ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UN SISTEMA MOBILE PER IL TRACCIAMENTO PREOSPEDALIERO

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
TOMMASO BOMBARDI

Co-relatori
Ing. ANGELO CROATTI
Dott. VITTORIO ALBARELLO

Seconda Sessione di Laurea
Anno Accademico 2017 – 2018

PAROLE CHIAVE

TraumaTracker

PreH Soccorso

Dispositivi mobile

Cross-platform

NativeScript

Angular

a Paola e Alberto

Indice

Introduzione

Il progresso tecnologico, che si è verificato negli ultimi anni nei paesi più sviluppati, ha portato e porterà grandi cambiamenti nella società, tanto che oggi si parla di quarta rivoluzione industriale. In questo processo di rinnovamento un ruolo centrale è ricoperto dalle tecnologie informatiche, le cui applicazioni sono in grado di portare vantaggi in moltissimi ambiti.

Un settore in cui le tecnologie informatiche possono fornire un valido contributo è quello medico. Mettendo a disposizione strumenti che supportino il lavoro dei medici, sarà infatti possibile ottenere benefici sia in termini di miglioramento della qualità delle cure sia in termini di riduzione dei costi.

In quest'ottica si inserisce il progetto TraumaTracker, che è nato della collaborazione tra un gruppo di ricerca del Dipartimento di Informatica - Scienza e Ingegneria (DISI) dell'Università di Bologna e l'unità operativa di anestesia e rianimazione del Dipartimento Chirurgico e Grandi Traumi Cesena - Ausl Romagna. Trauma Tracker fornisce un valido aiuto ai medici impegnati nella gestione dei traumi e si pone come principale obiettivo la realizzazione del tracciamento sistematico di tutte le informazioni relative a un trauma. Infatti, attraverso il tracciamento, è possibile ottenere un aumento sia della qualità sia della quantità dei dati raccolti e produrre una documentazione accurata e automatizzata (fondamentale per migliorare la qualità delle cure).

Dopo aver toccato con mano (grazie a TraumaTracker) i vantaggi portati da un sistema informatico per il tracciamento di dati, i medici dell'ospedale Bufalini di Cesena hanno fatto emergere un'ulteriore necessità: eseguire il tracciamento anche nell'ambiente preospedaliero. TraumaTracker entra in funzione quando il paziente raggiunge la struttura ospedaliera, ma vi è una fase precedente, quella definita "preospedaliera", che è cruciale per la sopravvivenza del paziente e che attualmente viene documentata manualmente.

Da questa necessità, e grazie all'ulteriore collaborazione con il Servizio di Elisoccorso Ausl Romagna, il 118 Romagna Soccorso e il Dipartimento di Emergenza Ausl Romagna, è nato il progetto PreH Soccorso. Esso si pone l'obiettivo di realizzare un sistema per il tracciamento preospedaliero, che non dovrà essere un'estensione di TraumaTracker, ma sarà autonomo e permetterà il tracciamento di avvenimenti generici (senza trattare in modo specifico la

gestione dei traumi). Inoltre, il sistema dovrà essere utilizzato in situazioni di emergenza e, a seconda dei casi, in luoghi differenti e spesso ostili.

Per via delle loro dimensioni ridotte, i dispositivi mobile risultano essere molto adatti alle esigenze del progetto. Essi mettono inoltre a disposizione una notevole capacità computazionale, che nella realizzazione del sistema può essere sfruttata per implementare le funzionalità più varie.

All'interno del progetto PreH Soccorso, il mio contributo consiste nella realizzazione di un'applicazione mobile che consenta di raccogliere, gestire e inviare dati nel corso di una missione di soccorso. Questa tesi di laurea si pone quindi l'obiettivo di progettare e sviluppare il sistema mobile descritto. Nell'elaborato presentato sono state descritte le fasi che hanno portato allo sviluppo dell'applicazione, e sono stati inseriti i seguenti capitoli:

1. Nel primo capitolo, viene fatta una breve panoramica sull'uso delle tecnologie informatiche in ambito ospedaliero e viene presentato Trauma-Tracker. Relativamente a questo, sono state evidenziate la struttura e le funzionalità dell'attuale sistema. Infine è stata introdotta e motivata la necessità di realizzare un sistema per il tracciamento preospedaliero.

2. Nel secondo capitolo è stato presentato il progetto PreH Soccorso, descrivendo gli obiettivi da realizzare e chiarendo il mio contributo. Facendo riferimento all'applicazione mobile da realizzare, è stata riportata la fase di analisi con cui sono stati definiti i requisiti del sistema.

3. Il terzo capitolo è stato dedicato a un approfondimento tecnologico sui dispositivi mobile e, in particolare, sullo sviluppo cross-platform. In base ai requisiti definiti, questo approccio è stato considerato il più adatto al progetto e sono stati quindi valutati alcuni framework cross-platform per individuarne uno da usare nello sviluppo dell'applicazione.

4. Nel quarto capitolo, sono state descritte le fasi di progettazione e sviluppo che hanno portato alla realizzazione del sistema. In particolare sono stati presentati la tecnologia scelta, i modelli prodotti nella fase di progettazione, l'implementazione svolta e il risultato ottenuto.

5. Nel quinto capitolo, sono state riportate le attività svolte con l'obiettivo di validare e discutere il lavoro fatto.

6. L'appendice finale, inserita per terminare l'elaborato, è dedicata alle conclusioni generali e ai possibili sviluppi futuri.

Capitolo 1

Tecnologie informatiche per il tracciamento dei traumi: il progetto TraumaTracker

In questo primo capitolo dell'elaborato sarà descritto il contesto applicativo in cui si colloca il progetto presentato con questa tesi di laurea.

Inizialmente sarà fatta una panoramica sull'uso di tecnologie, e in particolar modo di quelle informatiche, in ambito ospedaliero (1.1). Successivamente sarà presentato TraumaTracker, un progetto nato con l'obiettivo di realizzare un sistema informatico per il tracciamento dei traumi. In particolare, saranno analizzate la struttura e le funzionalità di TraumaTracker (1.2).

Infine sarà motivata la necessità di realizzare un sistema, esterno a TraumaTracker ma fortemente legato ad esso, che permetta il tracciamento di dati nell'ambiente preospedaliero (1.3). Da questa necessità scaturisce l'obiettivo della tesi presentata, che sarà poi approfondito nel capitolo 2.

1.1 Tecnologie ICT in ambito ospedaliero

Nei paesi più sviluppati si sta vivendo oggi quella che viene definita la quarta rivoluzione industriale poiché, grazie all'evoluzione tecnologica, i processi di produzione stanno diventando sempre più automatizzati e interconnessi. Si parla infatti di Industria 4.0, un nuovo modo di concepire la produzione industriale con cui viene introdotto il concetto di smart factory[1] [1].

Lo sviluppo che va verso l'Industria 4.0 prevede dei sistemi basati su Pervasive Computing, ossia la crescente tendenza di incorporare la capacità com-

[1]**Smart factory**: Letteralmente fabbrica intelligente, indica un'industria che è digitalizzata e interconnessa (grazie alla rete).

putazionale in oggetti di uso quotidiano per fare in modo che siano in grado di svolgere delle attività in modo efficace, e su IoT[2], cioé l'nterconnessione di dispositivi informatici incorporati in oggetti di uso quotidiano (le cosiddette "things", cose) che permette loro di comunicare e cooperare in real time.

Come evidenziato dalla figura 1.1, l'Industria 4.0 beneficia dei progressi tecnologici che si sono verificati in diversi campi, come ad esempio big data[3], cloud computing[4], sistemi cyber-physical[5], realtà aumentata[6] e robotica.

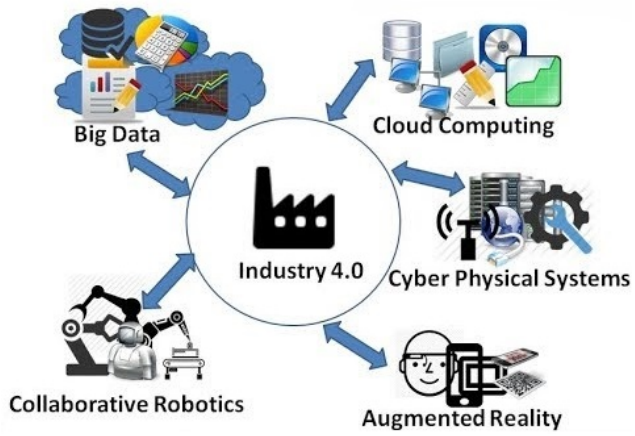

Figura 1.1: Tecnologie a supporto dell'industria 4.0 [1]

Le innovazioni portate dall'Industria 4.0 non stanno rivoluzionando soltanto l'industria manifatturiera, ma hanno applicazioni in tutti i campi della società attuale. Perciò, questa rivoluzione non poteva non investire anche l'azienda sanitaria, un mondo che in Italia conta più di un milione di lavoratori e, almeno "virtualmente", sessanta milioni di clienti [2].

Con la quarta rivoluzione industriale le strutture ospedaliere sono cambiate e cambieranno ulteriormente, tanto che si è iniziato a parlare di Ospedale 4.0. Questo concetto introduce un nuovo modo di concepire l'ospedale, poiché questo non viene più visto come "edificio" ma come "insieme di servizi" (accessibili sia nell'ambiente ospedaliero sia all'esterno). L'Ospedale 4.0 è il luogo dove discipline profondamente diverse come medicina, informatica e ingegne-

[2]**IoT**: Internet of Things, letteralmente internet delle cose.

[3]**Big data**: Insieme delle tecnologie e delle metodologie per l'analisi di dati massivi.

[4]**Cloud computing**: Paradigma di erogazione di risorse informatiche, caratterizzato dalla possibilità di accedere a servizi (come ad esempio memorizzazione e elaborazione di informazioni) attraverso la rete.

[5]**Sistemi cyber-physical**: Sistemi che integrano la computazione con processi fisici.

[6]**Realtà aumentata**: Arricchimento della percezione sensoriale mediante informazioni, in genere manipolate e convogliate elettronicamente, altrimenti non percepibili.

ria clinica collaborerranno per garantire ai pazienti il miglior servizio possibile [2, 3]. Verranno usati molti strumenti tecnologici, come ad esempio:

- Sensori e dispositivi medici per il monitoraggio dei pazienti, molti dei quali collocati all'interno di un ambiente domestico (in modo che i pazienti possano ricevere assistenza direttamente da casa).
- Sistemi mobili e pervasivi per il tracciamento e l'accesso a informazioni, tra cui troviamo anche dispositivi wearable[7] e eyewear (figura 1.2).
- Strumenti per consentire ai medici l'accesso remoto a dati clinici.
- Occhiali per la realtà aumentata, proiezioni di immagini in realtà virtuale, stampanti 3D e altri strumenti per aumentare le informazioni percepite dal medico e rendere così le cure più efficienti.
- Strumenti di cloud computing per l'archiviazione remota dei dati.
- Sistemi che utilizzano le tecniche di intelligenza artificiale[8] per supportare i medici che devono prendere decisioni.
- Robot per fornire ai medici un ausilio in ambito chirurgico.

Figura 1.2: Smart-glasses, un dispositivo eyewear, usati da un medico [3]

I dispositivi e i sistemi appena descritti hanno le potenzialità per cambiare radicalmente la concezione che si ha del mondo ospedaliero grazie all'IoT,

[7]**Dispositivi wearable**: Sistemi embedded indossabili, che costituiscono validi strumenti per la comunicazione hands-free e per la realtà aumentata. Tra questi troviamo i dispositivi eyewear, come ad esempio gli smart-glasses.

[8]**Intelligenza artificiale**: Disciplina che studia metodologie e tecniche per progettare sistemi hardware e software capaci di fornire a un elaboratore elettronico capacità che, apparentemente, sembrerebbero essere di pertinenza esclusiva dell'intelligenza umana.

poiché quest'ultima dà la possibilità di gestire e monitorare tutte queste entità e di creare così un ecosistema completo. Si prevede che, grazie a sistemi basati sull'IoT che permettano l'automatizzazione di attività attualmente svolte dal personale umano, ci saranno benefici sia medici, in termini di miglioramento della qualità delle cure, sia economici, in termini di riduzione dei costi.

Come è stato visto il fulcro di questo processo di cambiamento è costituito dalle tecnologie ICT[9], che si occupano della gestione delle informazioni, e di seguito sarà descritto TraumaTracker, un progetto frutto dell'applicazione di tecnologie ICT all'interno del contesto ospedaliero.

