicazioni delle quali per ciascuna se ne vuole memorizzare il tipo, scelto tra i campi messi a disposizione dall'enumerazione *ComplicationType* mostrata nella Figura 3.9, e il momento nel quale avviene.

Somministrazioni

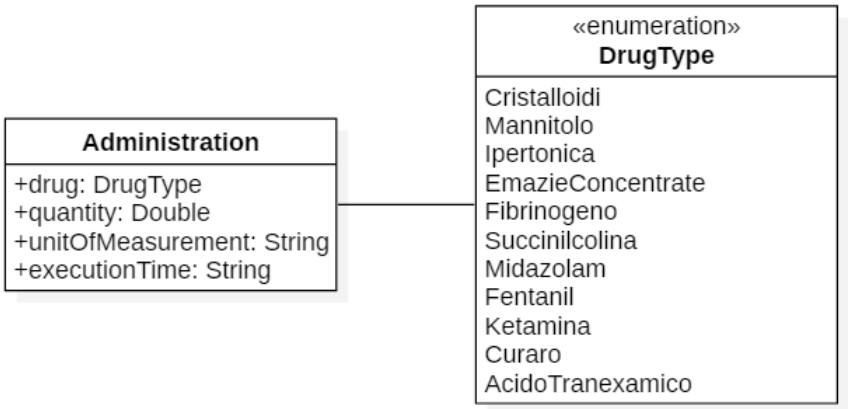


Figura 3.10: Diagramma classi UML delle somministrazioni

Le somministrazioni che vengono effettuate a un certo paziente sono descritte dal nome del farmaco, dalla dose con rispettiva unità di misura e dal tempo (data e ora) di esecuzione. I vari medicinali che si possono somministrare sono indicati nei campi che compongono l'enumerazione *DrugType* mostrata in Figura 3.10.

3.2 Analisi dei requisiti

Dopo aver effettuato una prima analisi del dominio del progetto, viene di seguito svolta l'analisi dei requisiti.

L'analisi dei requisiti è un processo che permette di stabilire con precisione e sicurezza i servizi che il committente richiede che siano offerti dal sistema (**requisiti funzionali**) e i vincoli con i quali esso deve funzionare ed essere sviluppato (**requisiti non funzionali**).

3.2.1 Requisiti funzionali

Affinché il sistema adempia agli obiettivi del progetto PreH è necessario che rispetti i seguenti requisiti funzionali:

- **Avvio della missione:** il sistema deve permettere di creare una nuova missione di soccorso dall'applicazione richiedendo obbligatoriamente il nome del medico che è a capo della squadra e il mezzo utilizzato. Verrà anche prevista la possibilità di inserire un codice missione opzionale.

  A fronte dell'avvio di una missione verrà generato dal sistema un paziente e un evento (che ha originato l'intervento), corrispondenti univocamente ad essa.

- **Sistema di autenticazione dashboard:** il sistema deve avere un servizio di login nella dashboard per poter permettere l'accesso solo agli utenti autorizzati.

- **Gestione anagrafica paziente:** una volta avviata la missione, il programma deve consentire:

  - *lato app* di modificare e visualizzare l'anagrafica del paziente.
  - *lato dashboard* di visualizzare tali dati solo in lettura a seguito dell'autenticazione e dell'apertura della schermata specifica relativa alla missione.

- **Gestione stato paziente:** una volta avviata la missione, il sistema deve dare la possibilità:

  - *lato app* di modificare e visualizzare lo stato più recente del paziente.
  - *lato dashboard* di visualizzare tali dati solo in lettura a seguito dell'autenticazione e dell'apertura della schermata specifica relativa alla missione.

Lo **stato del paziente** include il tipo di trauma subito, la presenza o meno di dispositivi di sicurezza, se sussiste un'emorragia esterna in atto, se le vie aeree sono pervie, se è presente la tachipnea e/o dispnea, se risulta un volet e/o una deformità costale, se l'ecofast è positivo o negativo, se il bacino è instabile e/o deformato, se c'è amputazione o subamputazione, se è presente una frattura cranica e infine se risulta esserci qualche tipo di paralisi.

- **Giudizio clinico:** una volta avviata la missione, l'applicazione deve permettere al medico di registrare un giudizio clinico relativo allo stato del paziente nel caso in cui i campi presenti non siano sufficienti. Ovviamente tale giudizio dovrà essere possibile visionarlo anche nella dashboard.
- **Gestione Glascow Coma Scale (GCS):** l'applicazione, dopo aver avviato la missione, dovrà permettere l'inserimento dei valori di risposta motoria, verbale e visiva del paziente e sulla base di questi calcolare il corrispondente indice di GCS. La Tabella 3.29 qui di seguito rappresenta come viene svolta la valutazione per determinarlo:

Risposta verbale	**Risposta motoria**	**Apertura occhi**	**Valore**
Nessuna	Nessuna	Nessuna	**1**
Suoni	Estensione	Stimolo pressorio	**2**
Parole	Flessione anormale	Stimolo sonoro	**3**
Confusa	Flessione normale	Spontanea	**4**
Orientata	Localizzata		**5**
	Esegue ordini		**6**

Tabella 3.29: Tabella GCS.

- **Gestione criteri di identificazione di trauma maggiore:** durante la missione, l'applicazione dovrà avere delle apposite schermate per la visualizzazione e gestione dei criteri di identificazione di trauma maggiore. Essi si dividono in tre categorie: **anatomici, dinamici e fisiologici**. I primi due saranno stabiliti direttamente del medico a seguito di un'analisi sul paziente, mentre i criteri fisiologici dovranno essere determinati automaticamente dall'applicativo in base allo stato ed ai parametri vitali dell'individuo soccorso.

- **Notifica di trauma maggiore:** se uno dei criteri citati nel punto precedente è presente, l'applicazione dovrà attivare degli allarmi di notifica (visivi o sonori) per evidenziarlo in maniera immediata al medico.
- **Gestione parametri vitali del paziente:** una volta avviata la missione, il sistema deve permettere:
  - *lato app* di modificare e visualizzare i parametri vitali più recenti del paziente.
  - *lato dashboard* di visualizzare tali dati solo in lettura a seguito dell'autenticazione e dell'apertura della schermata specifica relativa alla missione. Dovrà essere mantenuto anche uno storico di come questi valori sono variati nel tempo per permettere una valutazione dell'evoluzione delle condizioni del paziente.
- **Gestione somministrazioni:** il programma, dopo che la missione è stata avviata e il team di soccorso è arrivato sul posto, dovrà permettere:
  - *lato app* di inserire una specifica somministrazione scelta tra quelle presenti nella schermata. Esse saranno suddivise in due categorie, **infusioni** e **generici**, e per ciascuna di esse sarà necessario inserire il dosaggio somministrato al paziente.
  - *lato dashboard* di visualizzare l'elenco di tutte le somministrazioni effettuate, a seguito dell'autenticazione e dell'apertura della schermata specifica relativa alla missione.
- **Gestione manovre:** l'applicativo, dopo che la missione è stata avviata e il team di soccorso è arrivato sul posto, dovrà consentire:
  - *lato app* di aggiungere o rimuovere una specifica manovra scelta tra quelle presenti nella schermata apposita.
  - *lato dashboard* di visualizzare l'elenco di tutte le manovre effettuate, a seguito dell'autenticazione e dell'apertura della schermata specifica relativa alla missione.
- **Gestione trattamento:** dopo che la missione è stata avviata e il team di soccorso è arrivato sul posto, il sistema dovrà permettere:
  - *lato app* di aggiungere uno specifico trattamento scelto tra quelli presenti nella schermata apposita. I trattamenti saranno suddivisi in diverse categorie definite sulla base della zona anatomica interessata: **rianimazione cardiopolmonare, trattamenti vie aeree, trattamenti respiratori, trattamenti vie venose, neuroprotezione, protezione termica**.

  - *lato dashboard* di visualizzare l'elenco di tutti i trattamenti a cui il paziente è stato sottoposto, a seguito dell'autenticazione e dell'apertura della schermata specifica relativa alla missione.

- **Gestione complicanza:** dopo che la missione è stata avviata e il team di soccorso è arrivato sul posto, il sistema dovrà consentire:
  - *lato app* di aggiungere o rimuovere una specifica complicanza, scelta tra quelle presenti nella schermata apposita. Le complicanze sono tutti gli imprevisti che avvengono durante il soccorso.
  - *lato dashboard* di visualizzare l'elenco di tutte le complicanze verificatasi, a seguito dell'autenticazione e dell'apertura della schermata specifica relativa alla missione.

- **Gestione tracciamento posizione team:** una volta avviata la missione, il programma dovrà permettere il tracciamento completo della posizione del team di soccorso, in particolare:
  - *lato app* dovrà consentire di aggiungere il raggiungimento di ogni step dell'iter di soccorso. Questo si compone delle seguenti fasi: partenza equipaggio, arrivo sul posto, arrivo sul paziente, caricamento paziente, partenza dal posto, atterraggio e arrivo in PS.
  - *lato dashboard* dovrà dare la possibilità di visualizzare tutti gli step inseriti in tempo reale dalla squadra di soccorso che saranno tutti caratterizzati dal luogo e dall'ora. Ovviamente tale sezione sarà raggiungibile solo in seguito all'autenticazione e all'apertura della schermata specifica relativa alla missione.

- **Visualizzazione storico:** nell'applicazione dovrà essere presente una schermata riepilogativa delle manovre, trattamenti, somministrazioni effettuate sul paziente e delle complicanze verificatasi. Infatti, è possibile che alcuni soccorsi si protraggano per molto tempo e/o che le azioni intraprese sul paziente siano molteplici, quindi avere uno storico che possa rammentare al leader PreH ciò che stato fatto, è di fondamentale importanza.

- **Gestione dati evento della missione:** una volta avviata la missione, il sistema dovrà permettere:
  - *lato app* di inserire in una apposita schermata i dati relativi all'evento che ha generato la missione: codice di invio, dinamica, indirizzo, se è un intervento secondario o primario, il numero di missioni attive sull'evento interessato, il numero di pazienti coinvolti e delle note.

  - *lato dashboard* di visualizzare tali dati solo in lettura, in seguito dell'autenticazione e dell'apertura della schermata specifica relativa alla missione.

- **Gestione note missione:** una volta avviata la missione, il programma dovrà consentire:

  - *lato app* di immettere in un apposito spazio di inserimento delle note relative alla missione in generale.
  - *lato dashboard* di visualizzare tale testo solo in lettura, in seguito dell'autenticazione e dell'apertura della schermata specifica relativa alla missione.

- **Terminazione missione:** in qualsiasi momento il medico potrà decidere di terminare la missione inserendo il codice di rientro, l'ospedale di arrivo e opzionalmente il luogo di rilascio (pronto soccorso o sala operatoria) in una specifica schermata dell'app.

  Una volta premuto l'apposito pulsante di terminazione verranno date tre scelte all'utente: inviare la missione al servizio di back-end, salvarla sul dispositivo in locale oppure eliminarla. Se si tenta l'invio ma la rete non è disponibile o il server non risponde, la missione verrà automaticamente salvata sul tablet e potrà essere inoltrata (e terminata) successivamente. Se invece l'invio va a buon fine, la missione viene visualizzata terminata anche nella dashboard e i dati locali sul dispositivo vengono eliminati poiché superflui.

- **Gestione missioni salvate sul dispositivo:** nell'applicazione dovrà essere presente un'apposita schermata che consenta di visualizzare tutte le missioni salvate sul dispositivo cioè quelle che non sono ancora totalmente terminate. Per ciascuna di esse deve essere visualizzata la data, l'ora di inizio, il medico responsabile e la possibilità di eliminarla o tentarne l'invio al servizio di back-end.

- **Visualizzazione missioni in corso e terminate:** in seguito del login nella dashboard il sistema deve mettere a disposizione dei medici in ospedale l'elenco delle missioni ancora in corso e quelle terminate. Per ciascuna di esse deve essere presente:

  - l'id della missione.
  - il veicolo usato per il soccorso.
  - il nominativo del medico leader del team di soccorso.

  - un pulsante che permetta di visualizzare maggiori dettagli della specifica missione.
  - un pulsante che permetta di eliminare la missione.

  Inoltre, per gli interventi non ancora conclusi, deve essere indicato anche il tempo trascorso da quando sono stati iniziati.

- **Visualizzazione somministrazioni, manovre, trattamenti, complicanze:** in seguito del login nella dashboard e dell'apertura della schermata specifica relativa alla missione, dovrà essere possibile per i medici in ospedale visualizzare tutte le somministrazioni, manovre, trattamenti che i soccorritori hanno eseguito sul paziente e le complicanze verificatasi.

  In particolare, per **le somministrazioni** dovrà essere visualizzato:

  - il nome del farmaco.
  - dose del farmaco.
  - data e ora di somministrazione.

  Per **i trattamenti** verrà indicato:

  - il nome del trattamento.
  - data e ora di esecuzione.
  - un pulsante per visualizzare, se presenti, ulteriori dettagli relativi al trattamento inseriti dal soccorritore.

  Per **le manovre** sarà precisato:

  - il nome della manovra.
  - data e ora di esecuzione.
  - un pulsante per visualizzare, se presenti, ulteriori dettagli relativi alla manovra inseriti dal soccorritore.

  E, infine, per **le complicanze** sarà visualizzato:

  - il tipo della complicanza.
  - data e ora dell'avvenimento.

- **Esportazione dati missione in pdf:** in seguito del login nella dashboard e dell'apertura della schermata specifica relativa alla missione, dovrà essere possibile, per il medico, esportare in pdf un riepilogo di tutti i dati inseriti. Tale file dovrà contenere:

- **le informazioni relative alla missione:** medico, veicolo di soccorso, codice missione, codice di rientro, ospedale di arrivo e luogo di rilascio.
- **le informazioni relative all'evento:** codice di invio, dinamica, indirizzo, se è un intervento secondario, numero di missioni attive sull'evento, numero di pazienti coinvolti e le note.
- **le informazioni sul tracciamento della posizione del team di soccorso** corredate per ogni step di luogo, data e ora.
- **note in generale della missione.**
- **i dettagli del paziente:** nome, cognome, data di nascita, luogo di nascita, residenza, sesso, anticoagulanti, antipiastrine.
- **i dettagli sullo stato più recente del paziente:** tipo di trauma, presenza casco/cintura, esistenza di un'emorragia esterna in atto, vie aeree non pervie, tachipnea/dispnea, esistenza di un volet costale/deformità costale, risultato ecofast, bacino instabile/deformato, presenza di un'amputazione/subamputazione, esistenza di una frattura cranica, presenza di paralisi e testo del giudizio clinico.
- **l'elenco delle somministrazioni, trattamenti, manovre e complicanze.**
- **l'elenco di tutti i parametri vitali registrati.**

3.2.2 Requisiti non funzionali

Vengono ora presentati i requisiti non funzionali classificandoli in base a se sono **relativi al prodotto**[4], **al processo**[5], o **esterni.**[6]

Relativi al prodotto

- **Usabilità:** l'interazione dell'utente con il sistema e, in particolare, con l'applicazione dovrà essere semplice e immediata.

  L'interfaccia utente del programma applicativo sul tablet dovrà permettere un'interazione rapida e intuitiva, in modo da facilitare i medici impegnati nelle operazioni di soccorso ed evitare loro di creare ulteriori

[4]**Requisiti non funzionali relativi al prodotto:** descrivono le caratteristiche del prodotto, in termini di usabilità, efficienza, affidabilità, portabilità ecc...

[5]**Requisiti non funzionali relativi al processo:** riguardano i metodi utilizzati durante lo sviluppo del software. Sono inclusi quindi tutti i requisiti relativi alla consegna e all'implementazione del progetto.