1.2 Il progetto TraumaTracker

TraumaTracker è un progetto in fase di sviluppo, nato dalla collaborazione tra un gruppo di ricerca del Dipartimento di Informatica - Scienza e Ingegneria (DISI) dell'Università di Bologna e l'unità operativa di anestesia e rianimazione del Dipartimento Chirurgico e Grandi Traumi Cesena - Ausl Romagna per aiutare i medici impegnati, all'interno dell'ospedale, nella gestione dei traumi.

1.2.1 Descrizione e obiettivo

Il progetto TraumaTracker si colloca in un dominio applicativo estremamente difficile da gestire, ossia all'interno del Trauma Center[10] di una struttura ospedaliera. Questo è uno degli ambienti medici più critici e impegnativi, poiché richiede che il personale sanitario prenda decisioni in modo corretto, rapido e reattivo. In un contesto di questo tipo, ci sono diverse problematiche da affrontare e possono essere riassunte in quattro punti:

1. Avere una documentazione del trauma accurata è fondamentale per migliorare la qualità della cura[11].

2. I medici devono monitorare una grande quantità di parametri vitali e di fattori collegati alla sicurezza del paziente, e in base a questi prendere una decisione sulla prossima operazione da eseguire.

3. La sequenza frenetica degli eventi che avvengono durante un trauma lascia poco tempo ai medici per ragionare su quale possa essere il trattamento migliore da eseguire sul paziente.

[9]**ICT**: Information and comunication technology, indica l'insieme delle tecnologie che forniscono l'accesso alle informazioni attraverso le telecomunicazioni.

[10]**Trauma Center**: Reparto ospedaliero dedicato alla gestione dei traumi.

[11]**Qualità della cura**: Nell'ambito di un trauma, si tratta del raggiungimento del miglior risultato possibile (in base alle circostanze di partenza).

4. Il cordinamento tra i vari attori (medici, infermieri, ...) coinvolti nella gestione del trauma è fondamentale per raggiungere il miglior risultato.

Il progetto TraumaTracker nasce per realizzare un sistema che affronti le prime due problematiche citate, e fornisca un valido ausilio al personale medico.

Per quanto riguarda il primo punto, attualmente negli ospedali italiani per produrre la documentazione di un trauma il Trauma Leader[12] è chiamato a riportare gli eventi più significativi della fase di rianimazione dopo essere stato impegnato in essa. La documentazione avviene quindi dalla memoria e non in tempo reale e, inoltre, i dati vengono informatizzati soltanto in un secondo momento e devono essere trascritti. Tutto ciò fa sì che la documentazione sia spesso imprecisa e carente, e a volte anche errata. Da qui, emerge l'utilità di uno strumento in grado di automatizzare la documentazione di un trauma.

Per quanto riguarda il secondo punto, ovviamente le decisioni sui trattamenti da eseguire su un paziente spettano ai medici del Trauma Team[13]. Questi però, dato il contesto di emergenza, potrebbero essere notevolmente aiutati da suggerimenti generati automaticamente in base allo stato clinico del paziente.

Partendo da queste considerazioni, il progetto TraumaTracker si pone l'obiettivo di tenere traccia degli eventi che si verificano durante la gestione di un trauma e delle informazioni rilevanti relative ad esso e lo persegue per raggiungere due scopi: il tracciamento e l'assistenza. Si parla infatti di due differenti livelli di supporto, che saranno introdotti di seguito.

Il livello di tracciamento è il livello base di TraumaTracker, nonché quello principale. A breve termine, infatti, l'obiettivo del progetto è realizzare un tracciamento sistematico e il più fedele possibile di tutti i traumi gestiti nel Trauma Center. Questo implica un aumento sia della qualità sia della quantità dei dati raccolti e la produzione di una documentazione del trauma accurata e automatizzata, che sarà strutturata in report e che sarà utile per valutare (e eventualmente migliorare) il lavoro del TraumaTeam.

A medio termine, il progetto si pone anche l'obiettivo di introdurre diversi tipi di assistenza per sostenere il TraumaTeam nella gestione di un trauma e si parla infatti di livello di assistenza. Tale assistenza, che potrà essere espressa in termini di generazione in tempo reale di avvertimenti e suggerimenti, sarà prodotta valutando gli eventi e i dati precedentemente tracciati [4, 5].

1.2.2 Architettura del sistema

Il progetto TraumaTracker si basa sull'idea di PMDA[14], ossia un'estensione dell'agente software per l'assistenza personale. In questo progetto sono

[12] **Trauma Leader**: Medico che ha il compito di supervisionare il Trauma Team.
[13] **Trauma Team**: Gruppo di medici impegnati nella gestione di un trauma.
[14] **PMDA**: Personal Medical Digital Assistant Agent.

richieste particolari caratteristiche al PMDA, perché è necessaria l'osservazione continua dello stato dinamico del contesto in cui agisce il medico e di ciò che egli sta facendo, e perché l'assistenza deve essere fornita mentre i medici svolgono la loro attività senza interromperli o disturbarli.

Da un punto di vista tecnologico i PMDA sfruttano i dispositivi mobile, ma beneficiano anche della disponibilità di tecnologie wearable e eyewear (nel caso di TraumaTracker, sono utilizzati degli smart-glasses).

Nella figura 1.3 inserita di seguito è riportato un diagramma che illustra l'architettura a grana grossa[15] di TraumaTracker.

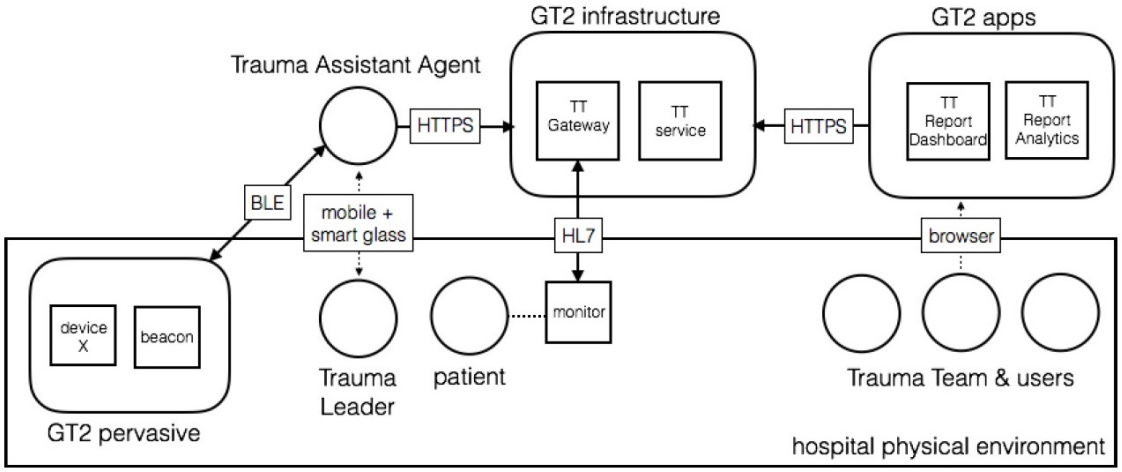

Figura 1.3: Architettura a grana grossa di TraumaTracker [4]

L'archutettura appena mostrata è orientata agli agenti[16] e ai servizi[17], ed è caratterizzata da quattro componenti principali:

- Il PMDA, chiamato *Trauma Assistant Agent*, che è in esecuzione su dispositivi mobile (tablet) e wearable (smart-glasses) e che viene utilizzato dal Trauma Leader.
- Un insieme di servizi pervasivi, messi a disposizione da servizi distribuiti nell'ambiente fisico dell'ospedale, indicati come *GT2 pervasive*. Attualmente il *GT2 pervasive* include una serie di beacon posizionati in

[15] **Architettura a grana grossa**: Architettura software che divide il sistema in un numero, relativamente piccolo, di elementi.

[16] **Architettura orientata agli agenti**: Architettura software per la progettazione e la realizzazione di sistemi concorrenti e distribuiti, nata come un'evoluzione concettuale di architetture ad oggetti e di architetture basate su attori.

[17] **Architettura orientata ai servizi**: Architettura software adatta a supportare l'uso di servizi web per garantire l'interoperabilità tra diversi sistemi, così da consentire l'uso delle singole applicazioni in un'ottica molto più ampia.

tutte le stanze, coinvolti nella gestione di un trauma poiché abilitano la localizzazione a livello di camera per il PMDA.

- Una serie di servizi web distribuiti nella rete locale dell'ospedale, a cui si fa riferimento parlando di *GT2 infrastructure*. Questi supportano le componenti del sistema sopraelenncate e attualmente comprendono: *TT Report Management Service*, che fornisce delle API REST-ful[18] per la raccolta e la gestione dei report sui traumi e per l'accesso ai relativi dati statistici, e *TT Vital Signs Monitor Service*, che mette a disposizione delle API per il recupero dinamico dei parametri vitali di un paziente e offre la possibilità di realizzare un monitoraggio continuo.

  I dati raccolti da questi servizi sono resi disponibili anche ad altre strutture ospedaliere tramite applicazioni in esecuzione sulla stessa infrastruttura, nella prospettiva di un ecosistema aperto.

- Una serie di applicazioni web, a cui si fa riferimento con il termine *GT2 apps*, che permettono agli utenti di accedere ad alcuni dei servizi che fanno parte della *GT2 infrastructure*. Attualmente nel sistema è presente il *TT Report Dashboard*, a cui accedono i membri del Trauma Team.

1.2.3 Funzionalità del sistema

Come detto in precedenza, TraumaTracker si propone di realizzare due differenti livelli di supporto: quello di tracciamento e quello di assistenza. Sarà fatta ora una panoramica delle principali funzionalità che il sistema attuale mette a disposizione, descrivendole senza però scendere nei dettagli relativi a come queste funzionalità sono state implementate [4, 5].

Livello di tracciamento

La maggior parte delle funzionalità di TraumaTracker appartengono al livello di tracciamento, che è già stato introdotto in precedenza. In particolare:

- Il sistema è in grado di tenere traccia delle azioni eseguite dal Trauma Team, che possono essere sia procedure effettuate (come ad esempio applicazione di un laccio emostatico o drenaggio toracico) sia farmaci o emoderivati somministrati (come ad esempio cristalloidi, soluzione ipertonica, adrenalina o atropina). Per le azioni dipendenti dal tempo, il sistema è in grado di memorizzare anche la durata.

[18] **API REST-ful**: Interfaccia del programma applicativo che utilizza le richieste HTTP per le operazioni GET, PUT, POST e DELETE.

- Il sistema consente al Trauma Leader di scattare istantanee e registrare video o audio da includere nel report del trauma, utilizzando la camera equipaggiata agli smart-glasses.

- Il tracciamento di eventi e note avviene reagendo ai comandi che sono esplicitamente richiesti dal Trauma Leader, sfruttando l'interfaccia utente fornita dal dispositivo mobile su cui è in esecuzione il PMDA. Questa interfaccia è stata progettata per ridurre al minimo le interazioni richieste per specificare l'azione eseguita.

- Il sistema permette di recuperare, visualizzare e tracciare il valore corrente dei parametri vitali del paziente interagendo direttamente con il *TT Vital Signs Monitor Service*, un sistema che attraverso sensori connessi al paziente riesce a monitorare in tempo reale i suoi parametri. Questi dati devono essere recuperati e memorizzati automaticamente sia quando viene eseguita una certa azione (procedura o farmaco) sia periodicamente, con un periodo che dipende dalla posizione specifica in cui si trova il paziente (ad esempio, se il paziente si trova nella shock-room i parametri saranno monitorati molto frequentemente).

- Ogni informazione tracciata dal sistema comprende un dato temporale (data e ora) e un dato spaziale (stanza specifica in cui il paziente si trova).

- Una volta completata la gestione del trauma e annotata la destinazione finale del paziente, il report completo del trauma (che comprende anche foto, video e note locali) viene inviato e archiviato in modo automatico su un server, sfruttando la rete Wi-Fi dell'ospedale.

- Il sistema dà la possibilità di accedere, gestire, stampare ed esportare la documentazione (report) relativa ad uno o più traumi.

Livello di assistenza

Il livello di assistenza è costruito sopra al livello di tracciamento, poiché è basato sull'osservazione degli eventi tracciati e sulla storia del trauma.

Questo livello produce degli avvertimenti (warning), che sono percepiti direttamente dal Trauma Leader attraverso il PMDA e che sono prodotti attraverso regole definite in base al dominio applicativo (e quindi grazie al confronto con i medici). Una determinata regola prevede una condizione e un warning: al verificarsi di quella determinata condizione viene generato il warning.

Le regole si dividono in tempo indipendenti, se prevedono una condizione che riguarda lo stato attuale del trauma, e tempo dipendenti, se la loro condizione include controlli temporali sull'evoluzione dello stato del trauma.

1.3 Un sistema per il tracciamento preospedaliero

Tutti i medici del Trauma Center dell'ospedale Bufalini di Cesena che sono stati coinvolti nella valutazione del sistema TraumaTracker hanno concordato sulla sua utilità. Inoltre, dopo aver toccato con mano i vantaggi portati da un sistema informatico che consente il tracciamento nell'ambiente ospedaliero, i medici hanno fatto emergere un'ulteriore necessità: realizzare un sistema che renda possibile il tracciamento nell'ambiente preospedaliero.

Il sistema TraumaTracker, infatti, entra in funzione quando il paziente raggiunge l'ospedale trasportato dal mezzo di soccorso (ambulanza, elicottero, automedica, ...). Vi è però una fase precedente all'arrivo in ospedale, ossia quella che viene definita "preospedaliera", che è cruciale per la sopravvivenza del paziente e che attualmente viene documentata soltanto manualmente (compilazione di schede cartacee con una struttura predefinita).

Introdurre un sistema per il tracciamento preospedaliero porterebbe gli stessi vantaggi avuti grazie al livello di tracciamento di TraumaTracker, ossia una documentazione più accurata del trauma e quindi una migliore qualità delle cure prestate (sia nell'ambiente preospedaliero che in quello ospedaliero).

Per questo motivo il team universitario guidato dal professor Alessandro Ricci ha deciso di portare avanti la realizzazione di questo sistema, che non dovrà essere un'estensione di TraumaTracker ma sarà autonomo (perché verrà utilizzato in un momento precedente). Inoltre esso dovrà consentire il tracciamento della fase preospedaliera di avvenimenti generici e, a differenza di TraumaTracker, non tratterà in modo specifico la gestione dei traumi.

Ho avuto il privilegio di essere coinvolto in questo progetto, estremamente significativo per via del suo contesto applicativo, e con questa tesi mi sono posto l'obiettivo di realizzare una parte del sistema descritto (vedi 2.1). Nei prossimi capitoli verranno illustrate, in accordo con gli standard definiti dall'ingegneria del software[19], le principali fasi che hanno portato alla sua realizzazione.

[19]**Ingegneria del software**: *"Corpus di teorie, metodi e strumenti, sia di tipo tecnologico che organizzativo, che consentono di produrre applicazioni con le desiderate caratteristiche di qualità."* [6]

Capitolo 2

Il progetto PreH Soccorso

Dopo aver messo in evidenza quanto ottenuto grazie a TraumaTracker e aver introdotto la necessità di un sistema per il tracciamento preospedaliero (1), in questo capitolo sarà presentato il progetto PreH Soccorso.

In primo luogo, saranno descritte le motivazioni del progetto e i suoi obiettivi. Sarà prestata particolare attenzione a come viene attualmente tracciata la fase di soccorso preospedaliera, a quali sono le problematiche che i medici devono affrontare e a cosa si vuole fare per risolverle (2.1). Successivamente, verrà descritto il mio contributo nel progetto PreH Soccorso e sarà riportata la fase di analisi dei requisiti relativa al sistema software realizzato (2.2).

2.1 Descrizione e obiettivi

Il dominio applicativo in cui si colloca il progetto PreH Soccorso è, proprio come quello di TraumaTracker, estremamente critico da gestire. In aggiunta alle difficoltà già descritte per le operazioni di soccorso ospedaliere (necessità di intervenire in situazioni di emergenza, necessità di prendere decisioni in tempi molto brevi, ...), nella fase di soccorso che precede l'arrivo in ospedale i medici sono costretti ad operare in luoghi e condizioni spesso ostili.

Parlando di TraumaTracker (1.2), è stato evidenziato come una documentazione accurata sia fondamentale per migliorare la qualità delle cure. Com'è facile intuire, questo concetto vale anche per la fase di soccorso preospedaliera: tenere traccia di quanto è accaduto e delle operazioni fatte prima dell'arrivo in ospedale è fondamentale per prendere decisioni sui trattamenti a cui sottoporre il paziente (sia nella fase preospedaliera che in quella ospedaliera).

Dal confronto con i medici dell'ospedale Bufalini di Cesena è emerso che attualmente, quando sono impegnati in un intervento di soccorso (con ambulanza, elicottero, ...), hanno il compito di documentare la fase preospedaliera compilando una scheda medica cartacea per ogni paziente trasportato in ospe-

dale (nelle figure 2.1 e 2.2 è riportata la scheda cartacea dell'AREU della regione Lombardia, che mostra un esempio dei dati attualmente raccolti nel preospedaliero). Inoltre, a volte i medici contattano telefonicamente la centrale operativa e comunicano alcune informazioni riguardanti l'intervento. Questo modus operandi implica due problematiche non affatto trascurabili:

- Nella maggior parte dei casi, i medici in servizio nella struttura ospedaliera non hanno nessuna informazione sui pazienti che si preparano ad accogliere. Nelle rare situazioni in cui hanno a disposizione alcuni dati, questi sono stati comunicati loro dalla centrale operativa, a sua volta contattata dai medici impegnati nel soccorso. Le informazioni ottenute in questo modo sono sommarie, per la durata limitata della chiamata, e spesso inesatte, a causa del "passaparola" appena descritto.

  L'assenza di informazioni dettagliate nei momenti che precedono l'arrivo in ospedale è un aspetto estremamente negativo, poiché i medici non possono prepararsi ad eventuali interventi (ad esempio, aprendo una sala operatoria) e sono costretti a decidere come comportarsi dopo aver ricevuto il paziente. Questo causa la perdita di alcuni minuti, che possono essere cruciali per la sopravvivenza del paziente.

- I dati inseriti dal medico impegnato nel soccorso attraverso la compilazione della scheda sono soggetti ai suoi possibili errori. Inoltre, per essere poi informatizzati e memorizzati, devono essere trascritti (solitamente da un altro medico) e questo comporta altri possibili errori.

Per risolvere le questioni illustrate è nata l'idea del progetto PreH Soccorso, che si pone l'obiettivo di realizzare un sistema per il tracciamento preospedaliero. Questo dovrà sotituire la scheda cartacea attualmente utilizzata e permettere al medico impegnato nel soccorso di comunicare informazioni relative all'intervento e allo stato clinico dei pazienti, cercando di automatizzare e migliorare quanto più possibile la raccolta di questi dati. Essi saranno integrati con informazioni che attualmente non possono essere raccolte, come ad esempio documenti multimediali (foto, video, ...).

Il sistema dovrà essere composto da due parti: un'applicazione eseguita su dispositivi mobile (in questo contesto applicativo preferibili rispetto ai tablet, poiché meno ingombranti), che sarà utilizzata dai medici impegnati nella fase di soccorso e che memorizzerà informazioni relative al loro intervento, e un servizio web, che sarà in grado di ottenere i dati raccolti dal sottosistema mobile, conservarli e permettere l'accesso remoto ad essi.

All'interno del progetto PreH Soccorso, il mio contributo consiste nella realizzazione dell'applicazione mobile descritta. Di seguito sarà presentata la fase di analisi dei requisiti relativa a questo sistema mobile.