[6]**Requisiti esterni:** si riferiscono a fattori estranei al sistema ed al relativo processo di sviluppo, come requisiti legislativi e requisiti di interoperabilità.

difficoltà. Specificatamente, i colori e i font utilizzati dovranno favorire la lettura anche in situazioni sfavorevoli (forte luminosità, mancanza di luce..), dovrà essere limitato il più possibile lo scorrimento verticale delle schermate, i pulsanti e le caselle di testo attraverso i quali avviene l'interazione uomo-app dovranno essere di grandi dimensioni per diminuire il più possibile il rischio d'errore.

Il medico dovrà *poter apprendere come utilizzare il sistema (app e dashboard) velocemente e senza difficoltà.* È necessario, inoltre, che alcuni *passaggi all'interno dell'applicazione siano guidati* per garantire la corretta sequenza temporale delle informazioni inserite. Questo può essere realizzato disabilitando momentaneamente alcune funzionalità e usando popup informativi.

- **Correttezza:** il sistema deve essere corretto ovvero deve implementare tutti i requisiti funzionali e non, presenti nella specifica dei requisiti.

- **Portabilità:** l'applicazione e la dashboard dovranno poter funzionare rispettivamente su diversi dispositivi tablet e computer, perciò è necessario che siano *responsive*: le schermate devono essere scrollable e adattare la disposizione degli elementi allo schermo in modo che tutto sia visibile.

- **Efficienza:** l'applicazione dovrà essere efficiente in termine di risorse e spazio utilizzati e veloce nell'esecuzione delle operazioni richieste.

- **Affidabilità:** il sistema dovrà assicurare la correttezza dei dati salvati. In particolare, l'applicazione dovrà garantire anche la persistenza delle informazioni relative alle singole missioni fino a quando non verranno inviate con successo al servizio di back-end.

- **Manutenibilità:** le funzionalità che saranno offerte dal programma dovranno essere implementate per moduli[7] quanto più possibili indipendenti tra loro. In questo modo viene ottimizzata la manutenibilità e l'estendibilità del sistema e sarà ridotta anche la complessità.

- **Robustezza:** il sistema, e in particolare l'applicazione, dovrà comportarsi in maniera ragionevole anche in circostanze non previste come l'assenza di rete o la chiusura improvvisa dell'app. In quest'ultimo caso dovrà essere possibile ripristinare la missione da dove si era interrotta.

[7] **Modulo:** componente di base di un sistema che raccoglie un insieme di funzionalità tra loro strettamente legate.

Esterni

- **Interoperabilità:** è fondamentale che l'applicativo sia capace di cooperare con altri sistemi, tramite servizi RESTful HTTP[8] ad hoc o in altro modo, poiché il progetto PreH nasce proprio come estensione indipendente di Trauma Tracker e la previsione è quello di integrare ulteriori sottosistemi per rendere il tracciamento completo.
- **Sicurezza:** è imprescindibile che il sistema garantisca la privacy dei pazienti e la protezione da eventuali violazioni poiché vengono trattate informazioni molto sensibili. È necessario quindi creare un login robusto per l'accesso ai dati.

[8]**Servizi RESTful:** sono dei servizi che rispettano lo stile, imposto da REST, nella trasmissione di dati in una comunicazione HTTP.

Capitolo 4

Progettazione e Sviluppo

Dopo aver eseguito un'attenta analisi del dominio e dei requisiti, nel seguente capitolo verrà presentata la seconda fase del ciclo di vita del software: **la progettazione.** Precisamente, saranno elencate le varie parti di cui il sistema si compone, le loro architetture e il modo nel quale comunicano.

In seguito, dopo aver esposto le tecnologie e i linguaggi scelti per le varie componenti del progetto, si passerà al successivo step di **sviluppo** nel quale verranno illustrate le scelte implementative intraprese per il miglioramento e l'estensione del prototipo esistente. Per rendere più semplice tale descrizione saranno immesse, a titolo esemplificativo, brevi parti di codice.

4.1 Architettura generale del Sistema

Il sistema PreH sarà suddiviso in tre principali componenti:

- **PreH App:** applicazione destinata ai dispositivi tablet che verrà utilizzata dal leader PreH per registrare tutte le informazioni e le operazioni relative al paziente soccorso, gli spostamenti del team e i dati generali attinenti alla missione.
- **PreH Service:** servizio di back-end centrale del sistema. Si occupa di rispondere alle richieste ricevute dalle altre componenti interagendo in maniera opportuna con il database.
- **PreH Dashboard:** pagina web utilizzata dal personale sanitario in ospedale per visualizzare i dati aggiornati in real-time delle missioni in corso, inseriti dai vari team impegnati negli interventi, e le informazioni riguardanti i soccorsi già conclusi. Tale componente permette anche di eliminare le missioni create erroneamente e di esportare i dati in un file pdf.

In generale ogni sistema software è composto da una parte di Back-end e una di Front-end.

Con il termine **Back-end** ci si riferisce a tutta la porzione del sistema nascosta all'utente che incapsula la logica di funzionamento e che quindi gestisce anche le interazioni tra le varie componenti. Solitamente è composta da uno o più server che forniscono dei servizi alle parti client e, come detto sopra, generalmente è accessibile solamente allo sviluppatore, l'utente finale interagisce indirettamente con essa utilizzando il front-end. Nel nostro specifico caso, il Back-end corrisponde con il *PreH Service.*

Con il termine **Front-end** invece si indicano tutte le parti del sistema visibili all'utente che possono essere usate da quest'ultimo per l'interazione. Tali componenti richiedono e ricevono i servizi erogati dal back-end. In relazione al nostro progetto, appartengono al Front-end *l'applicazione e la dashboard PreH.*

4.1.1 Applicazione PreH

Architettura Il pattern architetturale che segue l'applicazione PreH è **MVVM (ModelView View Model)**, il quale permette di separare lo sviluppo della logica del dominio da quello dell'interfaccia grafica. Questo comporta diversi *benefici*: grazie all'indipendenza delle varie componenti diversi team di sviluppo possono lavorare in contemporanea, è possibile rinnovare completamente l'interfaccia grafica senza andar a toccare le classi del dominio ed anche il testing è molto più semplificato.

Le **componenti principali** del pattern sono:

- **Model:** è l'implementazione del modello del dominio dell'applicazione e rappresenta il punto di accesso ai dati. Solitamente è composto da strutture e classi semplici.
- **ViewModel:** è il punto di incontro tra la View e il Model. I dati ricevuti da quest'ultimo sono elaborati per essere presentati e passati alla View.
- **View:** rappresenta la vista dell'applicazione; si occupa infatti di renderizzare tutti gli elementi grafici che vengono visualizzati dall'utente e di riceverne l'interazione.

La comunicazione tra le varie parti percorre il seguente iter mostrato anche nella Figura 4.1: quando l'utente interagisce con la View essa utilizza il *Data binding* o attiva un metodo tramite un *Command* per inoltrare tale evento al ViewModel che a sua volta lo risolve aggiornando il Model. Appena esso viene modificato, invia una notifica di cambiamento al ViewModel, che fa conseguire per quest'ultimo un aggiornamento del suo stato e/o delle sue proprietà che si riflette anche sugli elementi mostrati dalla View.

Per **Data binding** ci si riferisce al processo che permette di stabilire un'associazione tra i dati contenuti nel Model e quelli presentati nell'interfaccia utente. Tale connessione permette, a fronte di modifiche, di mantenerli sempre sincronizzati. Il **Command** invece è un oggetto, che implementa l'interfaccia ICommand di .NET framework[1], al quale viene associato una funzione nel codice e un elemento nella GUI (Graphical User Interface). Quando l'utente interagisce con quest'ultimo, il Command viene attivato e viene richiamato il metodo corrispondente al quale è collegato.

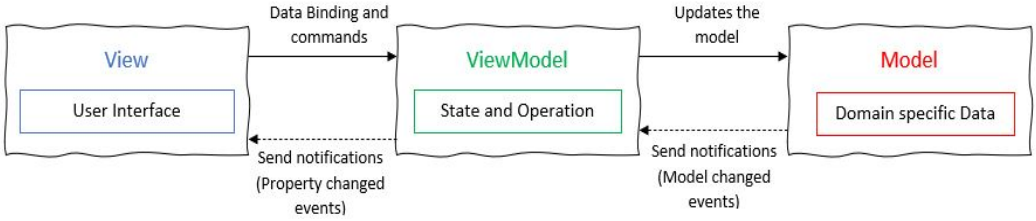


Figura 4.1: Comunicazione pattern MVVM

Entrando maggiormente nello specifico, l'architettura usata nel progetto è quella consigliata da *Android Developers* [2] ed è formata dai seguenti componenti della libreria *Android Jetpack* mostrati in Figura 4.2:

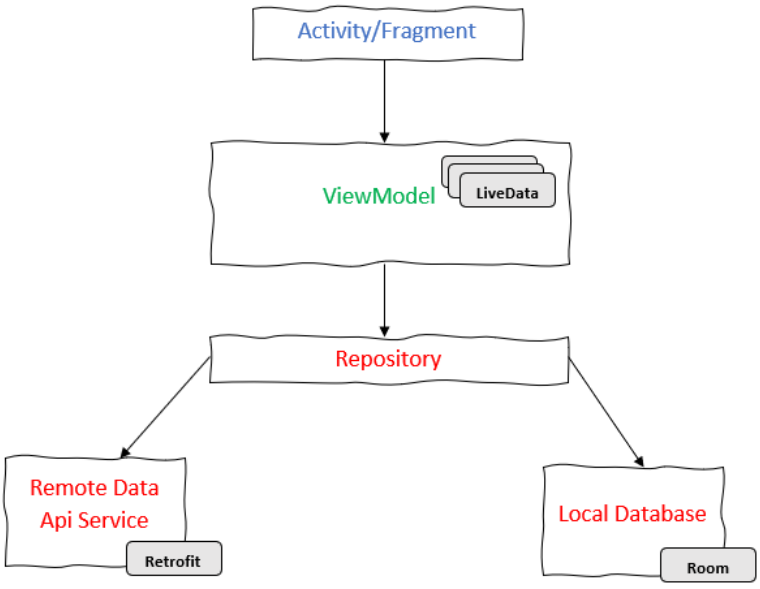


Figura 4.2: Architettura Jetpack MVVM

[1] **.NET Framework:** ambiente di esecuzione, disponibile solo in Windows, che offre un'ampia gamma di servizi alle applicazioni in run-time tra cui un motore di esecuzione e una libreria di classi.

- La View si compone di **Activities e Fragments**.

  Un'*Activity* è essenzialmente una finestra che contiene l'interfaccia grafica di un'applicazione ed il suo scopo è quello di permettere l'interazione con gli utenti.

  I *Fragment* sono delle porzioni di Activity con una propria funzione ed un loro ciclo di vita. Ovviamente essendo "inglobati", quest'ultimo è fortemente collegato con quello dell'Activity di appartenenza: non possono esistere se il loro contenitore non esiste. La convenienza nel loro uso è che apportano vantaggi in termini di performance e che possono anche essere aggiunti o rimossi mentre l'Activity "madre" (ossia quella che li gestisce) è in esecuzione.

  Come si può dedurre mentre ogni schermata può ospitare solo un'Activity in esecuzione per volta, un'Activity può invece contenere più Fragments indipendenti al suo interno.

- I **ViewModel** racchiudono i dati del Model all'interno di istanze di *LiveData:* classi contenitori di dati osservabili. Le View infatti, non accedono direttamente ai dati, ma si registrano come osservatrici dei LiveData in modo da essere notificate in caso di variazioni. Il loro comportamento quindi è molto simile a quello definito dal pattern Observer ma è presente un'importante peculiarità: a differenza di un normale dato osservabile, i LiveData sono *lifecycle-aware*, cioè rispettano il ciclo di vita degli altri componenti dell'app. Questo garantisce che essi notifichino solo gli osservatori che si trovano in uno stato del ciclo di vita attivo.

- Il **Repository** è un modulo che si occupa di gestire tutte le operazioni sui dati interagendo con i specifici componenti sottostanti: *database locale (Room) e servizio per l'interazione con il server remoto (Retrofit).* Egli incapsula il suo comportamento fornendo delle API ai ViewModel per l'accesso alle informazioni.

Come si può vedere dalla Figura 4.2 ogni componente dipende solo da quello sottostante.

4.1.2 PreH Service

Architettura La costruzione del back-end si basa sull'architettura **REST**, ad oggi quasi uno standard per l'implementazione dei servizi web.

Con il termine REST, acronimo di REpresentational State Transfer, ci si riferisce a un insieme di linee guida che definiscono le modalità con cui due dispositivi scambiano messaggi e dati. L'architettura REST si basa su

HTTP: il funzionamento prevede una struttura degli URL[2] ben definita, i quali permettono di identificare univocamente una risorsa o un insieme di risorse, e l'utilizzo dei metodi HTTP specifici per il recupero di informazioni (GET), per la modifica (POST, PUT, PATCH, DELETE) e per altri scopi (OPTIONS, ecc...).

Tale struttura rende il servizio di back-end indipendente dal resto del sistema perché permette di nascondere la sua logica di funzionamento fornendo all'esterno un'interfaccia di **API REST** per l'interazione. In tal modo si lascia anche aperta la possibilità di future evoluzioni del sistema, ad esempio con la modifica o l'aggiunta di altri servizi; in questo caso, infatti, basterà semplicemente definire nuovi URL.

Di seguito vengono elencate tutte le API ripartite in base all'elemento radice del loro percorso: *missione, evento, paziente e login.* Rispetto alla versione precedente del back-end, ne sono state modificate e aggiunte alcune per adeguarle alle nuove funzionalità richieste.

Mission API

Metodo	Path	Descrizione della richiesta
GET	/missions	restituisce tutti i dati delle missioni presenti nel database in formato JSON.
POST	/missions/{missionId}	ricerca nel database la missione con l'id specificato e se presente, la sovrascrive aggiornandola con i dati JSON passati in input nel corpo della richiesta, altrimenti crea una nuova missione restituendo l'identificativo come stringa.
GET	/missions/{missionId}	ricerca nel database e restituisce, se presente, la missione con l'id specificato come parametro.
PUT	/missions/{missionId}	aggiorna la missione che ha l'id indicato con i dati JSON passati in input nel corpo della richiesta.

[2]**URL:** Uniform Resource Locator.

DELETE	/missions/{missionId}	elimina la missione che ha l'id specificato nel path.

Tabella 4.1: Elenco delle Mission API

Qui di seguito nel Listato 4.1, vengono indicati i parametri in formato JSON che descrivono la missione. Tra di essi il *missionId* serve per identificare univocamente il documento nella collezione del database mentre *eventId* è l'identificatore dell'evento che ha generato tale missione. Come si può notare, i campi *returnInformation* e *tracking* sono dei vettori i quali possono essere ricavati seguendo la prima e la quarta rotta della Tabella 4.2.

```
{
   "missionId" : "string",
   "missionCode" : "string",
   "eventId" : "string",
   "vehicle" : "string",
   "onGoing" : "bool",
   "medic" : "string",
   "missionDatetime" : "string",
   "returnInformation" : [],
   "tracking" : [],
}
```

Listato 4.1: JSON missione

A partire da /mission/{missionId}, infatti, è possibile dettagliare ulteriormente il path per operare sulle informazioni di fine missione e di tracciamento del team:

Metodo	**Path**	**Descrizione della richiesta**
GET	/missions/{missionId} /return-information	restituisce le informazioni di fine missione relative al soccorso a cui corrisponde l'id specificato.
PATCH	/missions/{missionId} /return-information	aggiorna le informazioni di fine missione, contenute nel body della richiesta, per il soccorso desiderato.

DELETE	/missions/{missionId} /return-information	elimina le informazioni di fine missione per il soccorso a cui corrisponde l'id specificato.
GET	/missions/{missionId} /tracking	restituisce le informazioni relative a tutti gli step di tracciamento del team della missione specificata.
PATCH	/missions/{missionId} /tracking	aggiorna tutti gli step del tracciamento del team della specifica missione con le informazioni JSON contenute nel body della richiesta.
GET	/missions/{missionId} /tracking/ {trackingStep}	restituisce le informazioni relative ad un preciso step di tracciamento del team, specificato nel path, della missione desiderata.
PATCH	/missions/{missionId} /tracking/ {trackingStep}	aggiorna le informazioni relative ad un preciso step di tracciamento del team, specificato nel path, con quelle passate nel corpo della richiesta.
DELETE	/missions/{missionId} /tracking/ {trackingStep}	elimina le informazioni relative allo step di tracciamento del team indicato nel path della missione specificata.