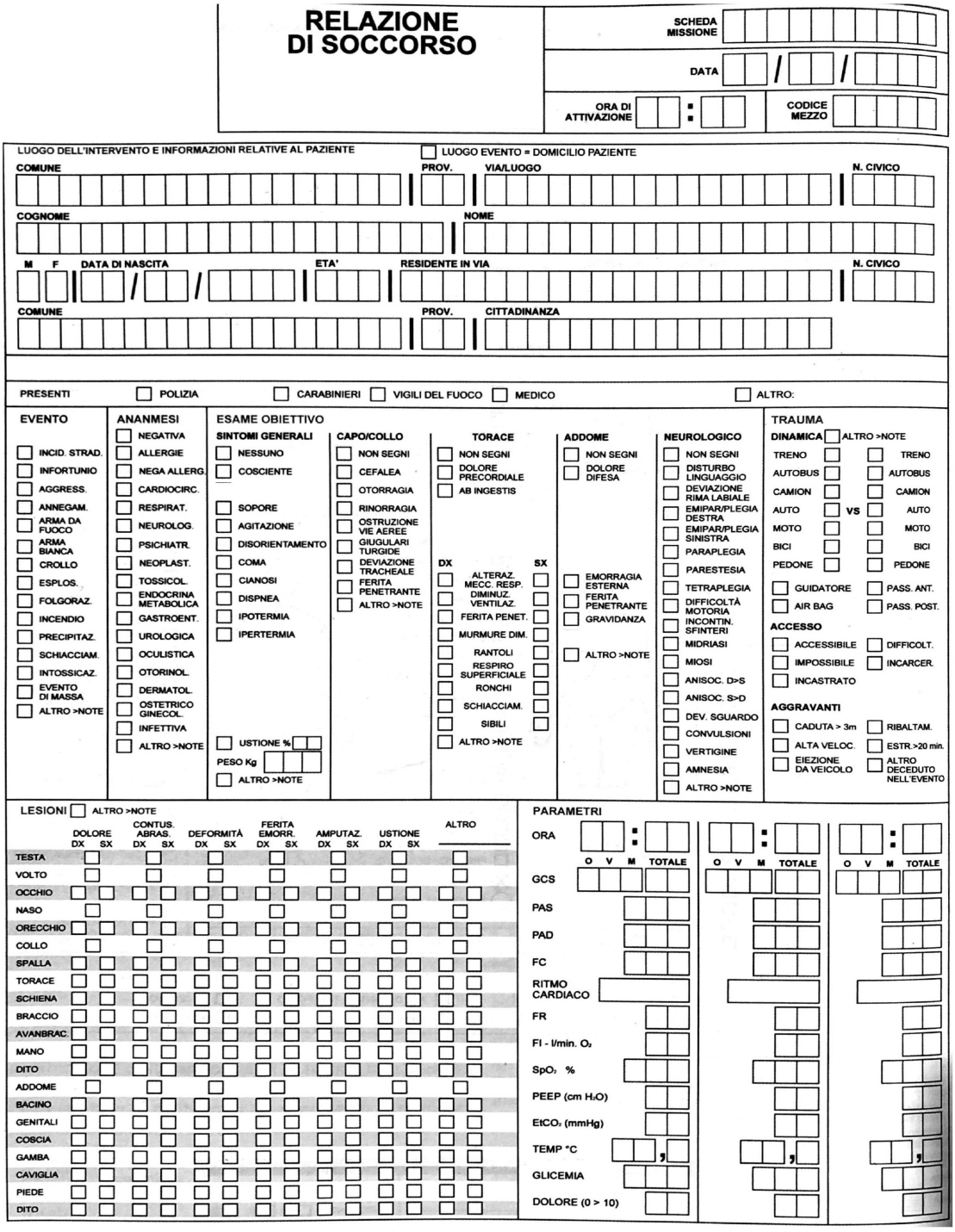

Figura 2.1: Esempio di scheda medica cartacea utilizzata nel PreH

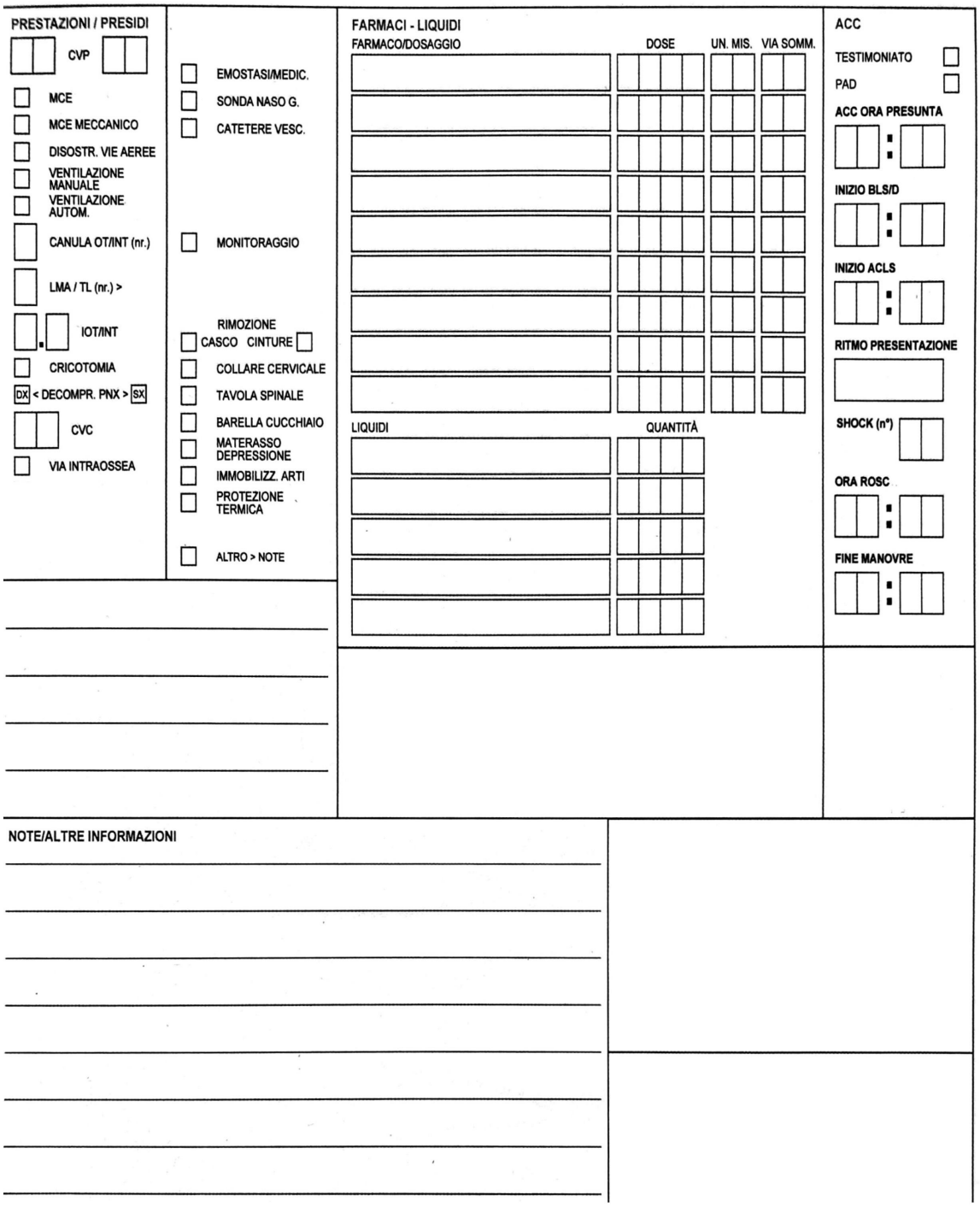

Figura 2.2: Esempio di scheda medica cartacea utilizzata nel PreH

2.2 Analisi dei requisiti

L'analisi dei requisiti è una fase fondamentale nello sviluppo software, poiché ha lo scopo di definire le caratteristiche del sistema da realizzare (in termini di requisiti che devono essere soddisfatti). Al termine di questa sarà prodotto il documento di specifica dei requisiti, un accordo tra produttore e consumatore che costituisce l'input per le fasi di sviluppo successive [7].

Data la complessità del dominio, i requisiti sono stati formalizzati solo dopo un confronto con quelli che saranno gli utilizzatori del sistema (ho incontrato personalmente il dottor Vittorio Albarello, dirigente medico presso il reparto di Anestesia e Rianimazione dell'ospedale Bufalini di Cesena).

Prima dei requisiti, sarà riportato un glossario che avrà il compito di definire il significato di alcuni termini importanti e ricorrenti nel dominio applicativo. Dopodiché saranno definiti i requisiti del sistema mobile da realizzare, suddivisi in funzionali, che descrivono le funzionalità che dovranno essere offerte, e non funzionali, che sono vincoli sul sistema o sul suo processo di sviluppo.

2.2.1 Glossario

Termine	**Significato**
Medico	Utente che è autorizzato a usare il sistema.
Medico attivo	Medico che ha attivato una missione, è impegnato nel soccorso ed è responsabile del tracciamento.
Missione	Intervento di soccorso, di cui è responsabile un medico e in cui sono coinvolti uno o più pazienti.
Scheda medica digitale	Documento digitale che contiene le informazioni relative a un determinato paziente nell'ambito di una missione.
Centrale operativa	Luogo in cui vengono ricevute telefonate dei cittadini e da cui partono le richieste di intervento.
Luogo dell'evento	Luogo in cui si è verificato l'avvenimento, a causa del quale è necessaria una missione.
Stato di una missione	Attuale situazione della missione. Stati possibili: Missione attivata, Soccorsi partiti verso il luogo dell'evento, Soccorsi arrivati sul luogo dell'evento, Soccorsi ripartiti dal luogo dell'evento, Soccorsi rientrati in ospedale e Missione Conclusa.
Cambiamento di stato di una missione	Cambiamenti di stato possibili: Attivazione della missione, Partenza dei soccorsi, Arrivo sul luogo dell'evento, Partenza dal luogo dell'evento, Rientro in ospedale e Conclusione della missione.

Dispatch di intervento	Richiesta di intervento ricevuta dai medici e fatta dalla centrale operativa, che invia un codice di chiamata tramite sms.
Attivazione di una missione	Momento iniziale di una missione, in cui un medico inserisce nel sistema un codice ricevuto con il dispatch e attiva una nuova missione.
Conclusione di una missione	Momento finale di una missione, successivo al rientro in ospedale, in cui il medico attivo completa l'inserimento delle informazioni raccolte.
Aborto missione	Cancellazione di una missione da parte del medico attivo, avvenuta per un particolare motivo.

2.2.2 Requisiti funzionali

Il sistema dovrà consentire al medico di effettuare le seguenti operazioni:

- Identificazione e riconoscimento come attuale utente del sistema (considerando che, in momenti diversi, questo può essere usato da più medici).
- In seguito all'identificazione, attivazione di una nuova missione con l'inserimento del codice ricevuto con il dispatch di intervento.
- Dopo l'attivazione di una missione, cambiamento del suo stato con memorizzazione automatica del tempo (data e ora) e della posizione (coordinate geografiche) in cui esso avviene.
- Dopo l'attivazione di una missione, aborto della missione con l'obbligo di specificare una motivazione.
- Inserimento di informazioni relative alla dinamica dell'avvenimento che è stato la causa dell'intervento di soccorso.
- Gestione parallela di più pazienti nell'ambito della stessa missione con inserimento di informazioni, sia generali sia mediche, relative ad essi (anagrafica, lesioni e ustioni riportate, anamnesi e esame obiettivo).
- Tracciamento di tutto ciò che viene fatto o avviene ad un paziente (misurazione di parametri vitali, prestazione effettuata, somministrazione di farmaci o liquidi e eventuale arresto cardiocircolatorio), con memorizzazione automatica del tempo (data e ora).
- Documentazione multimediale della dinamica e dei pazienti coinvolti attraverso fotografie scattate con la camera del dispositivo su cui è in esecuzione il sistema e, eventualmente, video registrati con una wearable camera collegata al sistema e indossata dal medico.

- Esportazione in formato digitale di una o più schede mediche (una per paziente) relative ad una missione, che sostituiranno quelle cartacee.
- Utilizzo del dispositivo mobile per contattare telefonicamente alcuni dipartimenti dei principali ospedali della zona.

Inoltre, il sistema dovrà comunicare con il servizio web (presumibilmente in esecuzione all'interno della centrale operativa) a cui verranno inviate tutte le informazioni raccolte relativamente a una determinata missione. Per rappresentare i requisiti funzionali è stato prodotto un diagramma dei casi d'uso [1] (figura 2.3), in cui gli attori individuati sono il medico e il servizio web.

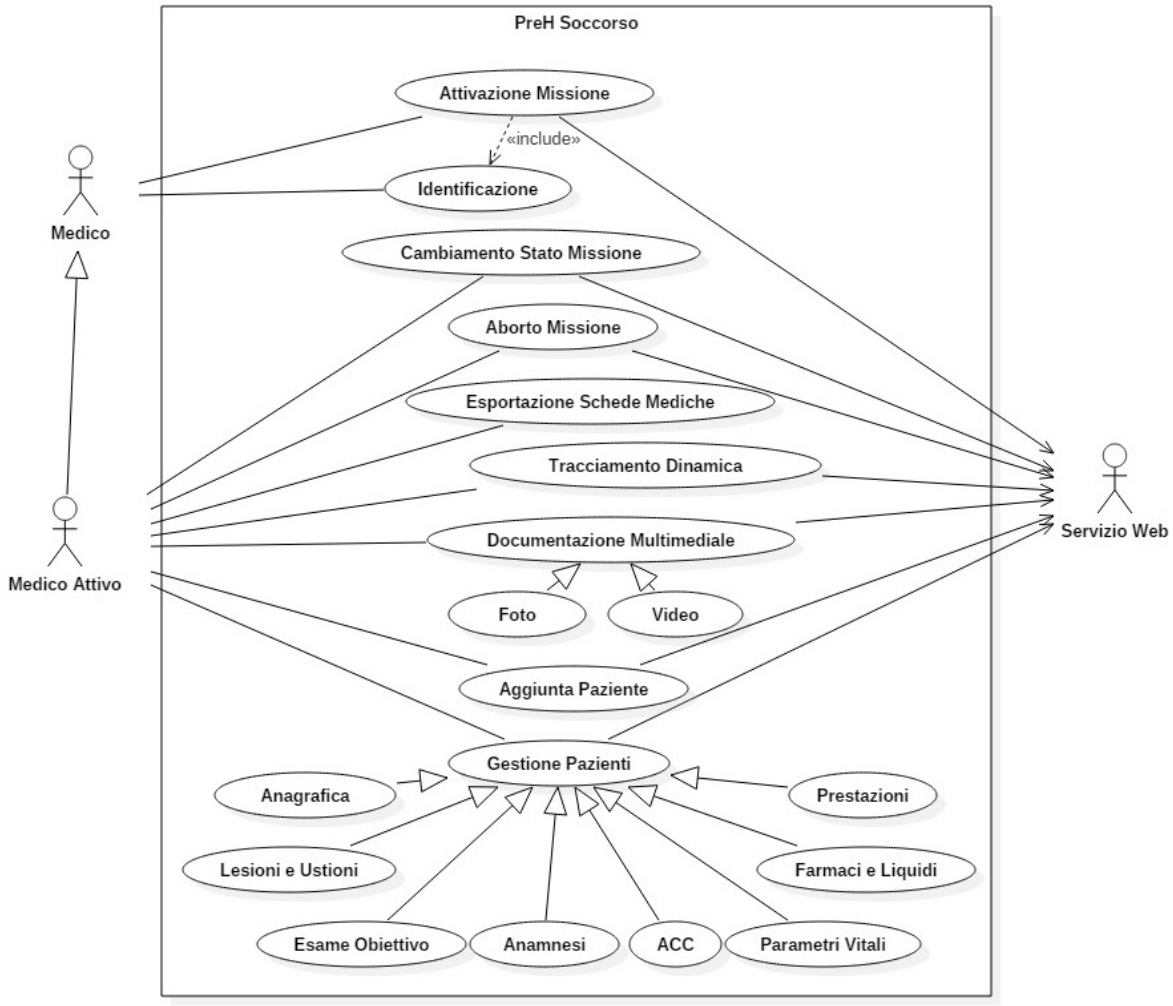

Figura 2.3: Diagramma dei casi d'uso relativo al sistema mobile PreH Soccorso

[1] **Diagramma dei casi d'uso**: In UML, è un diagramma che descrive l'interazione tra gli attori (utilizzatori del sistema) e il sistema stesso.

2.2.3 Requisiti non funzionali

Per i requisiti non funzionali, sono state seguite le linee guida descritte da ISO/IEC 9126[2] e sono state individuate queste caratteristiche di qualità:

- **Funzionalità**
  Dovranno essere soddisfatti i requisiti funzionali e dovrà essere prestata attenzione a qualità fondamentali come interoperabilità (capacità di cooperare con altri sistemi, e in particolare con il servizio web) e sicurezza (capacità di evitare violazioni dei dati dei pazienti).

- **Affidabilità**
  L'utente dovrà poter dipendere dal software, che dovrà produrre risultati corretti in ogni situazione. Perciò l'applicazione dovrà essere robusta e comportarsi in modo ragionevole anche in circostanze non previste. Questa caratteristica è particolarmente importante perché il sistema dovrà operare in un contesto di emergenza. Ad esempio, dovrà essere memorizzata la missione corrente per poterla ripristinare nel caso di un evento inatteso (come la chiusura accidentale dell'applicazione).

- **Usabilità**
  Il sistema dovrà poter essere utilizzato in modo semplice e immediato. L'interfaccia utente dovrà permettere un'interazione il più rapida possibile, in modo da facilitare i medici impegnati nelle operazioni di soccorso ed evitare di creare loro ulteriori difficoltà. Per gli utenti dovrà essere molto semplice comprendere e imparare il funzionamento del sistema.

- **Efficienza**
  Il software dovrà essere efficiente in termini di utilizzo delle risorse a disposizione e di velocità nell'esecuzione delle operazioni richieste.

- **Manutenibilità**
  Il sistema dovrà essere realizzato in modo da facilitarne la sua manutenzione e da poter effettuare facilmente delle modifiche, se necessarie. Affinché questo sia possibile il software dovrà essere caratterizzato da modularità, e quindi composto da un insieme di moduli[3].