Tabella 4.2: Elenco delle Mission API - return-information e team tracking

Event API

Metodo	Path	Descrizione della richiesta
GET	/events	restituisce tutti i dati degli eventi presenti nel database in formato JSON.

POST	/events	inserisce un nuovo evento nel database e restituisce il suo identificativo come stringa.
GET	/events/{eventId}	ricerca nel database e restituisce, se presente, l'evento che ha l'id specificato come parametro.
PUT	/events/{eventId}	aggiorna l'evento specificato con i dati JSON passati in input nel corpo della richiesta.
DELETE	/events/{eventId}	elimina l'evento che ha l'id specificato nel path.

Tabella 4.3: Elenco delle Event API

Qui di seguito nel Listato 4.2, vengono indicati i parametri in formato JSON che descrivono l'evento. Tra di essi *l'eventId* serve per identificare univocamente il documento nel database.

```
{
    "eventId" : "string",
    "onGoing" : "bool",
    "address" : "string",
    "dispatchCode" : "string",
    "dynamic" : "string",
    "secondary" : "bool",
    "notes" : "string",
    "callTime" : "string",
    "patientsNumber" : "int",
    "activeMissions" : "int",
}
```

Listato 4.2: JSON evento

Patient API

Per evitare ripetizioni, di seguito verranno descritte solo alcune rotte, la logica di progettazione per le altre è comune a quelle indicate precedentemente.

Metodo	Path	Descrizione della richiesta
GET	/patients	restituisce tutti i dati dei pazienti presenti nel database in formato JSON.
POST	/patients	inserisce un nuovo paziente nel database e restituisce il suo identificativo come stringa.
GET	/patients/{patientId}	ricerca nel database e restituisce, se presente, il paziente con l'id specificato come parametro.
PUT	/patients/{patientId}	aggiorna il paziente specificato con i dati JSON passati in input nel corpo della richiesta.
DELETE	/patients/{patientId}	elimina il paziente che ha l'id specificato nel path.
GET	/patients/{patientId} /status	restituisce lo stato del paziente specificato.
PATCH	/patients/{patientId} /status	aggiorna lo stato del paziente specificato con i dati JSON passati.
GET	/patients/{patientId} /anagraphics	restituisce l'anagrafica del paziente specificato.
PATCH	/patients/{patientId} /anagraphics	aggiorna l'anagrafica del paziente specificato con i dati JSON passati.
GET	/patients/{patientId} /notes	restituisce le note relative al paziente specificato.
PATCH	/patients/{patientId} /notes	aggiorna le note relative al paziente specificato con i dati JSON passati.

Tabella 4.4: Elenco delle Patient API

Qui di seguito nel Listato 4.3, vengono indicati i parametri in formato JSON che descrivono il paziente. Tra di essi il *missionId* indica la missione alla quale il soggetto soccorso è abbinato, invece, il *patientId* oltre a identificare univocamente il documento JSON nel database, permette di dettagliare maggiormente la rotta e dare accesso alle somministrazioni, alle complicanze, alle manovre, ai trattamenti e ai parametri vitali relativi a quel paziente specifico.

```
{
    "patientId" : "string",
    "missionId" : "string",
    "notes" : "string",
    "anagraphic" : [
        "anticoagulants" : "bool",
        "antiplatelets" : "bool",
        "birthday" : "string",
        "birthplace" : "string",
        "firstName" : "string",
        "gender" : "string",
        "lastName" : "string",
        "residence" : "string",
    ],
    "status" : [
        "amputation" : "bool",
        "clinicalJudgment" : "string",
        "deformedPelvis" : "bool",
        "ecofast" : "string",
        "externalHaemorrhage" : "bool",
        "helmetSeatbelt" : "bool",
        "paralysisType" : "string",
        "registrationTime" : "string",
        "respiratoryTract" : "bool",
        "skullFracture" : "string",
        "tachypneaDyspnoea" : "bool",
        "thoraxDeformities" : "bool",
        "traumaType" : "string",
    ]
}
```

Listato 4.3: JSON paziente

Le API per le somministrazioni seguono la progettazione logica delle altre elencate precedentemente, le uniche differenze sono per il metodo *DELETE* che in questo caso elimina tutte le somministrazioni relative al paziente e per *PUT* che le aggiorna tutte.

Metodo	Path
GET	/patients/{patientId}/administrations
POST	/patients/{patientId}/administrations
DELETE	/patients/{patientId}/administrations
PUT	/patients/{patientId}/administrations

Tabella 4.5: Elenco delle Patient API - somministrazioni

Le API delle complicazioni non si discostano nel funzionamento da quelle della Tabella 4.5 ma prevedono due tipologie di *DELETE*: la prima in elenco permette di eliminare tutte le complicazioni relative al paziente, la seconda, invece, consente di rimuoverne una specifica selezionata in base al tipo.

Per ciascuna missione infatti ogni varietà di complicazione si può verificare solo una volta. Le tipologie sono: *cardiocirculatory shock, deteriorating state consciousness, anisocoria mydriasis, respiratory failure, landing in itinere, death in itinere* e *death in PS*.

Metodo	Path
GET	/patients/{patientId}/complications
POST	/patients/{patientId}/complications
DELETE	/patients/{patientId}/complications
DELETE	/patients/{patientId}/complications/{complicationType}
PUT	/patients/{patientId}/complications

Tabella 4.6: Elenco delle Patient API - complicazioni

Relativamente alle API delle manovre, tramite la prima rotta della Tabella 4.7 si ottengono tutte quelle eseguite su un certo paziente in due liste distinte, differenziate rispetto al tipo che può essere *simple o pacing*. Se invece si precisa anche il *procedureType* e si segue quindi la quarta rotta, si ottiene uno solo dei due elenchi precedenti.

Poiché durante la missione il medico può eseguire più manovre di tipo pa-

cing, queste vengono contraddistinte tramite un *procedureId* attraverso il quale è possibile ottenere ed eliminare la specifica manovra. Diversamente, quelle simple possono essere eseguite solo una volta e per questo vengono identificate con il loro nome: *cervical collar, immobilization, electrical cardioversion, gastric probe, bladder catheter.*

Le rimanenti rotte seguono la semantica di quelle contenute nella Tabella 4.5 delle somministrazioni.

Metodo	**Path**
GET	/patients/{patientId}/procedures
DELETE	/patients/{patientId}/procedures
PUT	/patients/{patientId}/procedures
GET	/patients/{patientId}/procedures/{procedureType}
GET	/patients/{patientId}/procedures/simple/{procedureName}
POST	/patients/{patientId}/procedures/simple/{procedureName}
DELETE	/patients/{patientId}/procedures/simple/{procedureName}
GET	/patients/{patientId}/procedures/pacing/{procedureId}
POST	/patients/{patientId}/procedures/pacing
DELETE	/patients/{patientId}/procedures/pacing/{procedureId}

Tabella 4.7: Elenco delle Patient API - manovre

Similmente alle manovre, anche i trattamenti si distinguono in varie tipologie: *simple, timed, injection, ippv.* Specificando il tipo è possibile accedere o creare specifici trattamenti seguendo rispettivamente la quarta e quinta rotta della Tabella 4.8.

Poiché il paziente può ricevere lo stesso trattamento in momenti diversi, è necessario introdurre il *treatmentId* per poter identificare ciascuna esecuzione. Grazie ad esso sarà anche possibile reperirla o cancellarla successivamente seguendo le ultime due operazioni indicate sempre nella tabella sottostante.

Anche in questo caso, le altre rotte non descritte seguono la progettazione logica della Tabella 4.5 delle somministrazioni.

Metodo	Path
GET	/patients/{patientId}/treatments
PUT	/patients/{patientId}/treatments
DELETE	/patients/{patientId}/treatments
GET	/patients/{patientId}/treatments/{treatmentType}
POST	/patients/{patientId}/treatments/{treatmentType}
GET	/patients/{patientId}/treatments/{treatmentType}/ {treatmentId}
DELETE	/patients/{patientId}/treatments/{treatmentType}/ {treatmentId}

Tabella 4.8: Elenco delle Patient API - trattamenti

Metodo	Path
GET	/patients/{patientId}/vitalsigns
POST	/patients/{patientId}/vitalsigns
DELETE	/patients/{patientId}/vitalsigns
PUT	/patients/{patientId}/vitalsigns

Tabella 4.9: Elenco delle Patient API - parametri vitali

Login API

Poiché i requisiti richiedono che l'accesso ai dati della dashboard sia esclusivo per gli utenti autorizzati è necessario prevedere delle API che consentano di verificare le credenziali per il login.

Metodo	Path	Descrizione della richiesta
POST	/ login/	verifica che l'id utente e la password contenuti nel body della richiesta corrispondano a uno degli utenti registrati nel database. In caso affermativo restituisce la stringa "autorizzato", altrimenti "non autorizzato".

Tabella 4.10: Elenco delle Login API

Memorizzazione delle informazioni

Il modo in cui memorizzare i dati raccolti dal sistema è un punto centrale della progettazione. In particolare, occorre effettuare una scelta del tipo di DBMS[3] che si intende utilizzare: affidarsi ai tradizionali database MySql relazionali oppure a quelli NoSql, detti anche non relazionali?

È necessario prima di tutto evidenziarne le differenze. I **database relazionali** racchiudono i dati all'interno di *tabelle* che devono essere obbligatoriamente generate al principio perché supportano solo un formato di dati fisso. Esse sono collegate alle altre tramite delle relazioni. I **DB NoSql** invece, raccolgono le informazioni in *documents* contenuti a loro volta in *collections*. Effettuando un riferimento con i DB relazionali, le collections sono delle raccolte dati simili alle tabelle, mentre i documents corrispondono alle righe che le vanno a comporre. Ogni document però può avere una struttura diversa e pertanto possedere o meno, lo stesso numero e lo stesso tipo di campi. Riassumendo quindi si può dire che MySql lavora con uno *schema fisso*, mentre NoSql con uno *schema flessibile*.

I benefici nell'uso di un database non relazionale sono molteplici. Innanzitutto *la ricerca è più performante* perché non sono presenti aggregazioni tra i dati e quindi ci si aspetta di trovarli tutti all'interno di uno stesso document. Questo fatto comporta anche che non vi siano *rischi per l'integrità*, infatti, dato che non sono "spezzettati" in più posti, possono essere recuperati tutti in una volta. Infine, poiché la *scalabilità delle informazioni avviene in maniera orizzontale* e non verticale, è possibile gestire in maniera agevole un aumento consistente dei dati da memorizzare.

Dal momento che abbiamo la necessità di usare metodi flessibili di memorizzazione che possono evolvere agilmente nel tempo per adattarsi ad eventuali

[3]**DBMS:** DataBase Management System.

variazioni del dominio e anche per gli altri vantaggi appena elencati, la scelta ricade sull'uso di un database NoSql.

4.1.3 Dashboard

Architettura

Si è scelto di sviluppare la dashboard con l'architettura di una **single-page-Application (SPA)** cioè come un sito web che può essere usato o consultato attraverso una singola pagina web con l'obiettivo di fornire *un'esperienza d'uso più fluida per l'utente*. Non solo, anche la *velocità* del sito viene *fortemente incrementata* grazie al fatto che le SPA caricano tutte le sezioni e le risorse della single-page solo nella fase iniziale, successivamente l'interazione con il server avviene solo per richiedere esclusivamente il singolo contenuto da aggiornare. Quindi, dopo la fase iniziale, non si ricaricherà mai interamente la pagina, fatto che invece accade in un applicativo web standard.

Dal dominio si comprende che la principale funzionalità della dashboard è quella di mostrare tutti i dati relativi ad ogni missione, sia in corso che terminata. In particolare per le missioni attive è necessario che appena i dati vengono inseriti nel database siano mostrati aggiornati anche nella dashboard in real-time. Per tale motivo è necessario adottare un approccio di **polling** nell'effettuare le richieste al server in modo da avere sempre le informazioni più recenti.

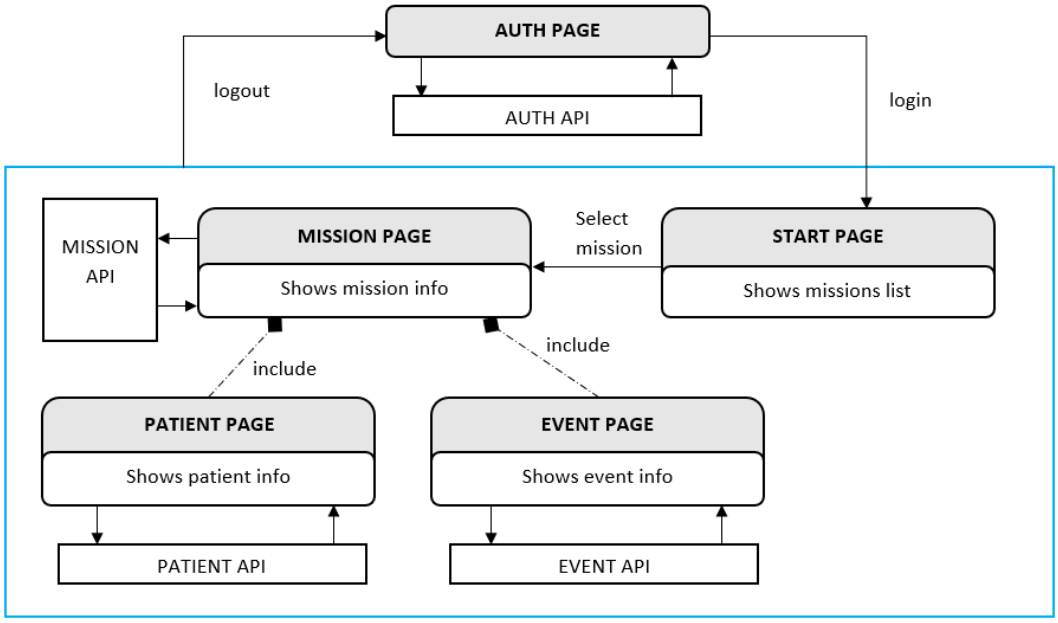


Figura 4.3: Componenti principali Dashboard

Le Figura 4.3 riassume le componenti principali della dashboard. Come si può notare anche in questo caso, come per il server e l'app, le entità sulle quali si fonda l'applicativo web sono sempre *la missione, l'evento e il paziente.*

Appena si raggiunge il sito della dashboard, il primo contenuto mostrato nella single-page è quello della **pagina di autenticazione** che permette, utilizzando le *Auth API* del back-end, di discriminare gli utenti che hanno l'autorizzazione di accesso ai dati rispetto a quelli che non la hanno.

Una volta che il login è andato a buon fine si accede alla **pagina iniziale** che contiene l'elenco delle missioni in corso e terminate in due tabelle distinte. Per ciascuna di esse sarà possibile decidere se eliminarla o se visualizzarne maggiori dettagli. Nel caso si scelga quest'ultima opzione, il contenuto della pagina sarà sostituito dinamicamente con quello relativo **alla schermata della specifica missione** che conterrà tutti i dati relativi al soccorso selezionato, ottenuti grazie all'interazione con le *Mission API* del back-end, e le informazioni più recenti relative al paziente e all'evento ad esso associati, richieste al server tramite rispettivamente *le Patient API e Event API.*

4.1.4 Architettura di comunicazione

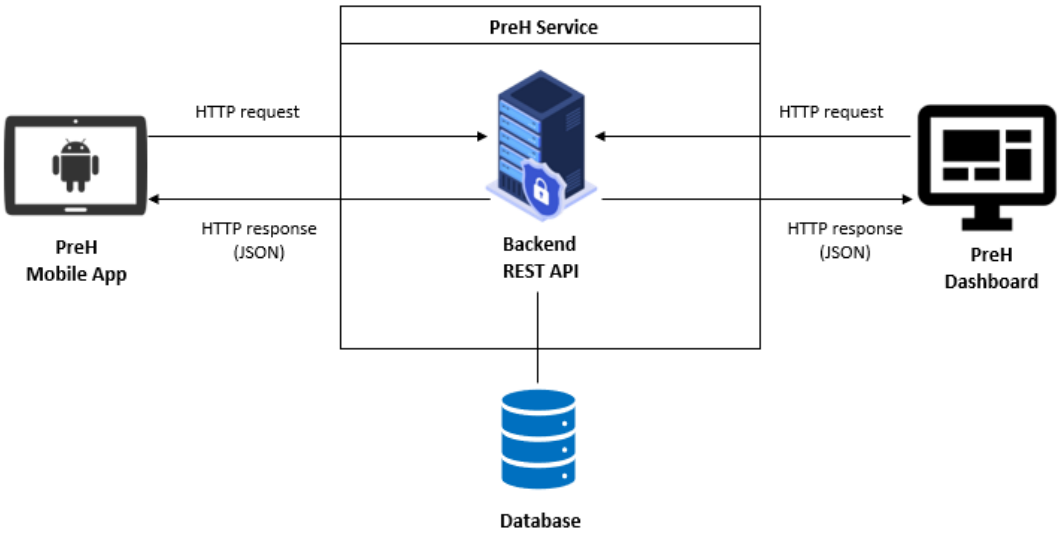


Figura 4.4: Architettura di comunicazione

L'architettura di comunicazione, raffigurata in Figura 4.4, si basa sul modello **client-server** in cui il Back-end rappresenta l'unico server di riferimento e la Mobile App e la Dashboard i due client.

Il back-end è l'unico che interagisce con il Database NoSql crittografato e, quando richiesto, permette l'accesso *indirettamente* ai documents memorizzati,

ai due client tramite le REST API. Grazie all'unicità del server si ha una **centralizzazione dei dati** che comporta un vantaggio importante: in questo modo si è sicuri che le informazioni propagate ai client non siano inconsistenti (cioè non aggiornate).

Per poter interagire con il PreH Service, l'app e la dashboard utilizzano la rete Internet (Wi-Fi o reti mobili 3G,4G,5G) e servendosi del **protocollo HTTP** effettuano delle richieste all'interfaccia delle REST API messe a disposizione dal back-end. I metodi HTTP utilizzati per l'accesso ai dati possono essere:

- **GET** che viene usato per richiedere un elemento presente nel database.
- **POST** che viene generalmente utilizzato per la creazione di una nuova risorsa nel database.
- **PUT** che viene usato per effettuare un aggiornamento totale di un elemento nel database.
- **PATCH** che viene utilizzato per effettuare un aggiornamento parziale di una risorsa nel database.
- **DELETE** che viene usato per cancellare un elemento nel database.

Tutte le informazioni inviate sia dai client che dal server sono rappresentate mediante il formato **JSON**.

4.2 Tecnologie e Linguaggi

Di seguito verranno descritte brevemente le tecnologie e i linguaggi utilizzati nelle varie componenti del sistema.

4.2.1 Applicazione PreH

L'applicazione PreH è un'*app Android* scritta con il linguaggio di programmazione *Kotlin*.

Kotlin è un linguaggio *general purpose* ed *open-source*, sviluppato dall'azienda JetBrains. È orientato verso la programmazione ad oggetti ma permette anche un pieno uso dell'approccio funzionale[4]. Si è scelto Kotlin rispetto a Java per diversi motivi:

[4]**Programmazione funzionale:** un paradigma di programmazione in cui il flusso di esecuzione del programma si basa sulla valutazione di espressioni e funzioni con l'obiettivo di ridurle ad un valore più sintetico possibile.

- Innanzitutto dal 7 maggio 2019 è il **linguaggio consigliato da Google** per la creazione di app Android ed è stato anche adottato e integrato nell'ambiente di sviluppo **Android Studio** a partire dalla versione 3.0. Per tale motivo si è scelto tale IDE per la produzione del codice dell'app.

- Le applicazioni sviluppate in Kotlin sono **molto portabili** infatti possono essere eseguite su qualsiasi ambiente che accetti la *Java Virtual Machine* (JVM). Inoltre, il compilatore è in grado di produrre anche codice *Javascript*.

- Kotlin è un linguaggio di programmazione **più coinciso e rapido** rispetto a Java, questo comporta una minore probabilità di bug ed accelerazione nei tempi di sviluppo.

- Kotlin è profondamente **interoperabile** con Java, tanto che è possibile convertire parti di codice nell'IDE da un linguaggio all'altro. Questa caratteristica è importante perché significa che si possono riusare importanti librerie Java e per chi conosce già quest'ultimo linguaggio l'apprendimento di Kotlin sarà facilitato.

- Kotlin ha eliminato l'errore più frequente che si riscontra in Java cioè il *NullPointerException* che avviene quando si cerca di assegnare un riferimento a una variabile o a un oggetto che ha valore nullo. Infatti, su Kotlin, occorre impostare esplicitamente che una variabile possa avere tale valore perché normalmente non può. Inoltre, in caso di uso di un **Nullable type**, il **safe call operator (?.)** garantisce che si acceda ad esso solo nel caso sia valido (cioè abbia valore diverso da *null*).

4.2.2 PreH Service

Vert.x e Java Vert.x è un *toolkit*[5] di sviluppo software *open source, event-driven*[6] il quale permette di sviluppare applicazioni reattive.

Un'applicazione Vert.x consiste di uno o più componenti chiamati **Verticle** ognuno specializzato nello svolgere uno specifico compito. Tali parti vengono eseguite ciascuna in maniera concorrente rispetto alle altre e comunicano tra loro scambiandosi eventi che vengono inviati appunto sul *Event bus*. Si riesce così a creare applicazioni multi-threaded senza dover gestire problematiche di concorrenza come la sincronizzazione o i lock tra i thread.

[5]**Toolkit:** insieme di strumenti software di base, in genere librerie, che vengono usati per facilitare lo sviluppo di applicazioni complesse.

[6]**Programmazione event-driven:** paradigma di programmazione in cui il flusso d'esecuzione del programma è determinato dall'avvenire di eventi quali azioni dell'utente, output di sensori o arrivo di messaggi, che vengono poi gestiti tramite specifiche funzioni di callback.

Una delle caratteristiche fondamentali di Vert.x è che è **poliglotta** cioè supporta più linguaggi JVM e non JVM.

Per lo sviluppo della parte di **back-end** si è scelto quindi di utilizzare questa tecnologia insieme a **Java** che è un linguaggio di programmazione *orientato agli oggetti* altamente prestante e affidabile. La sua prima caratteristica fondamentale è che non *lascia l'onere al programmatore di gestire le allocazioni/deallocazioni della memoria* (non prevede quindi i puntatori). È presente infatti un **garbage collector** che si occupa di assegnarla e rilasciarla a seconda delle esigenze del programma. La seconda peculiarità è che è *indipendente dalla piattaforma hardware di esecuzione* grazie al fatto che prima il codice viene compilato in un **linguaggio binario intermedio (bytecode)** e poi viene interpretato dalla **Java Virtual Machine (JVM).** Questo ci fa capire che è ampiamente supportato ed infatti uno dei suoi motti fondamentali è **WORA**: write once, run anywhere, ossia "scrivi una volta, esegui ovunque").

Entrando nel dettaglio, due strumenti in particolare, messi a disposizione da Vert.x, sono stati fondamentali per la programmazione del componente di back-end:

- **Vert.x-Web:** strumento che mette a disposizione gli elementi base per la costruzione di applicazioni Web. Nel nostro caso è stato utilizzato per creare *l'applicativo Web RESTful lato server* che gestisce le richieste HTTP ricevute dalla dashboard e dalla mobile app.

- **Vert.x MongoDB Client:** client che permette di interagire con un'istanza del DBMS MongoDB per recuperare, memorizzare, eliminare e modificare documenti nel database.

MongoDB Tra le varie tipologie di DBMS non relazionali si è scelto MongoDB che si caratterizza per il fatto di essere *open-source* e *basato su documenti* di formato *BSON* (Binary JSON), una variante di JSON.

Ogni documento è incluso in una specifica *collezione* ma non necessariamente la sua struttura segue un particolare schema a seconda di dove è contenuto. Oltre a ciò i dati racchiusi al suo interno possono essere eterogenei (non necessariamente tutti dello stesso tipo) e vengono rappresentati come delle coppie nome-valore.

È stato scelto per i seguenti motivi:

- I dati memorizzati in formato BSON possono essere *indicizzati*, aumentando così notevolmente le prestazioni nella ricerca. Inoltre, dato che JSON è supportato dalla maggior parte dei linguaggi di programmazione ed è uno standard de-facto per le comunicazioni client-server nei

contesti RESTful, si *minimizzano i problemi di trasmissione dei dati* tra il database e le varie componenti del sistema.

- *permette facilmente di modificare la struttura delle collezioni* del database a fronte di variazioni del modello dei dati del progetto.

- offre un'*alta scalabilità* infatti è in grado di supportare enormi volumi di dati e traffico. Ciò consente di rispondere adeguatamente all'eventuale successiva evoluzione del sistema.

4.2.3 Dashboard

Html, Css e Bootstrap Per la definizione della struttura e dello stile dei vari contenuti della pagina web di cui si compone la dashboard, sono stati utilizzati diversi linguaggi:

- **HTML (HyperText Markup Language):** è un linguaggio di *markup* nato per dare formato ai documenti ipertestuali. La struttura della pagina viene definita tramite degli elementi di markup chiamati *tag*: stringhe contenute tra <>.

- **CSS (Cascading Style Sheets):** è un linguaggio che va a definire lo stile della pagina web. In questo modo viene separata la presentazione (CSS) dal contenuto (HTML) della pagina.

Per evitare di scrivere completamente codice Html e Css a mano ci si è affidati a **Bootstrap:** un *web-framework*[7] *gratuito e open-source* per il design dei siti e delle applicazioni web. Esso mette a disposizione dei template basilari Html e Css pronti per essere inseriti nella pagina che sono anche *accessibili e responsive*.

Javascript e Ajax Per la cattura e la gestione degli eventi, derivanti dall'interazione dell'utente con il contenuto della single-page della dashboard, si è utilizzato **Javascript** sfruttando anche la sua popolare libreria **jQuery**. Javascript è un *linguaggio di scripting*[8] *orientato agli oggetti e agli eventi* che viene comunemente utilizzato nella programmazione web lato client per la creazione di siti web dinamici.

Invece, per implementare il meccanismo di polling verso il server è stata utilizzata come tecnologia **AJAX**, acronimo di Asynchronous Javascript And

[7] **Web-Framework:** architettura logica di supporto che ha l'obiettivo di facilitare l'uso di una tecnologia nello sviluppo web per il programmatore.

[8] Per dettagli `https://it.wikipedia.org/wiki/Linguaggio_di_scripting`

Xml. Essa permette di creare applicazioni web client-side e server-side fortemente interattive e dinamiche grazie ad uno scambio di dati in background tra browser e server che consente l'aggiornamento della pagina senza esplicito ricaricamento da parte dell'utente.

4.3 Sviluppo Applicazione PreH

4.3.1 Salvataggio missioni sul dispositivo

Il primo aspetto che si è migliorato dell'applicazione già esistente è che essa consentiva di gestire solo una missione per volta: finché questa non era terminata e inviata con successo al servizio di back-end non era possibile iniziarne una nuova senza perdere i dati di quella in sospeso, poiché venivano sovrascritti.

L'invio al server dei dati della missione si basa principalmente sulla disponibilità di rete che può essere, per sua natura, molto instabile. Durante il trasporto in elisoccorso, inoltre, è necessario che la connessione dati e il Wi-Fi siano spenti. Un altro rischio possibile è che il back-end stesso non risponda o tardi a rispondere. Viste tutte queste situazioni, è facile quindi immaginare che spesso si hanno delle difficoltà nell'invio dei dati al server.

Per ovviare a questo problema si è reso il funzionamento dell'applicazione il più possibile indipendente dalla connessione: è stato permesso *il salvataggio sul dispositivo delle missioni momentaneamente non inviabili* e si è evitato che *l'avvio di un nuovo soccorso sovrascrivesse i dati memorizzati in precedenza* sul tablet.

Memorizzazione in locale La memorizzazione dei dati sul dispositivo è stata effettuata utilizzando la libreria **Room** di Android che prevede tre componenti principali:

- Una **classe Database** che contiene il DB ed è il punto di accesso principale ai dati persistenti dell'applicazione.
- Delle **entità dei dati** che rappresentano le tabelle del database.
- Degli **oggetti di accesso ai dati (DAO)** che forniscono metodi per la ricerca, inserimento, modifica, eliminazione delle informazioni sul database.

Nel seguente Listato 4.4 viene mostrata l'implementazione della classe Database per il nostro progetto che è denominata *PrehRoomDatabase*. Come si

può notare inizialmente vengono elencate le varie *entità dei dati* di cui il database si compone, mentre all'interno della classe astratta si forniscono le *istanze dei DAO* che nel nostro caso corrispondono alle tre macro parti di base del sistema: *PatientDao, MissionDao e EventDao.*

```
@Database(
    entities = [
        PatientData::class,
        MissionData::class,
        EventData::class,
        VitalSignsData::class,
        SimpleProcedureData::class,
        PacingProcedureData::class,
        AdministrationData::class,
        ComplicationData::class,
        SimpleTreatmentData::class,
        IppvTreatmentData::class,
        TimedTreatmentData::class,
        InjectionTreatmentData::class,
        CriterionData::class
    ],
    version = 1,
    exportSchema = false
)
abstract class PrehRoomDatabase : RoomDatabase() {
    abstract fun patientDao(): PatientDao
    abstract fun missionDao(): MissionDao
    abstract fun eventDao(): EventDao
}
```

Listato 4.4: PrehRoomDatabase.kt

Per evitare la sovrascrittura dei dati delle missioni già salvate, per ciascuna di esse si è mantenuto un *identificativo numerico univoco* che si incrementa ogni volta che un nuovo soccorso viene attivato. L'id va anche a contraddistinguere per ogni missione, tutte le informazioni che sono associate ad essa e che sono divise nelle varie tabelle del database.

Terminazione della missione Quando il medico preme sul pulsante di terminazione missione è stata aggiunta la possibilità, tramite un popup [Figura 4.5], di effettuare una scelta tra tre opzioni:

- **Invia missione:** si tenterà l'invio al servizio di back-end della missione. Se va a buon fine, i dati locali diventano superflui e quindi per minimiz-

zare l'occupazione di memoria verranno eliminati, al contrario, se non ha successo, tutte le informazioni verranno automaticamente salvate sul dispositivo. In tal modo, si evita di bloccare la possibilità per l'utente di creare una nuova missione fino a quando la trasmissione di quella corrente non viene completata con successo.

- **Salva missione:** la missione verrà salvata sul dispositivo. Viene data la possibilità all'utente di decidere autonomamente se memorizzare direttamente i dati del soccorso in locale, senza tentare l'invio. Questa opzione sarà utilizzata solitamente quando il medico si accorge della mancanza di rete.

- **Elimina missione:** tutti i dati sul dispositivo relativi alla missione verranno eliminati. In generale l'eliminazione è utilizzata soprattutto nel primo periodo dopo il rilascio del sistema, poiché alcune missioni vengono avviate come test dai medici per capire il funzionamento dell'applicativo. Tale opzione può comunque tornare utile nel caso in cui il soccorso avviato deve essere annullato per altri motivi.