- **Portabilità**
  L'applicazione dovrà poter funzionare in più ambienti di lavoro. In particolare, trattandosi di un sistema destinato a dispositivi mobile, dovrà poter essere eseguito sulle piattaforme mobile più diffuse.

[2] **ISO/IEC 9126**: Serie di normative che indicano un modello di qualità del software [8].
[3] **Modulo**: Componente di base di un sistema con funzionalità strettamente legate.

Capitolo 3

L'universo mobile e lo sviluppo cross-platform

Dopo aver descritto il progetto PreH Soccorso e definito i requisti del sistema da realizzare (capitolo 2) è necessario un approfondimento tecnologico, che consentirà di scegliere quale tecnologia usare nello sviluppo.

In particolare in questo capitolo verranno trattati i dispositivi mobile (3.1), che per le loro caratteristiche sono in grado di venire incontro alle esigenze e ai requisiti del progetto, e saranno poi presentati i possibili approcci allo sviluppo mobile (3.2). Successivamente ci si concentrerà sullo sviluppo di applicazioni cross-platform, quelle ritenute più adatte a questo progetto (in base ai requisiti), e saranno approfonditi alcuni tra i principali framework che permettono di sviluppare questo tipo di applicazioni (3.3). Infine, sulla base di quanto emerso, verrà fatta una valutazione dei framework analizzati (3.4).

3.1 Dispositivi mobile

Il concetto di dispositivo mobile, che inizialmente si limitava a descrivere i telefoni cellulari, si è decisamente evoluto negli ultimi anni. Oggi *"per dispositivo mobile si intende qualsiasi dispositivo dotato di comunicazione wireless in grado di accedere alle funzioni di rete, come navigare sul web, consultare la posta elettronica e interagire con i social network, per esempio un telefono celluare, uno smartphone o un altro strumento (spesso multimediale)"* [9].

In informatica un altro concetto importante è quello di mobile computing, che *"designa in modo generico le tecnologie di elaborazione o accesso ai dati prive di vincoli sulla posizione fisica dell'utente o delle apparecchiature coinvolte"* [10] e nel quale si inseriscono le tecnologie che saranno descritte.

3.1.1 Storia

Il primo telefono mobile è stato progettato nel 1973, grazie alla collaborazione di Motorola e Bell Labs, e dieci anni dopo è comparso sul mercato il primo cellulare, un dispositivo che comunicava in modalità analogica e non possedeva ancora alcuna funzione per trasmettere contenuti testuali.

Nel 1992, con il passaggio a modalità digitali della rete di seconda generazione (2G), sono stati introdotti nuovi servizi come lo scambio di SMS[1] e attività ludiche di game-playing. La vera rivoluzione ci fu però nel biennio 2002-2003, durante il quale fu attivato in Europa il nuovo protocollo di terza generazione (3G), che permetteva ai dispositivi mobile di interagire con il web.

L'incremento delle prestazioni fu accompagnato da un aumento della qualità ergonomica, con l'introduzione di schermi grafici a colori, tastiere estese e fotocamere. Il mondo della telefonia s'integrava con quello dei PDA[2], per dare luogo a uno strumento più completo: lo smartphone[3].

Gli smartphone travolsero il mercato dei dispositivi mobile nel 2007, quando l'azienda statunitense Apple lanciò il primo modello dell'iPhone (figura 3.1). Questo nasceva dall'integrazione dell'asset musicale degli iPod con quello della telefonia e, attraverso uno schermo touch-screen, forniva all'utente un'esperienza di usabilità completamente innovativa per i dispositivi mobile.

Figura 3.1: Steve Jobs presenta la prima versione dell'iPhone (2007)

[1]**SMS**: Short Message Service, è un servizio per inviare brevi messaggi di testo.

[2]**PDA**: Personal Digital Assistant, più noto come palmare, è un dispositivo mobile che funziona come gestore di dati personali.

[3]**Smartphone**: Letteralmente "telefono intelligente", è un cellulare caratterizzato da un'ottima capacità di calcolo, di memoria e di connesione dati.

Successivamente, i dispositivi mobile hanno trovato il supporto di protocolli di comunicazione wireless quali Bluetooth, Wi-Fi[4] e GPS[5] e sono diventati sempre più polifunzionali, poiché in grado di adattarsi alle necessità specifiche dell'utente. Un fenomeno che ha avuto grandissimo successo è stato quello dell'interazione attraverso i social network, piattaforme che forniscono agli utenti un punto d'incontro virtuale per scambiarsi messaggi, foto e video.

Negli ultimi anni, i dispositivi mobile si sono evoluti sfruttando le tecnologie sempre più efficienti a disposizione. Nel 2012 è stata lanciata la rete 4G, che è caratterizzata da una trasmissione ad ampia larghezza di banda e garantisce prestazioni competitive rispetto ad una rete domestica [9].

I dispositivi mobile hanno rivoluzionato la nostra società tanto che gli smartphone sono al secondo posto tra le innovazioni del terzo millennio (secondi solo ai social network, strettamente legati ai dispositivi mobile) [11].

Secondo quanto emerge da Global Digital 2018 (un'indagine statistica sul mondo digitale [12], di cui è riportato un estratto riassuntivo nella figura 3.2), attualmente gli utenti mobile sono il 68% della popolazione mondiale e la quasi totalità degli utenti dei social media utilizza dispositivi mobile per accedervi. Inoltre, gli utenti mobile sono in costante crescita (+4% nell'ultimo anno). Questi dati mostrano l'importanza di questi dispositivi, sempre più utilizzati e adatti ai contesti applicativi più disparati.

Figura 3.2: Utenti internet, mobile e social media a livello mondiale [12]

[4]**Wi-Fi**: Wireless Fidelity, è una tecnologia di rete wireless che utilizza le onde radio per fornire connessioni internet ad alta velocità.

[5]**GPS**: Global Positioning System, è un sistema di posizionamento satellitare.

3.1.2 Caratteristiche

La grande diffusione dei dispositivi mobile, precedentemente descritta, è dovuta alle elevate performance che garantiscono i modelli sul mercato. Proprio come i computer sono dotati di processori, ossia sistemi centrali di elaborazione, che sui dispositivi mobile sono basati su system-on-a-chip[6] e che hanno prestazioni non troppo inferiori rispetto a quelli dei PC.

I dispositivi mobile sono inoltre dotati di un sistema operativo[7], che deve affrontare problematiche più critiche (legate alla natura del dispositivo) rispetto all'OS di un computer: limitatezza delle risorse di memoria, assenza di alimentazione esterna, differenti tecnologie per l'accesso a internet, nuovi metodi di immisione dei dati e dimensioni ridotte del display.

Le funzionalità sono messe a disposizione dell'utente attraverso dei software applicativi, le "app" che tutti noi conosciamo, che sfruttano il dispositivo, le sue risore (computazionali e di memoria) e i sensori di cui esso dispone (camera, GPS, ...) per realizzare i compiti più vari.

Sistemi operativi mobile

Il mercato dei sistemi operativi mobile è eterogeneo, poiché gli OS presenti hanno differenze significative. Perciò, qualora si voglia realizzare un'applicazione, è necessario considerare tutte le piattaforme a cui si vuole dare supporto. Come mostrato dalla figura 3.3, questo mercato è dominato da Android (con l'85,0% del volume di vendita totale) e iOS (con il 14,7%). Tutte le altre piattaforme (Windows Phone, ma anche BlackBerry OS e Nokia Symbian), pur avendo avuto numeri importanti in passato, ora sono marginali [13].

Period	Android	iOS	Windows Phone	Others
2016Q1	83.4%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%
2016Q3	86.8%	12.5%	0.3%	0.4%
2016Q4	81.4%	18.2%	0.2%	0.2%
2017Q1	85.0%	14.7%	0.1%	0.1%

Figura 3.3: Mercato mondiale dei sistemi operativi per smartphone [13]

[6] **System-on-a-chip**: Sistema in cui sono integrati il processore centrale, il chipset, eventuali controller di memoria, la circuiteria di I/O e il sottosistema video.

[7] **Sistema operativo**: Software di sistema che gestisce le risorse hardware e software del dispositivo, fornendo servizi di base ai software applicativi installati.

3.2 Approcci allo sviluppo: Nativo, web e cross-platform

Come già spiegato, a causa dell'eterogeneità delle piattaforme (i sistemi operativi) mobile, prima di procedere allo sviluppo di un'applicazione sarà necessario considerare tutte le piattaforme a cui essa è destinata.

Queste considerazioni, unite a valutazioni sull'obiettivo da realizzare e sulle risorse a disposizione, portano il progettista a scegliere un approccio allo sviluppo: nativo, web oppure cross-platform. Di seguito saranno illustrati questi approcci, evidenziandone vantaggi e svantaggi [14, 15].

3.2.1 Approccio nativo

Un'applicazione si definisce nativa se viene realizzata solo per un sistema operativo, che è caratterizzato da un proprio linguaggio di programmazione e da librerie proprietarie (attraverso cui vengono scritte e compilate tutte le applicazioni). Per sviluppare un'applicazione nativa occorre aver installato sul proprio computer l'IDE[8] e l'SDK[9] specifici della piattaforma e, spesso, questo implica l'utilizzo di un sistema operativo specifico (sul PC).

Infatti, facendo riferimento alle piattaforme mobile più diffuse, per realizzare un'applicazione per Android è necessario utilizzare i linguaggi Java, C e XML e un IDE tra Android Studio, Eclipse e Netbeans (questi possono essere però installati su più sistemi operativi) [16]. Per un'applicazione per iOS, bisogna invece servirsi dei linguaggi Objective-C, Swift e C e di Xcode come IDE, installandolo su un computer MAC con sistema operativo aggiornato [17].

Vantaggi dell'approccio nativo

- Le applicazioni possono interfacciarsi completamente con le API della piattaforma, interagendo con tutte le componenti hardware e software accessibili. In questo modo, potranno sfruttare e integrare le funzioni più importanti del device (come l'accesso alla camera e al gps).

- L'efficienza dell'applicazione realizzata è la massima che si può ottenere per il dispositivo. Ad esempio, il rendering grafico è accelerato e le transizioni tra le schermate sono agevoli e immediate.

[8] **IDE**: Integrated development environment, ossia ambiente di sviluppo integrato, è un software che aiuta i programmatori nello sviluppo del codice sorgente di un programma.

[9] **SDK**: Software development kit, indica genericamente un insieme di strumenti per lo sviluppo e la documentazione di software.

- Ottima user experience[10] e semplicità d'uso, poiché la piattaforma nativa è la scelta preferita dagli utenti che hanno familiarità con il dispositivo.

Svantaggi dell'approccio nativo

- Il principale svantaggio è la scarsa portabilità[11] poiché, per definizione, un'applicazione nativa è realizzata soltanto per un sistema operativo. Nel caso in cui si volesse garantire il supporto a più piattaforme, sarebbe necessario procedere con più linee di sviluppo parallele e, ovviamente, il costo di produzione del software crescerebbe in maniera notevole.
- Realizzare un'applicazione nativa richiede conoscenze specifiche della piattafoma, e lo sviluppatore deve necessariamente averle.

3.2.2 Approccio web

Le web application sono vere e proprie applicazioni web (conosciute dall'utente medio come "siti internet"), che vengono eseguite su un dispositivo mobile grazie al suo browser. Si tratta di applicazioni che non sono installate sul dispositivo, ma risiedono su un server remoto e sono implementate utilizzando le classiche tecnologie web (HTML, CSS e JavaScript).

Vantaggi dell'approccio web

- Le web application non presentano il problema della portabilità, poiché possono essere eseguite su ogni dispositivo che ha accesso a internet.
- Gli sviluppatori non devono avere alcuna conoscenza specifica delle piattaforme mobile su cui sarà eseguita l'applicazione.
- Il server su cui risiedono le applicazioni web si occupa delle operazioni computazionali e questo implica un risparmio delle risorse del dispositivo e, potenzialmente, una maggiore capacità di calcolo.

Svantaggi dell'approccio web

- Le web application non sono in grado di funzionare offline, ossia quando il dispositivo non è connesso a internet. Questo fa sì che si tratti di sistemi software non robusti[12].

[10] **User Experience**: Percezioni e reazioni di un utente, che derivano dall'uso o dall'aspettativa d'uso di un prodotto, sistema o servizio.

[11] **Portabilità**: Capacità di un software di funzionare su più piattaforme.

[12] **Robustezza**: Capacità di un software di comportarsi in modo ragionevole anche in circostanze non previste.

- Un altro grave difetto di questo tipo di applicazioni consiste nel fatto che, nella maggior parte dei casi, esse non sono in grado di accedere alle componenti software e hardware del dispositivo, poiché questo accesso non è mai diretto ma sempre tramite browser.