Figura 4.5: Popup termina missione

4.3.2 Fragment missioni salvate

Poiché si è espressa la necessità di salvare le missioni in locale, deve essere possibile offrire all'utente anche un modo di gestirle. Si è quindi deciso di creare, seguendo il *pattern MVVM*, una nuova schermata che contenesse un elenco di tutte le missioni memorizzate sul dispositivo, e per ciascuna di esse è stata data la possibilità di ritentarne la trasmissione o di eliminarla utilizzando due specifici pulsanti.

Implementazione Pattern MVVM Come mostrato nel Listato 4.5 è stato creato un apposito Fragment chiamato *LocalSavedMissionsFragment* al quale è stato associato uno specifico ViewModel. Quest'ultimo espone un elenco di oggetti *Mission*, appartenenti al Model, tramite un oggetto LiveData chiamato

liveMissionsList al quale il Fragment si registra come osservatore. In tal modo, ogni volta che la lista si modifica, esso viene notificato e quindi si occupa di aggiornarla anche a video.

```
class LocalSavedMissionsFragment : Fragment() {
    /* binds a ViewModel in one line using delegates */
    private val viewModel: LocalSavedMissionsViewModel by
        viewModels()

    override fun onViewCreated(view: View, savedInstanceState:
        Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        viewModel.liveMissionsList.observe(viewLifecycleOwner,
            Observer {
                elements -> listAdapter.setMissions(elements)
        })
    }

    override fun onResume() {
        super.onResume()
        viewModel.setLocalSavedMissionsDataList()
    }
}
```

Listato 4.5: LocalSavedMissionsFragment.kt

Per poter ottenere l'elenco delle missioni memorizzate nel database, come si può vedere nel Listato 4.6, nella fase di inizializzazione del *LocalSavedMissionsViewModel* si ottiene un'istanza del *MissionDao* con la quale viene creato un *MissionRepository*.

Ogni volta che il Fragment si trova nello stato di Resumed (cioè quando viene aperto o ripristinato) chiama la funzione *setLocalSavedMissionsDataList()* implementata nel ViewModel che, utilizzando il repository accede al database e recupera una lista di MissionData. Essa rappresenta tutte le missioni salvate sul dispositivo e viene settata in un MutableLiveData. Però ogni MissionData contiene oltre ai dati di una missione, anche un identificativo univoco per differenziarsi dagli altri; dato che al Fragment non interessa quest'ultimo, tali informazioni vengono mappate utilizzando la funzione **switchMap**: ogni volta che la lista di MissionData varia, tale metodo viene chiamato e viene eseguito il codice al suo interno. In questo caso, per ciascun elemento della lista vengono estrapolati solo i dati relativi alla missione e inseriti all'interno del LiveData *liveMissionList* che viene poi esposto al Fragment.

L'estensione **liveData(IO)** consente di definire una porzione di codice che

deve essere eseguita, quando possibile, in background da una Coroutine. Specificando un valore dentro al metodo **emit()**, esso viene avvolto da un oggetto LiveData ed assegnato alla variabile liveMissionsList.

```
class LocalSavedMissionsViewModel(application: Application) :
    AndroidViewModel(application) {
    //allow to handle local saved data
    private val missionRepository: MissionRepository
    private val liveMissionsDataList:
        MutableLiveData<List<MissionData>> = MutableLiveData()

    val liveMissionsList = switchMap(liveMissionsDataList) {
        liveData(IO) {
            emit(it.map { it.mission })
        }
    }

    init {
        val missionDao =
            PrehRoomDatabase.getDatabase(application).missionDao()
        missionRepository = MissionRepository(missionDao)
    }

    fun setLocalSavedMissionsDataList() = viewModelScope.launch(IO) {
        liveMissionsDataList.postValue(missionRepository.
        getLocalSavedMissionsData())
    }
}
```

Listato 4.6: LocalSavedMissionsViewModel.kt

Sempre nel Listato 4.6 si può vedere che ogni volta che si effettua l'accesso al database viene utilizzato un thread diverso da quello principale (*viewModelScope.launch(IO){...}*). La motivazione è che l'accesso e la lettura dei dati nel DB possono richiedere molto tempo e quindi si rischia di bloccare il thread principale che occupandosi della gestione della grafica farebbe comportare dei rallentamenti consistenti nell'interfaccia utente.

Implementazione lista Per l'implementazione della lista delle missioni salvate sul dispositivo si è deciso di utilizzare il componente **RecyclerView** messo a disposizione da *Android Jetpack*, poiché rende semplice ed efficiente la visualizzazione di grandi set di dati. Infatti, basta definire l'elenco di elementi da visualizzare ed il loro aspetto, poi è tale libreria che si occupa di creare dinamicamente ciascuno di essi quando sono necessari. Inoltre, ogni volta che un

elemento scorre fuori dallo schermo la sua visualizzazione non viene distrutta ma è riutilizzata (riciclata) per quelli nuovi che occorre mostrare. Questo riuso migliora notevolmente le prestazioni.

Per poter usare la RecyclerView occorre creare due classi fondamentali:

- **Recycler View Adapter:** classe che prende in input l'elenco di dati da visualizzare e per ognuno di essi gli associa un View Holder.
- **View Holder:** classe che definisce la struttura di un singolo elemento della lista.

Grazie a questa sua modularità la lista può essere riusata in vari punti dell'applicazione ed è facilmente modificabile.

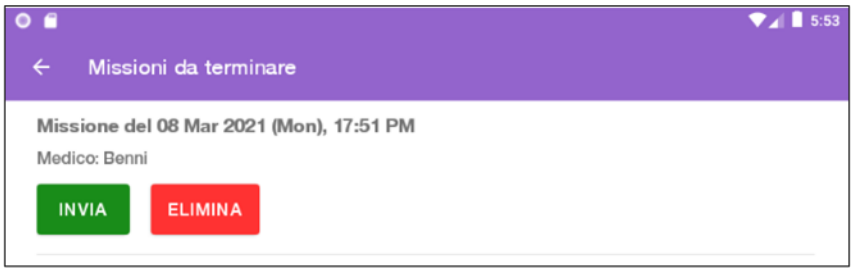


Figura 4.6: Elemento lista implementata con Recycler View

Nella Figura 4.6 è mostrata la rappresentazione a video di un elemento della lista implementata usando la libreria RecyclerView. Come si può vedere è composto da **un titolo** contenente la data, l'ora e il giorno di avvio della missione, un **sottotitolo** che indica il nominativo del leader PreH e due pulsanti: **invia ed elimina.** Premendo *elimina* tutti i dati locali dello specifico soccorso vengono cancellati e l'elenco a video viene aggiornato facendolo scomparire dalla schermata, invece con *invia* viene tentato l'inoltro al servizio di back-end.

Invio al back-end delle missioni salvate sul dispositivo Quando si tenta l'invio di un soccorso salvato sul dispositivo, poiché è possibile che alcune informazioni siano già state trasmesse in real-time durante l'esecuzione della missione, è necessario sovrascriverle con quelle mantenute in memoria locale che sono ovviamente le più aggiornate.

Per inviare richieste al back-end si è utilizzato il componente **WorkManager** messo a disposizione dalla libreria *Android Jetpack*, attraverso il quale è possibile far eseguire dei compiti specifici su thread asincroni in background. Come mostrato nel Listato 4.7, tali task sono definiti estendendo la classe *Worker* e il codice da far mandare in esecuzione al WorkerManager è contenuto

all'interno del metodo *override doWork()*. Questi thread vengono processati solamente se soddisfano alcuni vincoli impostati durante la loro costruzione tramite il metodo *setConstraints()*. Nel nostro caso l'unico requisito richiesto è che la rete, Wi-Fi o dati mobili, sia disponibile.

```
class AdministrationWorker constructor(context: Context,
    workerParams: WorkerParameters) : Worker(context, workerParams) {

    private val repository: AdministrationRepository

    companion object {
        const val WORKER_TAG = "WORKER_ADMINISTRATION"
        const val ADMINISTRATION_KEY = "ADMINISTRATION_KEY"
        const val HTTP_KEY = "HTTP_KEY"
        fun builder() = OneTimeWorkRequest
            .Builder(AdministrationWorker::class.java)
            .setConstraints(WorkerUtils.networkConstraint)
            .addTag(WORKER_TAG)
    }

    init {
        val patientDao = PrehRoomDatabase.getDatabase(context)
            .patientDao()
        repository = AdministrationRepository(patientDao)
    }

    override fun doWork(): Result = handleCall().let {
        when (it?.status) {
            Resource.Status.SUCCESS -> Result.success()
            Resource.Status.ERROR -> Result.retry()
            else -> Result.failure()
        }
    }

    private fun handleCall() = when (getHttpMethod()) {
        HTTPMETHOD.POST -> repository
            .postAdministration(getAdministrationFromInputData())
        HTTPMETHOD.PUT -> repository.replaceAllAdministrations()
        else -> null
    }

    private fun getAdministrationFromInputData() =
        Gson().fromJson(inputData.getString(ADMINISTRATION_KEY),
        Administration::class.java)
```

```
    private fun getHttpMethod() =
        HTTPMETHOD.valueOf(inputData.getString(HTTP_KEY)!!)
}
```

Listato 4.7: AdministrationWorker.kt

Quando si preme sul pulsante di invio missione nel fragment delle missioni salvate vengono quindi costruiti tutti i worker specificando parametri consoni che permettano, tramite il repository specifico, di recuperare i dati necessari dal database locale e di richiamare le API del server che ne effettuino la sovrascrittura in quello remoto.

Nel Listato 4.7 per indicare al Worker l'intenzione di sovrascrivere tutte le somministrazioni è stato associato al parametro HTTP_KEY il valore **HTTP PUT**.

Ogni Worker costruito viene quindi accodato nel WorkManager il quale si occuperà di metterli in esecuzione uno consecutivamente all'altro.

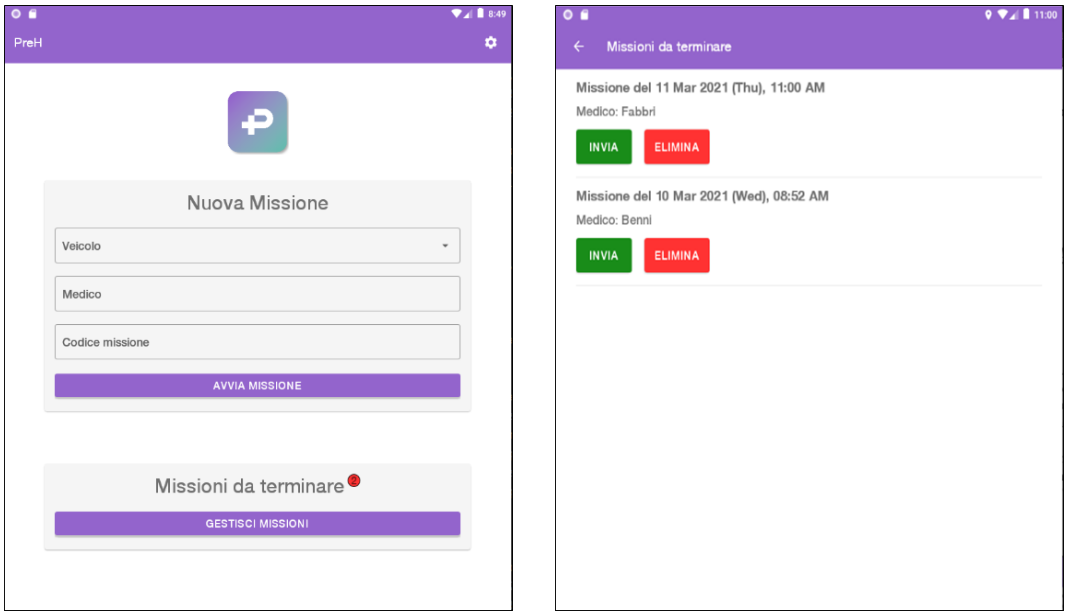
4.3.3 Interfaccia utente e User Experience

L'interfaccia utente dell'applicazione è stata strutturata in modo che risulti semplice ed immediata nel suo uso e che soddisfi alcuni dei requisiti esposti nella Sottosezione 3.2.2.

Innanzitutto **i colori ed i font** sono stati scelti attentamente in modo da consentire un'ottima visibilità anche nelle condizioni più sfavorevoli di luminosità. Per quanto riguarda le schermate, sono state rese tutte **responsive e scrollable** così da poter mandare in esecuzione l'applicazione anche su tablet con schermi di dimensioni diverse. Relativamente alla struttura, quando è stato possibile, per l'esposizione di informazioni correlate si è utilizzata la **navigazione a tab**, evitando così il cambiamento della schermata che avrebbe potuto causare confusione. Inoltre, per migliorare l'usabilità in ogni casella di input testo sono state usate delle **floating label** in modo da evidenziarne lo scopo anche una volta compilate.

Come è possibile vedere nella Figura 4.7a la **schermata iniziale** è composta da due pannelli: quello superiore permette l'inizio di una nuova missione, mentre quello inferiore consente l'accesso alla schermata delle missioni salvate sul dispositivo che non sono ancora state inviate correttamente al servizio di back-end (Figura 4.7b).

Per ottimizzare la User Experience, sopra il titolo "Missioni da terminare", è stato aggiunto un *badge circolare rosso* che indica il numero di missioni memorizzate nel tablet che occorre gestire.



(a) Schermata iniziale (b) Schermata missioni da terminare

Figura 4.7: UI Principali App

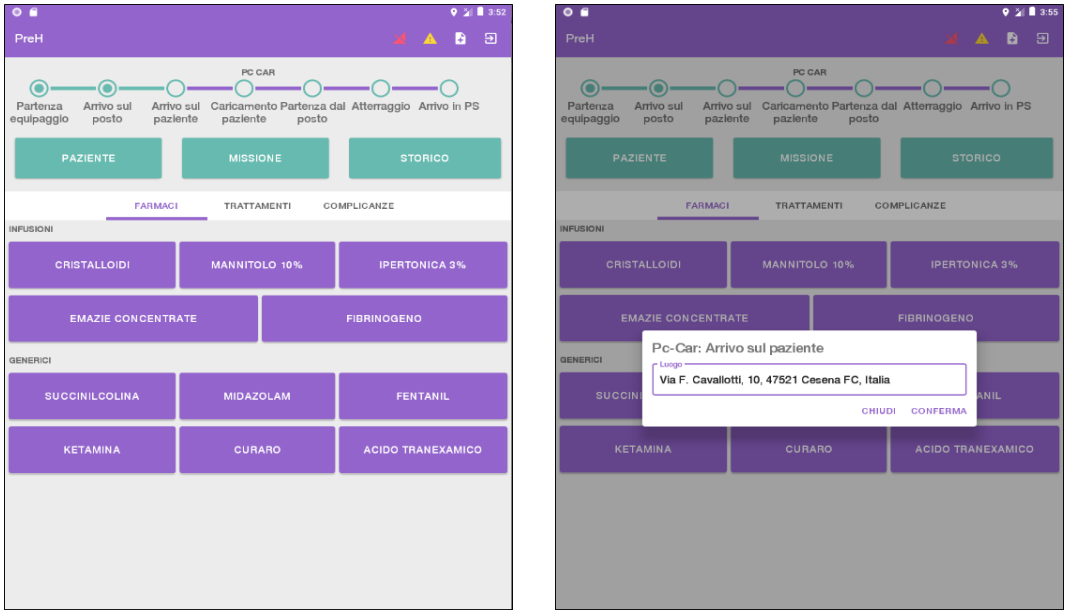
Una volta avviata la missione ci si ritrova nella **home dell'intervento** rappresentata nella Figura 4.8a. Tale schermata, come si può notare, è possibile suddividerla a livello visivo in due parti:

- **Sezione superiore:** si compone del *Pc Car* e di tre pulsanti denominati *Paziente, Missione e Storico*. Il primo consente l'accesso a una schermata che mostra e permette di modificare tutti i dati relativi al paziente, i quali sono suddivisi in tre schede di una navigazione a tab: *Anagrafica, Stato, Parametri Vitali*. Utilizzando il secondo pulsante si raggiunge una pagina che permette l'inserimento e la visualizzazione di tutte le informazioni relative all'evento che ha comportato l'avvio della missione. E, infine tramite il pulsante Storico si accede all'elenco contenente tutte le somministrazioni, trattamenti, manovre, complicanze già registrate dal medico corredate della data e ora corrispondenti.