- A livello di user experience, sebbene oggi molte web application siano progettate secondo un approccio mobile first[13], si ottengono risultati inferiori rispetto a quelli di un'applicazione nativa.

3.2.3 Approccio cross-platform

Negli ultimi anni si è sviluppato l'approccio cross-platform (multipiattaforma), nato per creare applicazioni simili a quelle native (utilizzabili offline e installabili nel browser) ma anche adatte a piattaforme eterogenee [18].

L'idea di base è quella di trovare un linguaggio "universale", e quindi usare lo stesso codice sorgente, per più piattaforme. I framework cross-platform usano tecniche diverse, ma forniscono tutti la possibilità di accedere alle funzionalità e alle componenti del dispositivo.

Vantaggi dell'approccio cross-platform

- Per definizione, questo approccio risolve il problema della portabilità.

- A differenza delle web application, le applicazioni cross-platform vengono installate sul dispositivo e sono utilizzabili offline (proprio come le applicazioni native). Nella maggior parte dei casi l'utente medio non nota la differenza tra un'applicazione di questo tipo e una nativa.

- È possibile l'interazione con le componenti (hw e sw) del dispositivo.

Svantaggi dell'approccio cross-platform

- A livello di prestazioni e user-experience, un'applicazione cross-platform è comunque inferiore rispetto a un'applicazione nativa.

- L'interazione con le componenti del device è si possibile, ma limitata dal supporto che fornisce la piattaforma. Perciò potrebbe essere complicato, e talvolta impossibile, implementare funzionalità molto specifiche.

- È necessaria una conoscenza media delle piattaforme di sviluppo (ad esempio, le parti non condivise sono scritte in codice nativo).

[13] **Mobile First**: Approccio alla progettazione di applicazioni web che prevede di partire dai limiti posti dai dispositivi mobili, per concentrarsi al meglio sui contenuti essenziali.

3.3 Framework per lo sviluppo di applicazioni cross-platform

Dall'analisi dei requisiti del sistema da realizzare (eseguita nel capitolo 2), è emersa l'importanza di due requisiti non funzionali: la portabilità e la robustezza. Il primo fa sì che un approccio nativo non sia adatto a questo contesto, poiché si deve realizzare un'applicazione supportata da più piattaforme. Il secondo esclude l'approccio web, perché in un contesto di emergenza non ha alcun senso realizzare un'applicazione che non funziona offline.

Perciò l'approccio cross-platform è quello che meglio si adatta al caso in esame, e sarà necessario approfondirlo. Oggi esistono numerosi framework per lo sviluppo mobile multipiattaforma [19], e qui ne saranno analizzati tre tra i più diffusi: Apache Cordova, Xamarin e NativeScript.

Si tratta di framework open-source[14] che utilizzano tecniche decisamente differenti per fornire il supporto cross-platform, e il loro approfondimento porterà ad avere un quadro tecnologico molto più ampio e dettagliato.

3.3.1 Apache Cordova

Apache Cordova è un framework che consente di sviluppare le cosiddette applicazioni ibride. Si tratta di applicazioni cross-platform che sono realizzate attraverso le tecnologie web standard (HTML5[15], CSS3[16]. e JavaScript[17]) ma che, a differenza delle web application, sono installate nel dispositivo e vengono eseguite all'interno di un wrapper nativo (che funge da contenitore).

In particolare, in Apache Cordova le applicazioni vengono implementate come pagine web e poi eseguite all'interno della WebView messa a disposizione dall'ambiente nativo (che fornisce loro l'interfaccia utente). C'è inoltre la possibilità di accedere alle funzionalità del dispositivo (sensori, dati, stato della rete, ...) attraverso le API della piattaforma [20]. Cordova fornisce supporto alle piattaforme mobile più diffuse: Android, iOS e anche Windows Phone. Di seguito (nella figura 3.4), sarà presentato un diagramma architetturale contente i principali componenti di un'applicazione Cordova.

[14] **Open-source**: Caratteristica di un software, che fa sì che l'utente finale possa liberamente accedere al suo codice sorgente e utilizzarlo.

[15] **HTML5**: Linguaggio di markup per la strutturazione delle pagine web.

[16] **CSS3**: Linguaggio usato, nel web, per definire la formattazione dei documenti HTML

[17] **JavaScript**: Linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione web lato client.

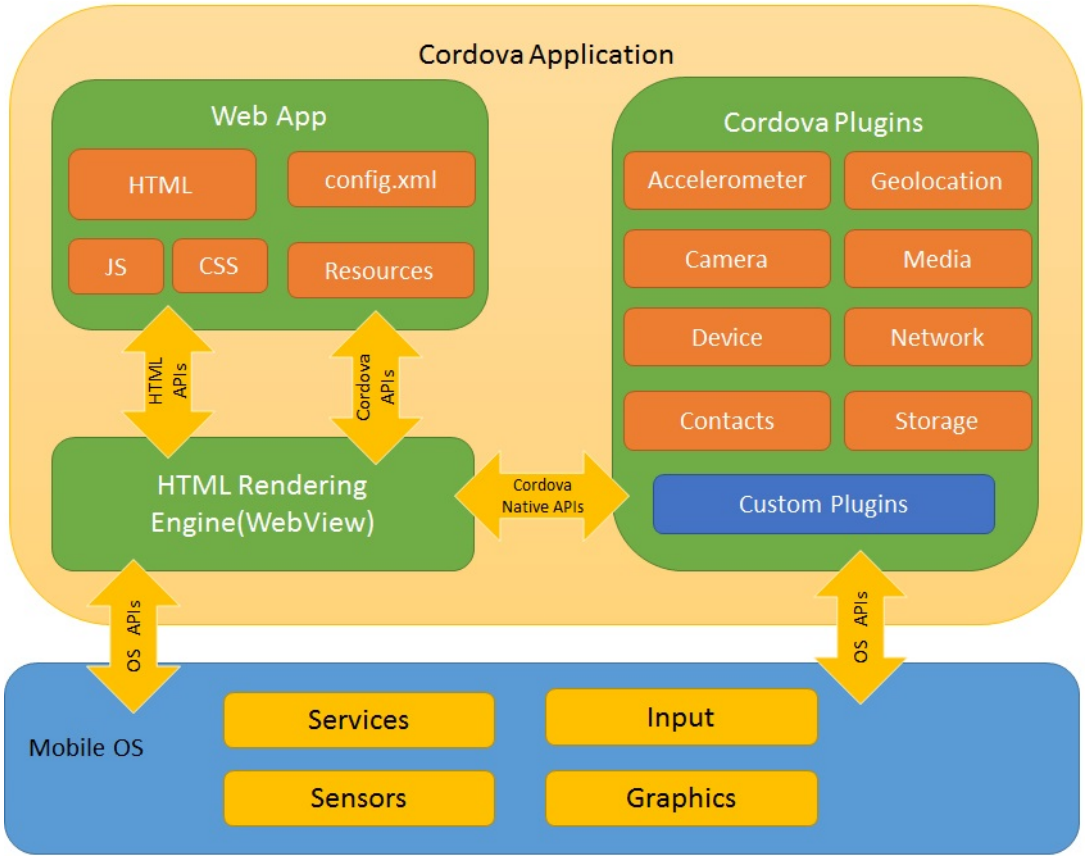

Figura 3.4: Vista di alto di livello dell'architettura di un'app Cordova [20]

Come emerge dal diagramma architetturale, una componente fondamentale delle applicazioni Cordova è costituita dai plugin. Essi utilizzano le API native della piattaforma e forniscono un'interfaccia che permette la comunicazione con le componenti del dispositivo. I plugin si possono richiamare direttamente dal codice JavaScript, e perciò lo sviluppatore può usarli facilmente.

Apache Cordova mantiene una serie di plugin "ufficiali", ossia i Core Plugin [21]. Questi forniscono delle funzionalità di base all'applicazione, come ad esempio l'acceso allo stato della batteria, alla fotocamera, al file-system del dispositivo e al sensore di geolocalizzazione. In generale i Core Plugin forniscono supporto a tutte le piattaforme mobile, e il team di Cordova si preoccupa di garantirne la compatibilità con nuove versioni dei sistemi operativi.

Oltre ai plugin di base esistono diversi plugin di terze parti che mettono a disposizione altre funzionalità, e possono essere compatibili con una o più piattaforme. I plugin "non ufficiali" sono reperibili tramite una ricerca nel sito `https://cordova.apache.org/plugins/`, ma prima di utilizzarli bisogna verificarne la compatibilità con l'attuale versione della piattaforma. Uno

sviluppatore impegnato nella realizzazione di un'applicazione Cordova può anche decidere di sviluppare un proprio plugin per implementare una certa funzionalità, ma questo comporta un importante dispendio di tempo (oltre alla necessità di conoscere il linguaggio nativo delle piattaforme).

Strumenti di sviluppo

Apache Cordova offre due differenti percorsi di sviluppo: uno multipiattaforma e uno centrato su una piattaforma specifica. Analizzando per ovvi motivi il primo si può dire che è basato su una CLI[18], uno strumento di alto livello che consente di gestire progetti destinati a molte piattaforme. Per ognuna di esse, la CLI ha i seguenti compiti: copiare un insieme comune di risorse web in sottodirectory, apportare le modifiche necessarie alla configurazione e eseguire script di compilazione per generare i binari dell'applicazione.

Lo sviluppatore, attraverso la CLI di Cordova, può in maniera semplice e immediata: creare un nuovo progetto, aggiungere le piattaforme che vuole supportare, aggiungere i plugin che vuole utilizzare e lanciare l'applicazione (o su un emulatore o su un device fisico). Il codice comune a tutte le piattaforme (ossia quello web) dovrà essere inserito all'interno della directory "www", creata automaticamente dalla CLI all'interno di un nuovo progetto.

Porting su dispositivi non mobile

Le applicazioni realizzate con Apache Cordova sono vere e proprie applicazioni web e, perciò, possono essere eseguite direttamente su dispositivi non mobile attraverso il browser. Questo comporta però delle limitazioni, in quanto le funzionalità implementate con i plugin (che, come visto, contengono codice specifico della piattaforma di destinazione) non saranno disponibili.

Per risolvere queste limitazioni il framework Cordova ha introdotto una nuova piattaforma: "browser". Può essere aggiunta e usata come le altre (Android, iOS e Windows Phone) e permette di creare applicazioni web partendo da un progetto Cordova, usando anche le funzionalità messe a disposizione dai plugin (ovviamente solo quelli che supportano la piattaforma browser).

3.3.2 Xamarin

Xamarin è un framework cross-platform, di proprietà della Microsoft. Nasce dalla considerazione secondo la quale l'approccio cross-platform, pur avendo il vantaggio di riutilizzare il codice, porta spesso ad avere applicazioni la cui

[18] **CLI**: Command-line interface, ossia interfaccia a riga di comando

interfaccia utente non si adatta bene a nessuna delle piattaforme di destinazione. Questo framework si pone l'obiettivo di ottenere il meglio da entrambi i mondi (nativo e cross-platform) e cerca di raggiungerlo così: presentando interfacce utente native su ciascuna piattaforma e condividendo la maggior parte possibile del codice restante (contenente la logica dell'applicazione).

Xamarin segue un approccio allo sviluppo di tipo cross-compilato: le applicazioni sono scritte in un linguaggio di programmazione comune, ossia C#, e un cross-compiler si occupa della traduzione del codice sorgente nei file binari necessari per l'esecuzione dell'applicazione. Le piattaforme supportate sono Android, iOS e Windows Phone, e per ognuna di esse Xamarin fornisce la possibilità di accedere a tutte le funzionalità native della piattaforma.

Nella fase di progettazione di un'applicazione di questo tipo è fondamentale la separazione dei compiti, per distinguere le parti dell'applicazione che dovranno essere condivise da quelle specifiche dell'interfaccia utente. Il codice di base (condiviso) è quello che fornisce servizi per le parti dell'applicazione che raccoglieranno e visualizzeranno queste informazioni. Spesso, per ottenere una migliore organizzazione dei contenuti, si usano pattern di progettazione come MVC (Model-View-Controller) e MVVM (Model–view–viewmodel) [22].

La figura 3.5, che segue, contiene un diagramma che riporta un esempio dell'architettura di un'applicazione realizzata con Xamarin. In essa, è evidente la distinzione tra il codice condiviso e quello replicato per le varie piattaforme.