  Il *Pc Car*, invece, consente di effettuare il tracciamento del team PreH. Per registrare il raggiungimento di uno step basta premere sul pallino corrispondente, inserire le informazioni richieste nel popup che appare [Figura 4.8b] e confermare, a quel punto si vedrà colorarsi in verde acqua la parte del percorso corrispondente. Alcune volte il sistema propone automaticamente il luogo richiesto per la registrazione poiché lo ricava per logica da quelli inseriti negli step precedenti o tramite la geolocalizzazione. Come voluto nei requisiti è obbligatorio inserire l'arrivo nei

vari stadi del pc-car seguendo l'ordine definito dal percorso, senza saltare nessuno step.

- **Sezione inferiore:** si compone di una navigazione a tab che consente l'accesso alle schede *Farmaci, Trattamenti, Complicanze* tramite le quali il medico può registrare le azioni svolte sul paziente o le complicanze verificatasi. La scheda dei trattamenti include anche le manovre.



(a) Schermata home missione (b) Pc Car Step

Figura 4.8: UI Principali App

Facendo sempre riferimento alla Figura 4.8a nella **toolbar** in alto a destra sono presenti alcune icone: partendo da sinistra, la prima in rosso viene mostrata quando la connessione è assente, la seconda in giallo è una notifica che indica la presenza di un trauma maggiore e premendo su quest'ultima apparirà un popup contenente la lista dei criteri attivi, la terza permette l'inserimento di eventuali note e l'ultima consente l'accesso alla schermata di terminazione missione mostrata in Figura 4.9.

Infine, per rendere semplice l'utilizzo dell'applicazione e per permettere un'apprendibilità veloce, molti passaggi sono stati guidati utilizzando **popup informativi e di conferma**. Ad esempio questi ultimi vengono mostrati ogni volta che l'utente richiede l'eliminazione della missione, mentre i primi vengono utilizzati per confermare l'esito positivo o negativo dell'invio dei dati del soccorso al servizio di back-end. In quest'ultimo caso se l'operazione non va

a buon fine a causa dell'assenza di rete, tale problema viene descritto esplicitamente nel popup poiché è frequente che i medici si scordino di riattivare la connessione dopo averla spenta quando sono saliti sull'elicottero.

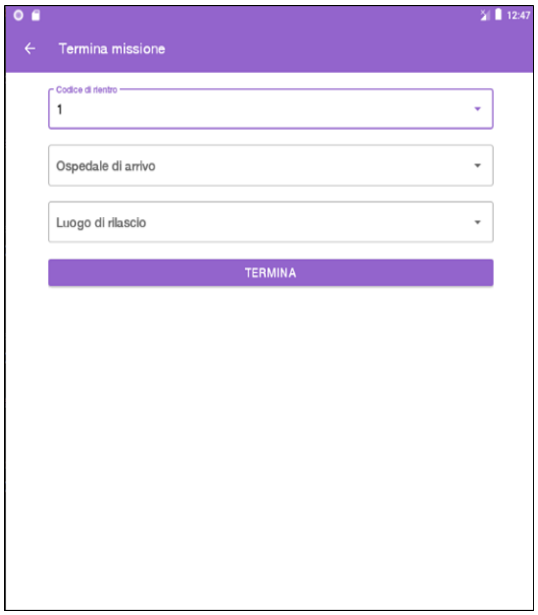


Figura 4.9: Schermata termina missione

4.4 Sviluppo PreH Service

Come già detto in precedenza, il back-end è stato implementato con l'ausilio del toolkit **Vert.x**. La classe principale *PrehService* mostrata nel Listato 4.8 è l'unico **Verticle** di cui il server si compone e tramite la quale si occupa della ricezione e della gestione delle richieste HTTP. Essa estende infatti la classe astratta *AbstractVerticle* e ne implementa il metodo *start()* all'interno del quale, mediante la funzione *createHttpServer()*, viene istanziato il server http che poi viene messo in ascolto su una specifica porta con il metodo *listen()*. Occorre mettere in evidenza che in tale creazione viene indicato anche l'handler che si occuperà di gestire tutte le richieste che sopraggiungeranno: è un oggetto *Router*, messo a disposizione dalla libreria *vertx-web*.

```
public class PrehService extends AbstractVerticle {
    @Override
    public void start(final Promise<Void> startPromise) {
      final HttpServer server =
          vertx.createHttpServer().requestHandler(createRouter());
      final int servicePort = httpConfig.getInteger(PORT);
```

```
      server.listen(servicePort, res -> {
         if (res.succeeded()) {
            LOG.info("Http server is running on port " + servicePort);
            startPromise.complete();
         } else {
            startPromise.fail(res.cause());
         }
      });
   }

   private Router createRouter() {
       final Router mainRouter = Router.router(vertx);
       final MongoClient mongoClient = MongoClient.create(vertx,
          mongoConfig);

       mainRouter.get(MAIN_ROUTE).handler(rc -> rc.response()
           .end("Entry point for the PreH API"));
       mainRouter.mountSubRouter(EVENTS_ROUTE, new
          EventRouter(vertx, mongoClient));
       mainRouter.mountSubRouter(MISSIONS_ROUTE, new
          MissionRouter(vertx, mongoClient));
       mainRouter.mountSubRouter(PATIENTS_ROUTE, new
          PatientRouter(vertx, mongoClient));
       mainRouter.mountSubRouter(LOGIN_ROUTE, new
          LoginRouter(vertx, mongoClient));
       return mainRouter;
   }
}
```

Listato 4.8: PrehService.java

Tale router principale viene creato dentro la funzione *createRouter()* nella quale gli vengono anche associati altri quattro sotto-router: *EventRouter, MissionRouter, PatientRouter* e *LoginRouter*. Ognuno di essi si occuperà della fornitura di specifiche API REST, che sono già state descritte nella Sottosezione 4.1.2, e utilizzando il riferimento all'oggetto della classe **MongoClient** passatogli durante l'associazione, potranno interagire con il database. La classe MongoClient è messa a disposizione dall'estensione *MongoDB Client* di Vert.x e permette di produrre un client in grado di eseguire delle query di ricerca, inserimento, modifica e cancellazione all'interno di un database MongoDB.

I singoli sotto-router sono implementati tutti similmente al Listato 4.9 che rappresenta nello specifico il codice della classe *PatientRouter*:

- tutti estendono dalla classe astratta *AbstractRouter* la quale definisce il

comportamento base che tutti i router condividono.

- il costruttore può contenere o meno l'associazione di altri sotto-router che permettono di gestire richieste con rotte ancora più dettagliate a partire da quella base del router al quale sono affiliati.
- tramite la funzione *setRouterHandlers()* vengono stabiliti gli handler delle singole richieste a seconda della rotta alla quale si riferiscono e al metodo http con le quali vengono effettuate.

```
public class PatientRouter extends AbstractRouter {
    public PatientRouter(final Vertx vertx, final MongoClient
        mongoClient) {
        super(vertx, mongoClient);
        mountSubRouter(BASE_PATH, new VitalSignsRouter(vertx,
            mongoClient));
        mountSubRouter(BASE_PATH, new ComplicationRouter(vertx,
            mongoClient));
        mountSubRouter(BASE_PATH, new AdministrationRouter(vertx,
            mongoClient));
        mountSubRouter(BASE_PATH, new ProcedureRouter(vertx,
            mongoClient));
        mountSubRouter(BASE_PATH, new TreatmentRouter(vertx,
            mongoClient));
    }

    @Override
    protected void setRouterHandlers() {
        get(BASE_PATH).handler(this::getPatients);
        post(BASE_PATH).handler(this::createPatient);
        get(PATIENT_PATH).handler(this::getPatientById);
        put(PATIENT_PATH).handler(this::updatePatient);
        delete(PATIENT_PATH).handler(this::deletePatient);
        /* extra routes */
        get(STATUS_PATH).handler(this::getStatus);
        patch(STATUS_PATH).handler(this::updateStatus);
        get(ANAGRAPHIC_PATH).handler(this::getAnagraphic);
        patch(ANAGRAPHIC_PATH).handler(this::updateAnagraphic);
        get(NOTES_PATH).handler(this::getNotes);
        patch(NOTES_PATH).handler(this::updateNotes);
    }
}
```

Listato 4.9: PatientRouter.java

Quindi, riepilogando, quando il router principale riceve una richiesta HTTP la affida al sotto-router con la rotta corrispondente, scegliendo tra PatientRouter, EventRouter, MissionRouter e LoginRouter. Quello designato a sua volta decide la funzione specifica che deve gestire la richiesta basandosi sul percorso indicato nella rotta e sul metodo HTTP con la quale è stata fatta oppure la delega ulteriormente a un altro sotto-router più specifico.

4.5 Sviluppo Dashboard

Come detto precedentemente nella Sottosezione 4.1.3, la dashboard è stata realizzata seguendo l'architettura di una **single-page-Application (SPA)**. Si è cercato di suddividere la sua implementazione in *moduli* corrispondenti alle singole interfacce in modo da privilegiare la comprensibilità e la manutenibilità del codice. Inoltre, grazie all'uso di **Bootstrap** tutte le schermate sono *responsive* e quindi è garantito che vengano visualizzate correttamente in schermi di varie dimensioni.

Il sito della dashboard si compone quindi di un'unica pagina nella quale vengono dinamicamente presentati tre contenuti principali: *login, home e missione.*

Pagina di Login
È la prima pagina che viene mostrata all'utente quando cerca di accedere al sito. Grazie ad essa, è possibile permettere l'accesso ai dati sensibili appartenenti ai pazienti esclusivamente alle persone autorizzate.

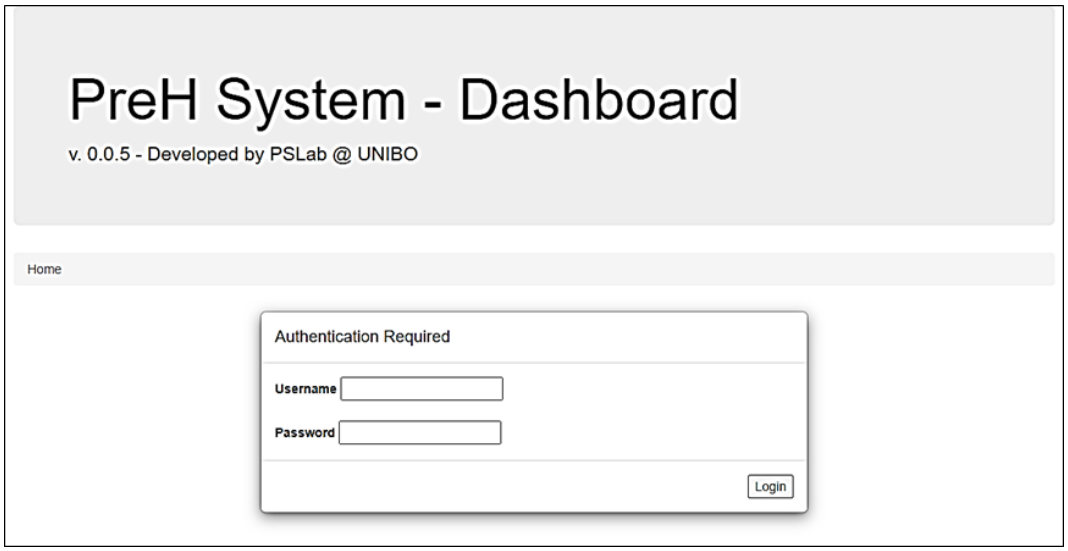


Figura 4.10: Schermata Login Dashboard

Quando viene effettuato il submit nel form della Figura 4.10, viene richiamata l'API del back-end corrispondente, tramite una richiesta POST alla rotta *"/api/v1/login"*, per verificare che le credenziali siano corrette.

Pagina Home

Una volta effettuato il login si accede alla home del sito contenente la lista delle missioni terminate e in corso [Figura 4.11]. Per ciascuna di esse viene visualizzato *l'id, il veicolo utilizzato per il soccorso, il medico responsabile* e se la missione non è completata anche *il tempo di attività.* Se un nuovo intervento viene avviato o uno non concluso viene terminato, tali elenchi vengono aggiornati in tempo reale.

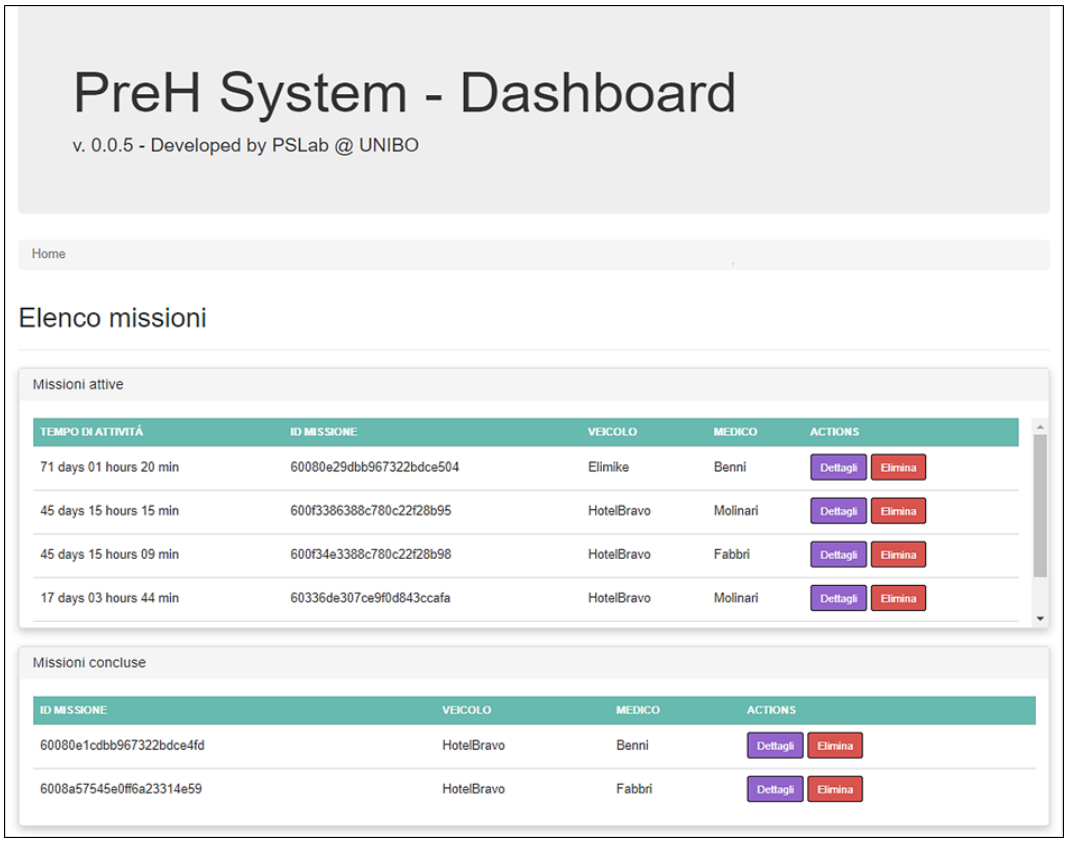


Figura 4.11: Schermata Home Dashboard

In relazione a tale pagina, sono stati apportati due principali miglioramenti: è stata data la possibilità di **eliminare le missioni** memorizzate nel database centrale utilizzando le API REST messe a disposizione dal server e per i soccorsi attivi, è stato aggiunto come ulteriore campo nella tabella il **tempo di attività** che misura il periodo trascorso da quando ciascuno di essi è stato

iniziato. Tale parametro è infatti molto utile perché se è elevato indica che la missione non è stata ancora completamente inviata al server. Questo si verifica in due casi:

1. *l'utente termina la missione, la salva sul dispositivo ma poi si dimentica di inviarla.* In tale contesto è bene sollecitarlo e richiederne l'inoltro.