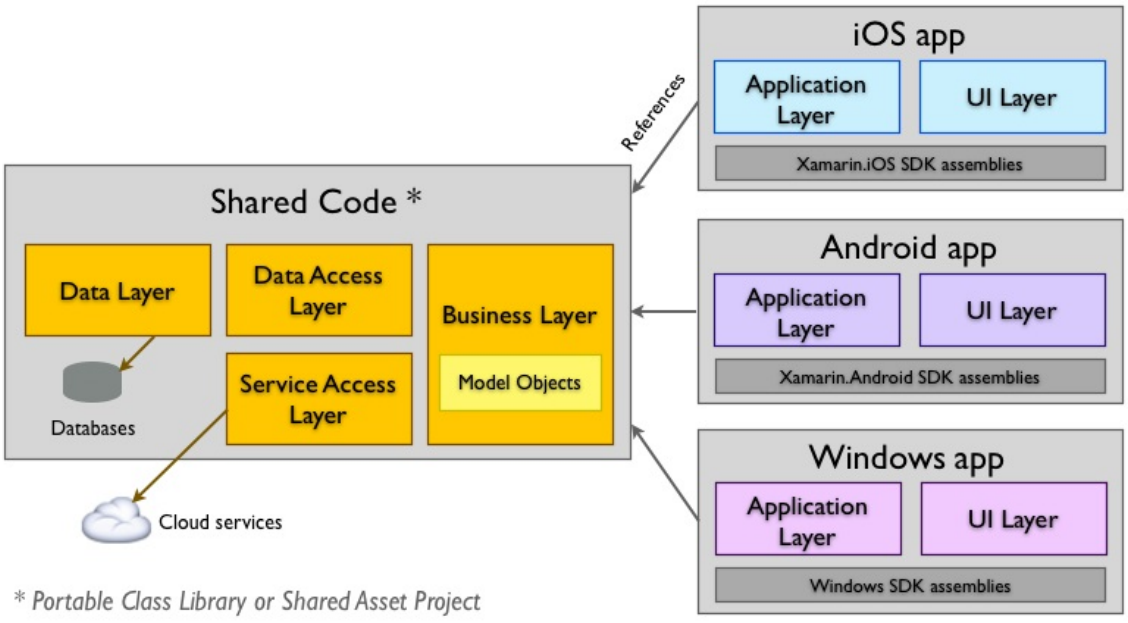

Figura 3.5: Esempio di architettura di un'applicazione Xamarin [24]

Strumenti di sviluppo

In modo schematico si pùò dire che Xamarin è composto dai seguenti elementi, che gli permettono di sviluppare applicazioni cross-platform [23]:

- Linguaggio C#, che consente di sviluppare utilizzando un approccio ad oggetti e, grazie a Xamarin, di accedere alle funzionalità fornite dagli SDK specifici delle piattaforme.
- Compilatore e Mono .NET framework, il cui compito è quello di produrre un'applicazione nativa (diversa in base alla piattaforma scelta) partendo dal codice sorgente in linguaggio C#.
- Strumento IDE, ossia Visual Studio (utilizzabile su un computer con sistema operativo Windows o MAC).

3.3.3 NativeScript

NativeScript è un framework introdotto più recentemente rispetto a Cordova e Xamarin, e fornisce supporto alle due piattaforme mobile più diffuse: Android e iOS. Propone un approccio allo sviluppo simile a quello ibrido, poiché le applicazioni sono realizzate utilizzando i linguaggi tipici dello sviluppo web, ma che si distingue da esso per due differenze sostanziali.

Innanzitutto, le applicazioni realizzate con NativeScript non sono applicazioni web incapsulate in una WebView ma sono completamente native (come in Xamarin, gli elementi dell'interfaccia grafica sono nativi). Inoltre è possibile accedere alle API native della piattaforma in modo diretto (dal codice web e senza passare attraverso uno strato intermedio, di cui sono un esempio i plugin di Cordova). Per i motivi appena elencati le applicazioni create con NativeScript si possono definire Native Web Applications [25].

Per quanto riguarga i linguaggi di programmazione, NativeScript fornisce la possibilità di usare JavaScript ma anche TypeScript[19]. Questo fa sì che in fase di progettazione possano anche essere scelti approcci ad oggetti. Un'altra particolarità di NativeScript consiste nella possibilità di integrarsi con due framework molto diffusi in ambito web (Angular e Vue.js, che utilizzano rispettivamente TypeScript e JavaScript) in modo da sfruttarne tutti i vantaggi, come ad esempio strutture architetturali ben definite che consentono di creare applicazioni di ottima qualità e facilmente manutenibili.

Nella figura 3.6 è riportato un diagramma che mostra le tecnologie usate da NativeScript, che gli permettono di trasformare un'applicazione da web

[19]**TypeScript**: Linguaggio che estende la sintassi di JavaScript e che è caratterizzato da un approccio ad oggetti. Il codice TypeScript viene ricompilato in JavaScript per poter essere interpretato da browser o applicazioni.

a nativa. In particolare in un'applicazione NativeScript, oltre al framework utilizzato a livello applicativo (Angular, Vue.js o un classico approccio web), hanno una grande importanza due concetti: Core Module e Plugin.

I Core Modules forniscono le astrazioni necessarie per accedere alle piattaforme native sottostanti, implementando così alcune funzionalità di base (come l'accesso alla console e la possibilità di eseguire chiamate http). Un altro elemento messo a disposizione dai Core Modules è una tecnica di base, basata su XML, per definire interfacce utente, associazione di dati e navigazione.

Il concetto di Plugin è invece simile a quello descritto per Cordova, poiché si tratta di blocchi di codice che incapsulano alcune funzionalità e aiutano a sviluppare applicazioni più velocemente (lo stesso NativescriptCoreModules è un plugin). A differenza dei plugin di Cordova, non sono scritti in linguaggio nativo ma in JavaScript o TypeScript [26].

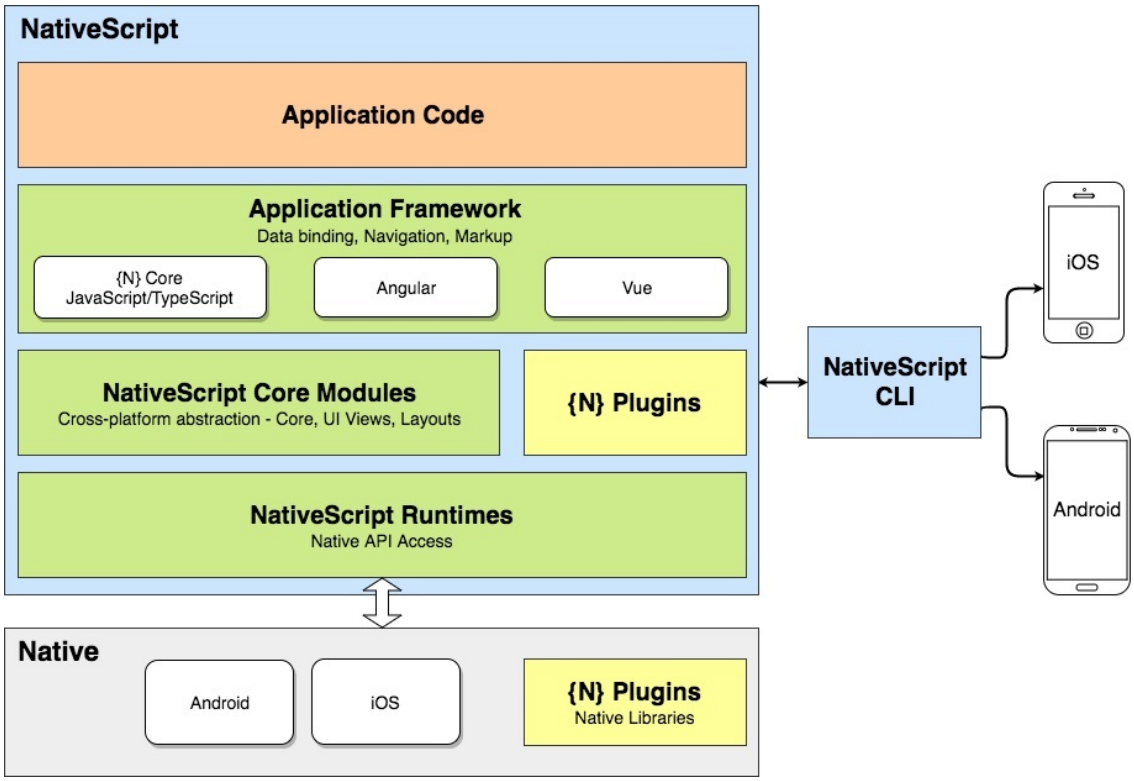

Figura 3.6: Diagramma che illustra il funzionamento di Nativescript [26]

Strumenti di sviluppo

Come emerge anche dal diagramma 3.6, i NativeScript Runtimes consentono di chiamare le API dei framework Android e iOS attraverso il codice Java-

Script. Per farlo usano JavaScriptVirtualMachines-V8 di Google per Android e l'implementazione JavaScriptCore di WebKit distribuita con iOS 7.0+.

Nativescript mette a disposizione degli sviluppatori una CLI, che attraverso semplici comandi consente di eseguire molte operazioni. Ad esempio, la CLI permette di creare un nuovo progetto, aggiungere o rimuovere una piattaforma, aggiungere o rimuovere un plugin, compilare e lanciare (su dispositivo fisico o emulatore) l'applicazione avendo come target una piattaforma specifica.

Porting su dispositivi non mobile

NativeScript realizza applicazioni mobile native, apparentemente non portabili su sistemi non mobile, ma l'integrazione con il framework Angular ha consentito notevoli progressi in questo senso. Gran parte degli elementi presenti in un'applicazione Angular, infatti, può essere usata sia nel web sia nel mobile (grazie a NativeScript) e Angular ha da poco fornito la possibilità di creare un progetto con struttura di condivisione del codice [27].

In particolare sarà condiviso il codice relativo alla logica dell'applicazione, mentre sarà necessario separare il codice dell'interfaccia utente (completamente diverso nel web e nel mobile, poiché in ambito mobile NativeScript mette a disposizione elementi grafici che vengono wrappati in elementi nativi) e i moduli specifici relativi alle piattaforme.

3.4 Valutazione dei framework analizzati

Dopo aver visto nel dettaglio questi tre framework, è possibile metterli a confronto. Nella tabella che segue sono riassunte alcune caratteristiche di Cordova, Xamarin e NativeScript, che saranno utili per valutarli.

	Cordova	**Xamarin**	**NativeScript**
Piattaforme supportate	Android, iOS e Windows Phone	Android, iOS e Windows Phone	Android e iOS
Linguaggi utilizzati	JavaScript, HTML e CSS	C#, XAML	JavaScript (o TypeScript), XML e CSS
Interfaccia utente	Interfaccia web incapsulata in una WebView nativa	Nativa su tutte le piattaforme	Nativa su tutte le piattaforme

Accesso alle API native	Possibile, ma soltanto attraverso plugin	Possibile direttamente da C#	Possibile direttamente da JavaScript (o TypeScript)
Quantità di codice replicato	Minima, limitata a funzionalità specifiche implementate attraverso plugin	Consistente, poiché il codice relativo all'interfaccia utente va replicato per ogni piattaforma	Minima, limitata a funzionalità specifiche implementate accedendo alle API native
Organizzazione dell'applicazione	Non facilitata dal framework e a discrezione dello sviluppatore	Facilitata dall'architettura proposta dal frameowrk	Facilitata dalla possibilità di utilizzare Angular o Vue.js
Porting su dispositivi non mobile	Semplice, con condivisione quasi totale del codice e limitazioni date solo dai plugin	Impossibile, poiché il framework è pensato solo per creare applicazioni mobile	Possibile grazie ad Angular, ma con l'obbligo di replicare l'interfaccia utente

Come evidenziato in precedenza, Cordova non permette di realizzare applicazioni mobile native e questo limite va considerato in relazioni ai requisiti individuati per il sistema (vedi 2.2). Esso deve essere usabile, efficiente e deve possedere un'interfaccia utente che permetta un'interazione molto rapida. Sicuramente una UI nativa garantisce risultati migliori rispetto a quelli che si possono raggiungere con la WebView usata da Cordova.

Xamarin supera questa criticità, ma obbliga a replicare parti di codice (l'interfaccia utente) per ogni piattaforma supportata. Questo implica un maggiore lavoro sia per lo sviluppo sia per eventuali modifiche, contraddicendo un altro requisito non funzionale individuato: la manutenibilità.

NativeScript non presenta queste problematiche, ed è stato perciò scelto come framework da utilizzare per la realizzazione dell'applicazione PreH Soccorso. In particolare verrà approfondita la sua integrazione con Angular, che permette di realizzare un sistema modulare e, potenzialmente, portabile su dispositivi non mobilre (con le limitazioni precedentemente descritte).