2. *l'utente ha eliminato la missione sul dispositivo.* È possibile infatti che al momento di attivazione di un soccorso, poiché la rete è presente, i dati vengano subito inviati al server e quindi memorizzati nel database centrale. Se l'utente poi decide di eliminare la missione, tale azione cancellerà solo i dati locali e non quelli nel DB remoto (poi visualizzati anche sulla dashboard). Per questo motivo tale missione su questa schermata risulterà sempre attiva e non verrà mai terminata.

   In questa situazione occorre richiedere conferma all'utente dell'effettiva cancellazione e provvedere all'eliminazione manuale dei dati anche nel database centrale richiamando le specifiche API messe a disposizione dal server. Per far ciò basta premere sul pulsante **elimina** nella tabella in corrispondenza della riga della missione, infatti così sarà inviata una *richiesta HTTP Delete* al PreH Service che provvederà ad eliminarla insieme anche a tutti i dati del paziente e dell'evento a lei correlati. Tale funzionalità potrà comunque tornare utile anche per tutte le missioni create come test dai medici o per quelle che devono essere annullate per altri motivi.

Pagina della missione

Dalla Home è possibile scegliere una missione della quale visualizzarne maggiori informazioni premendo il pulsante **"Dettagli"** presente nella riga corrispondente della tabella. Il contenuto della pagina verrà sostituito dinamicamente con quello in Figura 4.12 che espone, nella *sezione superiore*, tutti i dati relativi alla missione, al tracciamento del team e all'evento e in *quella inferiore* le informazioni inerenti al paziente (anagrafica, somministrazioni, trattamenti, manovre, complicanze, stato e parametri vitali). La scelta di accorpare tutto in un'unica schermata è stata intrapresa soprattutto per ottimizzare i tempi di accesso ai dati per i medici in ospedale.

Un fatto da mettere in evidenza è che se la missione è in corso e i soccorritori aggiornano i dati, la modifica viene visualizzata in *real-time.*

Per permettere di mostrare i cambiamenti in tempo reale, non solo nella schermata specifica per una missione ma anche nelle tabelle della home, vengono effettuate delle chiamate HTTP *asincrone, periodiche* alle API messe a disposizione dal server utilizzando **AJAX**. Tali HTTP requests sono effettuate

da script di Javascript senza la necessità di dover effettuare submit di form ne di ricaricare la pagina.

In particolare, nella schermata specifica relativa ad una missione [Figura 4.12], le diverse sezioni che contengono i dati del Paziente, Missione ed Evento interagiscono con il back-end ognuna in *maniera indipendente* dall'altra effettuando distinte richieste HTTP Ajax ai router corrispondenti.

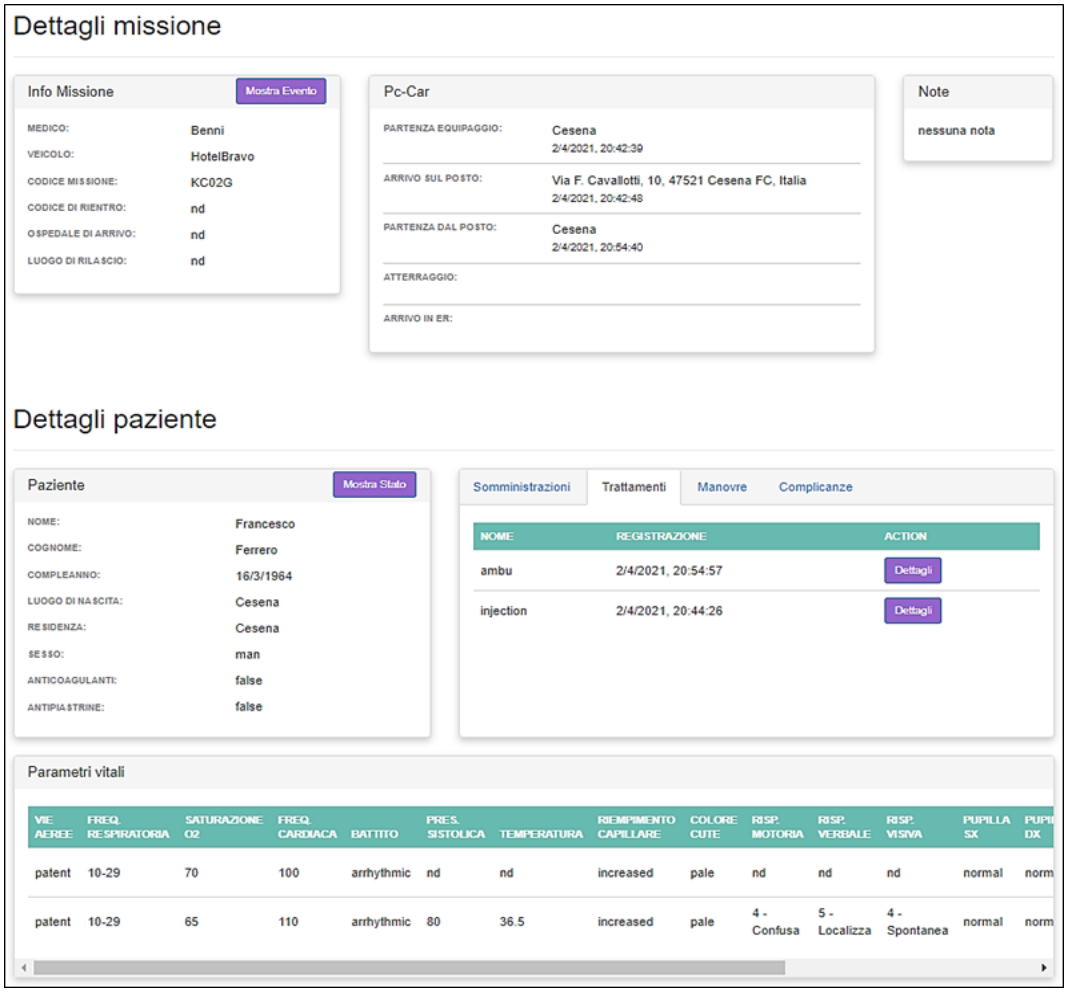


Figura 4.12: Schermata specifica di una missione Dashboard

Nel Listato 4.10 si può osservare un esempio di come avviene tale interazione: viene impostato un timer che, tramite la funzione *getJSON*, effettua periodicamente una richiesta GET alla rotta */api/v1/missions/missionId*, sfruttando Ajax e passando l'id della missione come parametro. La risposta a tale richiesta è un JSON contenente tutti i dati relativi alla missione. Que-

sti, dopo aver verificato che non siano nulli, vengono impostati nei pannelli corrispondenti *Info Missione e Pc-Car* senza ricaricare la pagina.

```
function loadMission(missionId) {
    $.getJSON(‘/api/v1/missions/${missionId}‘, function (mission) {
        const missionHtml = $("#mission-info");
        missionHtml.empty();

        if (typeof mission !== "undefined") {
            missionHtml.append(
                $(’<p>’).text(mission.medic),
                $(’<p>’).text(mission.vehicle),
                $(’<p>’).text(checkEmptyValue(mission.missionCode))
            );
        }

        setReturnInformationPanel(mission.returnInformation);
        setTrackingPanel(mission.tracking);
        timeoutManager.addTimeout(function () {
            loadMission(missionId) }, resourceType.MISSION, 1000);
    });
}
```

Listato 4.10: Richiesta Ajax - Mission.js

Capitolo 5

Validazione del prototipo

La validazione del nuovo prototipo creato è stata eseguita percorrendo due strade: da una parte è stato testato approfonditamente durante tutta la fase di sviluppo controllando che il comportamento fosse coerente con quello atteso, dall'altra ci si è affidati alla valutazione degli esperti del dominio per verificare se effettivamente tutti i requisiti richiesti erano stati soddisfatti pienamente.

Nella prima sezione di questo capitolo quindi si affronterà come si è svolta la validazione funzionale e gli esiti ottenuti. Saranno anche illustrate le varie imperfezioni emerse sia a livello implementativo che di usabilità e le possibili soluzioni applicabili. In seguito saranno descritti i risultati della valutazione del prototipo effettuata dall'esperto del dominio ponendo evidenza anche sulle successive modifiche che è stato necessario apportare poiché inizialmente dall'analisi del dominio non erano emerse come rilevanti. Queste hanno permesso di avvicinarsi ancor di più a una versione del sistema che possa essere utilizzata nel contesto reale.

5.1 Validazione funzionale

Il prototipo PreH è stato validato approfonditamente durante tutta la fase di sviluppo, concentrandosi in particolare sul verificare il corretto funzionamento delle parti di front-end (App e Dashboard) e che l'interazione tra i due client e il server avvenisse regolarmente senza errori.

5.1.1 Strumenti e metodologie usate per la validazione

Per validare *l'applicazione* è stato fondamentale l'uso dell'**Android Virtual Device (AVD)** messo a disposizione dall'IDE Android Studio che ha permesso di simularne l'uso sul tablet *Galaxy Tab Active2* utilizzato dai soccorritori. Gli stessi test sono stati poi eseguiti anche su dispositivi mobile fisici

per valutare la portabilità della PreH App e la responsività delle sue schermate. Per controllare l'esattezza delle informazioni memorizzate nella memoria locale del dispositivo durante l'esecuzione delle varie operazioni, si è invece fatto ausilio del tool **Database Inspector** messo sempre a disposizione dall'IDE di sviluppo.

Oltre a verificare il corretto funzionamento negli scenari base di utilizzo dell'app sono state riprodotte anche *situazioni di stress* come ad esempio:

- interruzione improvvisa della rete sia durante l'invio di una missione al servizio di back-end sia dopo l'inserimento di eventi e informazioni.
- chiusura improvvisa dell'applicazione.
- spegnimento inatteso del dispositivo.
- invio di grandi moli di dati al server derivanti da numerosi inserimenti di somministrazioni, trattamenti, complicazioni, manovre e modifiche consecutive all'anagrafica, parametri vitali, stato paziente, dati dell'evento e note del soccorso.
- switch veloce tra le varie interfacce per verificarne la fluidità.

Per validare il *funzionamento del server* invece si è semplicemente verificata la corretta gestione delle richieste HTTP che venivano ricevute, controllando da una parte i dati esposti sulla dashboard e dall'altra accertando l'esattezza di quelli inseriti nel database centrale con l'ausilio del programma **Robo3T**.

Infine, per testare la *correttezza della dashboard* si è controllato innanzitutto la validità delle informazioni esposte, poi utilizzando gli **Strumenti per sviluppatore** messi a disposizione dal browser si è verificata la giusta cattura e gestione delle interazioni dell'utente con il contenuto del sito e che il design della single-page fosse effettivamente responsivo. È stata testata anche la sua compatibilità con diversi browser ed è stato validato sia il markup html che l'accessibilità utilizzando dei validatori online appositi.

5.1.2 Risultati e possibili futuri miglioramenti

Tutti i test effettuati hanno prodotto risultati soddisfacenti poiché coerenti con quelli attesi ed i bug minori che si sono manifestati, sono stati prontamente corretti. La *manutenzione correttiva*[1] effettuata è stata di facile esecuzione grazie all'appropriatezza delle tecnologie e dell'architettura scelte.

[1]**Manutenzione correttiva:** elimina gli errori che causano il malfunzionamento del sistema. Possono essere errori presenti sin dall'inizio o introdotti con precedenti interventi di manutenzione.

Si è valutata anche la questione che l'applicazione PreH è stata sviluppata ad hoc per Android e non è stata testata su altri sistemi operativi mobile. In realtà tale fatto non incide sulla sua idoneità poiché essa non è destinata ad essere pubblicata su una piattaforma di distribuzione app ma deve solo essere installata sui tablet che vengono forniti ai soccorritori che sono tutti dispositivi dello stesso modello con sistema operativo Android.

Per migliorare ulteriormente il sistema permettendogli di soddisfare ancor di più i requisiti non funzionali, si potrebbe in primis aumentare la sicurezza del passaggio dati dal server alla dashboard migrando *dal protocollo HTTP a HTTPS*. In questo modo infatti, viene fornita una connessione **SSL/TLS**[2] crittografata che consente di trasmettere le informazioni in maniera cifrata e non più in chiaro.

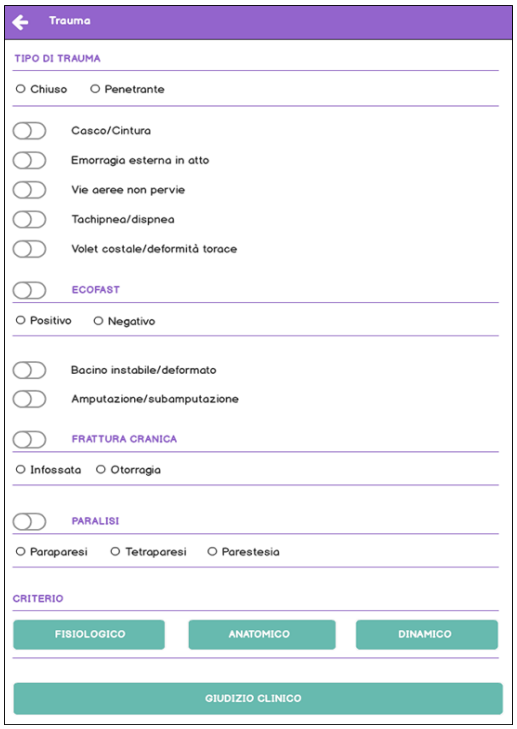
Inoltre, come già detto nei capitoli precedenti per realizzare l'aggiornamento in real-time dei dati presentati sulla dashboard ci si basa su un meccanismo di *polling* che è indiscutibilmente di facile implementazione ma non è sicuramente la soluzione ottimale poiché, anche se non ci sono variazioni nelle informazioni, esse vengono in ogni caso trasmesse alla dashboard quando questa le richiede al server. Quindi, come secondo cambiamento, si potrebbe valutare di sostituire tale approccio con il **Web Socket**[3]**:** è un protocollo che consente una comunicazione asincrona e full-duplex (in entrambe le direzioni in maniera contemporanea) su connessione TCP. È sufficiente che venga aperto il collegamento tra client (Dashboard) e server (PreH Service) utilizzando *l'handshake del protocollo WebSocket*, tramite il quale solitamente il client invia tutte le informazioni necessarie per iniziare la trasmissione dei dati. Una volta effettuato, il canale di comunicazione rimane aperto, quindi quando il server riceve nuove informazioni dall'applicazione può attivarsi e trasmetterle alla dashboard senza aspettare una sua richiesta realizzando in questo modo un vero e proprio real-time, infatti l'unico delay rimanente è quello del trasferimento dati. Oltre a risolvere il problema del polling, questo approccio permette di non aver nessun spreco di banda e bassa latenza.

Infine, come ultima modifica, si potrebbe perfezionare *l'usabilità delle varie schermate* dell'applicazione apportando alcuni miglioramenti descritti qui di seguito. Per facilitare la comprensione vengono presentati, come esempi, nelle seguenti Figure 5.1a e 5.1b i mockup relativi alle interfacce dello stato del paziente e dei trattamenti poiché rappresentano le pagine più complicate a livello visivo.

[2]**SSL/TLS**: Secure Sockets Layer/Transport Layer Security.

[3]Per dettagli: `https://italiancoders.it/websocket-la-panacea-al-polling/`

(a) Mockup schermata Trattamenti



(b) Mockup schermata Stato Paziente

Figura 5.1: Mockups con miglioramenti di usabilità

- Il *font* utilizzato all'interno dei text input e quello delle rispettive floating label che li descrivono *potrebbe essere ingrandito*.
- *I titoli* delle varie sezioni (rianimazione cardiopolmonare, vie aeree, respiro e circolo in Figura 5.1a) potrebbero *essere ingranditi e colorati in viola*.
- Utilizzare dei *divisori viola tra le sezioni* e *grigio scuro per suddividere le sotto-sezioni* al loro interno.
- Aumentare le *distanze* tra le sezioni.
- Poiché è emerso il problema che una volta selezionato un Radio Button non era più possibile dismettere la selezione, si è deciso di adottare come soluzione l'uso di uno *switch per abilitare/disabilitare la possibilità di selezionare le alternative nei radio group* che riguardano attività che possono non essere sempre presenti in ogni soccorso (vedi la frattura cranica, la paralisi, o l'ecofast nella Figura 5.1b).
- Mettere gli *switch a sinistra del testo* e non a destra, fortificandone così il riferimento.

- *Uniformare i colori dei pulsanti* per ogni schermata scegliendo o tra il colore primario -viola- o quello secondario -verde acqua- (nelle Figure 5.1a e 5.1b è stato scelto quest'ultimo).
- *Trasformare da pulsanti a switch* tutte le operazioni per le quali non è necessario immettere ulteriori informazioni ma occorre solo confermare la loro esecuzione (come emostasi o pelvic binder nella Figura 5.1a) .

Riferendosi nello specifico alla *schermata dello stato del paziente*, le sezioni *Paralisi* e *Criterio* erano una accanto all'altra con rispettivamente i radio button e i pulsanti disposti in verticale. Per evitare di creare confusione e permettere di capire chiaramente l'inizio e la fine di ciascuna sezione si consiglia di metterle in parallelo orizzontalmente come mostrato nella Figura 5.1b.

Infine, si potrebbe rendere più usabile il *Pc-Car* come in Figura 5.2: sono stati cambiati i colori utilizzati per denotare il percorso svolto e non svolto rispettivamente da viola e verde acqua a verde e grigio, è stata inserita la freccia come collegamento tra uno step e l'altro piuttosto che la sola linea poiché permette di indicare meglio il verso di percorrenza del percorso ed infine è stato aggiunto un divisore dai pulsanti paziente, trauma e storico per fortificare maggiormente la loro non correlazione.

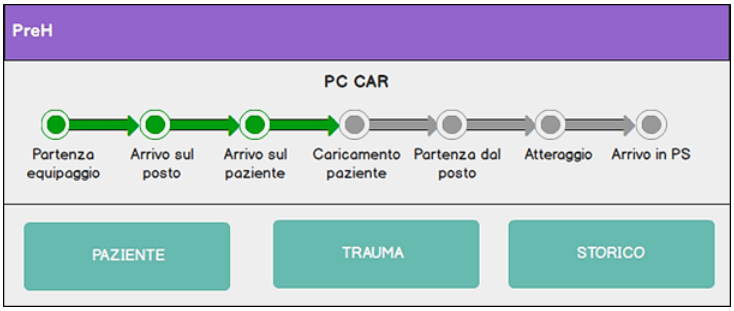


Figura 5.2: Mockup Pc-Car

5.2 Validazione esperti del dominio

Il sistema PreH era già utilizzato in via sperimentale nel team di anestesisti-rianimatori del Trauma Center di AUSL Romagna in cooperazione con la Centrale Operativa 118 Emilia Romagna. Come referente per effettuare i test sul nuovo prototipo è stato scelto il Dott. Marco Benni, anestesista-rianimatore presso la U.O. di Terapia Intensiva dell'Ospedale Bufalini di Cesena con mansioni di soccorritore in elisoccorso.

La validazione con l'esperto del dominio è stata condotta tramite alcune **interviste** effettuate a posteriori di rilasci incrementali del sistema. Esse sono

state molto proficue poiché da una parte hanno permesso di constatare che *le nuove funzionalità implementate sono corrette e conformi a quanto richiesto dai requisiti*, la nuova versione del prototipo infatti è stata valutata molto più usabile ed accessibile e non sono stati riscontrati particolari malfunzionamenti, dall'altra hanno fatto risultare la necessità di *rivedere alcuni aspetti* che non erano emersi come rilevanti dall'analisi del dominio svolta inizialmente. Ciò ha comportato i seguenti cambiamenti:

- Nella *schermata iniziale* e in quella *relativa all'evento* si sono inseriti degli **Spinner** (menù a tendina) nei campi di testo medico, codice missione e codice di invio per velocizzare l'inserimento delle informazioni. In particolare, negli ultimi due citati è stato necessario applicare una tripla tendina poiché i codici sono costituiti da tre parti variabili.

- È emerso che il *codice missione* corrisponde con il *codice di invio*, quindi se il primo viene inserito nella schermata di avvio missione, viene riproposto automaticamente anche nel campo codice di invio presente nella pagina relativa all'evento.

- Nel *Pc-Car* quando si preme sullo step "partenza equipaggio" si è preferito far attivare la geolocalizzazione per determinare automaticamente la posizione del team, mostrandola anche sulla mappa [Figura 5.3], piuttosto che farla inserire manualmente allo scopo di risparmiare tempo ed evitare errori d'inserimento.

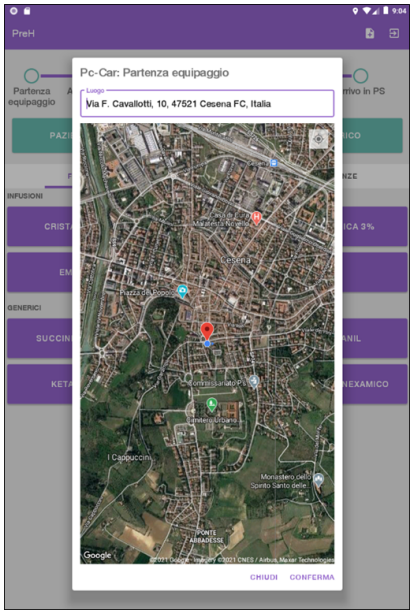


Figura 5.3: Step partenza equipaggio Pc-Car

- La *sezione delle manovre* è stata inserita in una scheda apposita della navigazione a tab al fine di snellire i contenuti della schermata dei trattamenti, ove era inclusa precedentemente.

- Alcune *schermate dell'applicazione* sono state riorganizzate nell'ottica di permettere la gestione del tracciamento di tutte le patologie e non solo il trauma, in particolare:

  - La schermata relativa all'evento, raggiungibile con il tasto Missione, è stata spostata *in una scheda apposita della navigazione a tab riguardante il paziente*, accanto a quelle dell'anagrafica e dei parametri vitali. La scelta è derivata dal fatto che queste tre pagine sono uguali per tutti i pazienti (sia traumatizzati che non).
  - Al posto del tasto verde acqua Missione è stato immesso quello del *Trauma* che da accesso alla schermata dello stato del paziente. Essa è infatti l'unica relativa solo ai soggetti traumatizzati.

- È emersa la necessità di tracciare all'interno dello *storico* anche *l'inserimento dei parametri vitali e degli step del Pc-Car* corredati rispettivamente dall'orario di rilevamento e orario e luogo di arrivo. Si è quindi organizzata la schermata in sezioni, date dagli step del Pc-Car, all'interno delle quali sono stati elencati tutti i trattamenti, manovre, parametri vitali, somministrazioni e complicanze registrate durante ogni step. Ogni evento è comprensivo di orario e della possibilità di visualizzare maggiori dettagli, se presenti.

Si è valutato, con il medico, anche l'eventuale possibilità di diversificare nell'app sin dall'inizio un percorso specifico da seguire per ogni macropatologia (trauma, patologia cardiaca, ictus...) che comprendesse ovviamente una visualizzazione ad hoc. Questo però è stato considerato non attuabile poiché il dominio non consente di dare una diagnosi precisa al momento dell'avvio della missione e anche perché è possibile che un soggetto abbia più patologie nello stesso momento.

Conclusioni e sviluppi futuri

Questa tesi si colloca nel contesto del progetto *Tracking For Care (T4C)* ed ha avuto l'obiettivo di estendere e potenziare il prototipo già esistente che espandeva il sistema *Trauma Tracker* nella gestione del tracciamento della fase preospedaliera di un soccorso. Attualmente infatti tale monitoraggio è svolto utilizzando delle schede cartacee che non permettono però una raccolta di informazioni corretta, completa ed accurata, una facile trasmissione delle informazioni raccolte a Trauma Tracker e soprattutto non consentono di conoscere in anticipo le condizioni del soggetto trasportato per i medici che lo attendono in ospedale.

Il *progetto PreH* è stato quindi fondamentale nella risoluzione di questi problemi ed il nuovo prototipo ha permesso di avvicinarsi ancora di più a una versione che possa diventare operativa sul campo. La validazione ha confermato il raggiungimento di tutti gli obiettivi preposti da questo elaborato e grazie alle nuove funzionalità implementate il sistema è diventato molto più usabile ed accessibile per gli utenti finali. Fondamentale è stata la valutazione effettuata dal Dott. Marco Benni poiché ha consentito di constatare che il nuovo prototipo ha soddisfatto pienamente i requisiti richiesti dai medici, grazie anche alle ulteriori modifiche apportate dopo i vari deployment che inizialmente non erano emerse come necessarie dall'analisi dominio. Si è assodato però che è opportuno effettuare ulteriori piccoli cambiamenti nelle varie componenti del sistema per ottimizzarne l'usabilità, l'efficienza e conformarle pienamente alla gestione del tracciamento di tutte le patologie generalizzando dal solo trauma. Comunque, grazie all'accortezza posta in particolare durante le fasi di analisi e progettazione ma anche di sviluppo e alle tecnologie implementative scelte, il prototipo sarà facilmente manutenibile ed estendibile. Ulteriori **sviluppi futuri** potrebbero essere:

- **Ampliamento Dashboard:** sarebbe utile *introdurre allarmi o sistemi di notifica* che avvisassero i medici in ospedale del verificarsi di specifici eventi come l'avvio di una nuova missione, la conclusione di un soccorso oppure l'arrivo in corso di un paziente con un trauma maggiore o con codice rosso. Potrebbe essere valevole ricevere una notifica anche quando

il tempo di attività di una missione è superiore a una certa soglia e quindi sarebbe opportuno eliminarla o sollecitare il medico soccorritore. In tal caso si potrebbe creare un apposito *sistema di comunicazione* che permetta tramite uno specifico pulsante di inviare un messaggio sms al tablet o una notifica all'interno dell'applicazione.

Un ultimo possibile ampliamento potrebbe essere aggiungere un bottone che permetta di generare un *referto in formato pdf o csv* contenente tutti i dati relativi ad una specifica missione. Tale documento potrà poi essere stampato per essere immesso nella cartella clinica PreH e/o nell'archivio cartaceo della centrale operativa 118, oppure importato all'interno del database Access che attualmente viene usato per raccogliere la documentazione digitale.

- **Perfezionamento usabilità App:** è opportuno migliorare l'aspetto grafico dell'applicazione secondo lo studio svolto nella Sottosezione 5.1.2 del capitolo di validazione.

- **Implementazione protocollo HTTPS:** nella comunicazione di dati dal server alla dashboard è bene migrare dal protocollo HTTP a HTTPS per cifrare le informazioni trasmesse aumentando così la sicurezza.

- **Sostituzione del polling della Dashboard:** è bene aumentare l'efficienza della richiesta dati al server sostituendo il polling *con il protocollo WebSocket* per consentire una comunicazione asincrona full-duplex in tempo reale. In questo modo la dashboard riceverà i dati solo se sono variati.

- **Condivisione foto e video in real-time:** mantenendo una visione molto ampia riguardo i possibili sviluppi del sistema sarebbe utile senza dubbio poter *inviare file multimediali dall'app alla Dashboard* avvalendosi di smart glasses o altri wearable devices.

Ringraziamenti

Vorrei ringraziare innanzitutto la mia famiglia che mi ha sostenuto in ogni modo possibile nel corso di questo percorso e soprattutto nei momenti più difficili, permettendomi di trovare la forza di continuare e di raggiungere tale traguardo che senza loro sarebbe stato molto più arduo da conquistare.

Grazie in particolare a mia mamma Sabrina che è sempre stata al mio fianco nel momento del bisogno, consigliandomi e confortandomi quando necessario.

Grazie a mio padre Severino che mi ha appoggiato in tutte le scelte che ho voluto intraprendere nella vita e anche per quella universitaria.

Grazie al mio fidanzato Alessandro, che oltre a sopportarmi e supportarmi da quando avevo 14 anni, mi è sempre stato accanto durante questi 3 anni, incoraggiandomi e sostenendomi. Affrontando insieme il percorso di Ingegneria, anche se in facoltà diverse, è stata la persona che meglio ha compreso e con la quale ho maggiormente condiviso le mie preoccupazioni, i fallimenti, ma anche i successi e le soddisfazioni che l'università può dare. Grazie soprattutto per la sua sincerità dimostrata nel mostrarmi quando sbagliavo e per avermi dato tutte le carte necessarie per prendere consapevolezza delle mie capacità e per credere un po' più in me stessa.

Dedico un ringraziamento speciale ai miei nonni, Benito e Giordana, che in questi anni universitari si sono sempre interessati e hanno avuto un pensiero per me.

Ringrazio inoltre tutte le mie amiche e i miei amici, anche quelli conosciuti grazie all'università, che anche se alcune volte purtroppo li ho dovuti mettere da parte per dare priorità allo studio, non hanno smesso di volermi bene e di interessarsi a me. Grazie in particolare a Giulia che mi è stata accanto in questo percorso e mi ha ascoltato e consigliato quando ne avevo bisogno.

Ringrazio infine il Professore Alessandro Ricci e l'Ing. Angelo Croatti per avermi permesso di partecipare a questo progetto così interessante e per la pazienza e la costanza con cui mi hanno seguito durante questi mesi di lavoro. Un ringraziamento anche al Dott. Marco Benni per la disponibilità mostrata nei vari incontri.

Bibliografia

[1] Eversense. `https://global.eversensediabetes.com`.

[2] Jetpack, guida all'architettura dell'app. `https://developer.android.com/jetpack/guide`.

[3] Propeller health. `https://www.propellerhealth.com`.

[4] Diabetes facts & figures. `https://idf.org/aboutdiabetes/what-is-diabetes/facts-figures.html`, Feb 2020.

[5] Pierangelo Afferni, Mario Merone, and Paolo Soda. Hospital 4.0 and its innovation in methodologies and technologies. In *2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS)*, pages 333–338, 2018.

[6] Paolo Colli Franzone. *The healthcare digital revolution*. PKE, 2018.

[7] Simona Lissemore. Iot in sanità: una rivoluzione possibile. *Inno 3, Milano*, 2019.

[8] Tarouco L.MR, Bertholdo L.M., Granville L.Z., Arbiza L.MR, Carbone F., Marotta M., and de Santanna J.JC. Internet delle cose nella sanità: problemi di interoperatibilità e di sicurezza. In *Atti della IEEE International Conference Communications*, pages pp. 6121 – 6125, 2012, giugno.

[9] Alberto Marfoglia. *Progettazione e sviluppo di un sistema software per il tracciamento del trauma in fase preospedaliera*. Tesi di laurea triennale, 2019-2020.

[10] Geertruid Marres, Luc Taal, M. Bemelman, Jos Bouman, and Luke Leenen. Online victim tracking and tracing system (vitts) for major incident casualties. *Prehospital and disaster medicine*, 28:1–9, 05 2013.

[11] S. Montagna, A. Croatti, A. Ricci, V. Agnoletti, and V. Albarello. Pervasive tracking for time-dependent acute patient flow: a case study in

trauma management. In *2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS)*, pages 237–240, 2019.

[12] Sara Montagna, Angelo Croatti, Alessandro Ricci, Vanni Agnoletti, Vittorio Albarello, and Emiliano Gamberini. Real-time tracking and documentation in trauma management. *Health Informatics Journal*, 26(1):328–341, 2020. PMID: 30726161.

[13] Collins SA, Cato K, Albers D, et al. Relationship between nursing documentation and patients' mortality. *American Journal of Critical Care: An Official Publication, American Association of Critical-Care Nurses*, 22(4):306–313, 2013.