Capitolo 4

Progettazione e sviluppo dell'applicazione PreH Soccorso

In seguito all'approfondimento tecnologico svolto nel capitolo 3 si è scelto di usare NativeScript, integrato con Angular, per realizzare l'applicazione PreH Soccorso. Infatti questo framework è stato considerato adatto in relazione ai requisiti definiti per il sistema (capitolo 2), sia in termini di funzionalità che in termini di qualità del software (portabilità, modularità, efficienza, ...).

In questo capitolo saranno presentate le fasi di progettazione e sviluppo dell'applicazione, mettendo in evidenza le scelte progettuali fatte.

Per quanto riguarda la progettazione, sarà descritta in modo dettagliato la tecnologia di riferimento, costituita da NativeScript & Angular, e saranno illustrati i principali concetti che la caratterizzano (4.1). Successivamente saranno riportati il design dell'interfaccia utente (4.2), la modellazione dei dati che l'applicazione dovrà raccogliere (4.3) e l'architettura del sistema (4.4).

Per quanto riguarda lo sviluppo, sarà presentato il sistema ottenuto mettendo in luce le sue caratteristiche e l'implementazione svolta (4.5).

4.1 Tecnologia scelta: NativeScript & Angular

"NativeScript doesn't require Angular, but it's even better when you use it". Angular è un framework open-source, la cui prima versione risale al 2016, che sta spopolando nello sviluppo di applicazioni web. NativeScript dà la possibilità di usare gli elementi propri di un'applicazione Angular (architettura di riferimento, concetti principali usati nello sviluppo, ...) in ambiente mobile, integrandoli perfettamente con le caratteristiche che fornisce lui stesso come piattaforma (viste nel capitolo precdente). Perciò, l'integrazione con Angular rende NativeScript un framework molto più completo [28].

Saranno presentati i concetti principali su cui si basa un'applicazione NativeScript creata con Angular 5. Relativamente alle caratteristiche specifiche di Angular, rispetto alle applicazioni web, quelle mobile realizzate usandolo con NativeScript presentano alcune differenze e qui saranno descritte le caratteristiche di queste (trattandosi delle tecnologie usate per realizzare un'applicazione mobile, sarebbe superfluo approfondire lo sviluppo Angular sul web).

4.1.1 Struttura progettuale

Come introdotto nel capitolo precedente, la CLI di NativeScript dà la possibilità di creare una nuova applicazione attraverso il comando *tns create* seguito dal nome dell'applicazione. È inoltre possibile, con l'opzione *–template* seguita dal nome del template, specificarne uno e definire così un modello predefinito che costituisce un punto di partenza per lo sviluppo. Se non viene indicato nessun template, NativeScript ne utilizza uno di default.

Ci sono template creati appositamente per utilizzare Angular all'interno di un'applicazione NativeScript, come il *nativescript-template-ng-tutorial*. Esso prevede l'utilizzo di TypeScript e fa sì che, inizialmente, venga creata un'applicazione a singola pagina e senza stili personalizzati, in modo da consentire la massima autonomia nella progettazione e nella strutturazione.

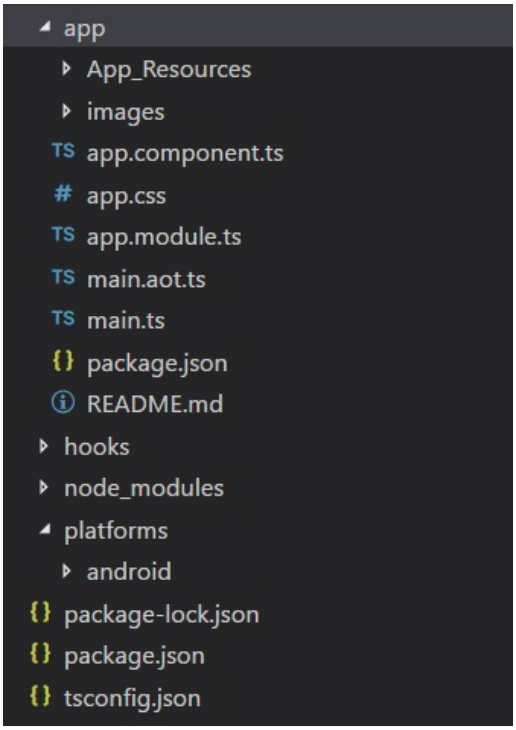

Figura 4.1: Struttura d'esempio di un progetto NativeScript che usa Angular

Nella figura 4.1 è riportata la struttura di un progetto NativeScript creato con il template *nativescript-template-ng-tutorial*, dopo l'aggiunta della piattaforma Android. Si notano tre cartelle principali: *app*, nella quale lo sviluppatore inserisce il proprio codice, *node _modules*, in cui si trovano i moduli usati dall'applicazione e *platforms*, che contiene i file specifici per Android e iOS.

Sono importanti i file *package.json*, che descrive il progetto e tiene traccia di tutto ciò che è stato installato (piattaforme, moduli, ...), *main.ts* e *main.aot.ts*, che si occupano del bootstrap dell'applicazione. Lo sviluppatore deve modificare soltanto il codice contenuto nella cartella *app*, mentre le *platforms* e i *node_modules* possono essere aggiunti con la CLI.

Moduli

Una caratteristica fondamentale delle applicazioni NativeScript create con Angular è la modularità. Esse sono costituite infatti da un insieme di moduli, file contenenti blocchi di codice dedicati a scopi specifici e utilizzabili da altre parti dell'applicazione (grazie alle clausole *import* e *export*) [30].

Tra i moduli si possono distinguere quelli creati dallo sviluppatore, quelli forniti direttamente con NativeScript e quelli (resi disponibili dal framework o da terzi, ad esempio sottoforma di plugin) che si possono installare con NPM[1].

4.1.2 TypeScript

TypeScript è il linguaggio di programmazione di riferimento per Angular e costituisce un super-set di JavaScript poiché estende la sintassi di quest'ultimo. Le sue caratteristiche principali sono l'orientamento agli oggetti e lo static typing, ossia il fatto che il tipo delle variabili è definito al momento della compilazione (che garantisce una maggiore semplicità nella fase di debug).

In TypeScript può essere usato anche codice scritto con JavaScript e per questo motivo lo static tiping non è obbligatorio, anche se fortemente consigliato. I file TypeScript hanno estensione *.ts* e vengono compilati in JavaScript per essere poi interpretati dal browser (o, in questo caso, da NativeScript).

L'orientamento agli oggetti di TypeScript, con tutti i vantaggi che ne derivano, è dovuto al fatto che questo linguaggio supporta le classi e in tal senso la sintassi messa a disposizione è molto simile a quella di Java. Ad esempio è possibile creare interfacce ed enumerazioni, le classi possono ereditare metodi e atributi e possono essere definite delle classi generiche.

[1] **NPM**: Node Package Manaher, è un'utility che viene installata con Node.js e che gestisce i package di Node. NPM permette infatti di installare, condividere e distribuire il codice.

4.1.3 Architettura

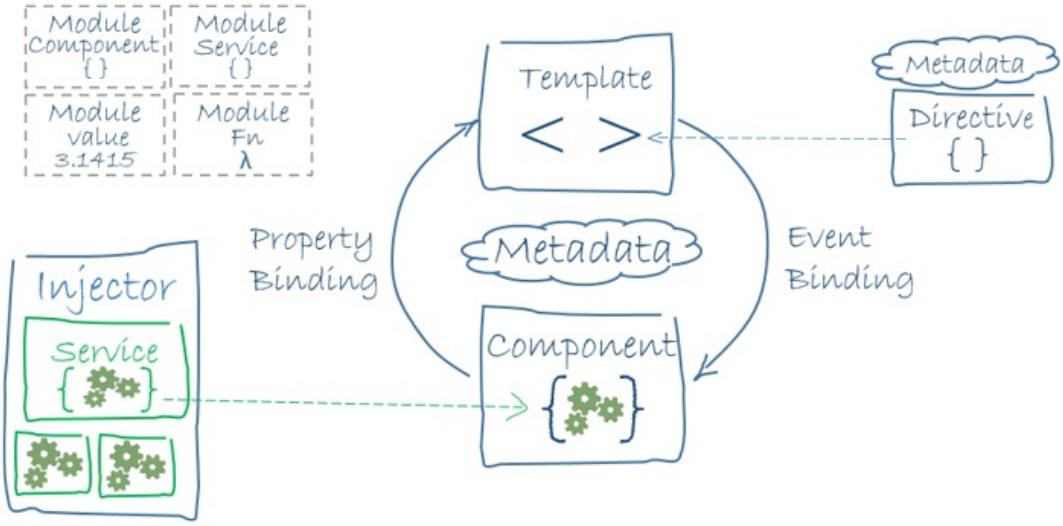

Figura 4.2: Panoramica dell'architettura Angular [31]

Angular presenta un'architettura specifica a cui fanno riferimento anche le applicazioni NativeScript create con Angular (vedi figura 4.2).

All'interno di essa gli elementi principali sono i componenti, che definiscono l'interfaccia utente e la logica che la controlla. I componenti utilizzano dei servizi, condivisi all'interno dell'applicazione, che forniscono funzionalità specifiche non direttamente correlate alle viste. Sia i componenti che i servizi sono semplici classi, caratterizzate da decoratori[2] che contrassegnano il loro tipo e forniscono metadati che indicano ad Angular come usarli [29, 31].

I componenti di un'applicazione generalmente definiscono molte viste, disposte gerarchicamente. Angular mette a disposizione il servizio Router, che permette allo sviluppatore di definire i percorsi di navigazione tra le viste.

4.1.4 Componenti

Ogni applicazione NativeScript creata con Angular contiene una serie di componenti, che definiscono schermate o elementi dell'interfaccia utente. Un componente è costituito da due elementi principali: una classe TypeScript a cui è associato il decoratore *@Component* e che stabilisce la logica e il comportamento del componente, e un template (chiamato anche *view*), che ha il compito di definire la sua interfaccia utente [30].

[2]**Decoratore**: In inglese decorator, è un pattern usato nella programmazione a oggetti per arricchire una classe con funzionalità specifiche.

Il decoratore *@Component* permette di creare un particolare componente, a cui poi sarà aggiunta la logica con la definizione della classe specifica, attraverso dei metadati. In particolare, i metadati disponibili per un componente sono:

- *selector*

  Selettore che fa in modo che Angular crei e istanzi questo componente in altri componenti, se essi contengono questo selettore nel template.

- *template, templateUrl*

  Template XML in cui è contenuta l'interfaccia utente associata al componente, che può essere definito sia inline (metadato template) sia in un altro file (individuato grazie al percorso indicato in templateUrl).

- *styles, styleUrls*

  Direttive CSS che definiscono lo stile di un componente, che possno essere indicate sia inline (styles) sia in appositi fogli di stile (styleUrls).

- *providers*

  Servizi che il componente ha la possibilità di utilizzare.

Template

Come è stato visto, i template sono fondamentali nella definizione di un componente poiché determinano ciò che viene effettivamente visualizzato. Mentre nelle applicazioni Angular create per il web il template è scritto in HTML, in quelle sviluppate con NativeScript si usa il linguaggio XML[3] [29].

Attraverso quest'ultimo è possibile realizzare interfacce utente con gli elementi grafici propri di NativeScript, che forniscono API comuni facendo però riferimento a due elementi mobile nativi (uno per Android e uno per iOS).

Sebbene vengano usati elementi diversi rispetto a quelli caratteristici dell'ambiente web, la sintassi del template funziona allo stesso modo ed è quindi possibile usare le stesse tecniche di associazione di dati e le stesse direttive.

Data binding

Senza avere un framework a disposizione, sarebbe necessario scrivere manualmente dei meccanismi che trasformano le azioni degli utenti in aggiornamenti dei valori a livello di logica. Questo comporterebbe grandi perdite di tempo e un rischio maggiore di commettere errori.

[3] **XML**: eXtensible Markup Language, è un linguaggio di markup che è nato come estensione di HTML.

Per evitare queste problematiche Angular supporta il data binding, un meccanismo che permette di coordinare l'interfaccia utente di un componente (template) con la sua logica (classe). La sintassi prevista dal data binding va inserita nei tag XML, in modo che Angular sappia "collegare i due lati" [32].

Esistono quattro tipi di data binding, che sono elencati di seguito e di cui, nell'ordine, è riportata la sintassi nella figura 4.3:

- Interpolazione, che mostra una proprietà del componente nel template.
- Property binding, che passa un valore dal componente a un elemento figlio (contenuto nel template).
- Event binding, che si occupa di chiamare un metodo di un componente al verificarsi di un determinato evento.
- Two-way data binding, detto anche data binding bidirezionale, che in un'unica notazione raggruppa property ed event binding. In questo caso infatti un valore del componente si trova nel template, come nel property binding, e le modifiche dell'utente non hanno ripercussioni solo sul template ma anche sul componente, come nell'event binding.

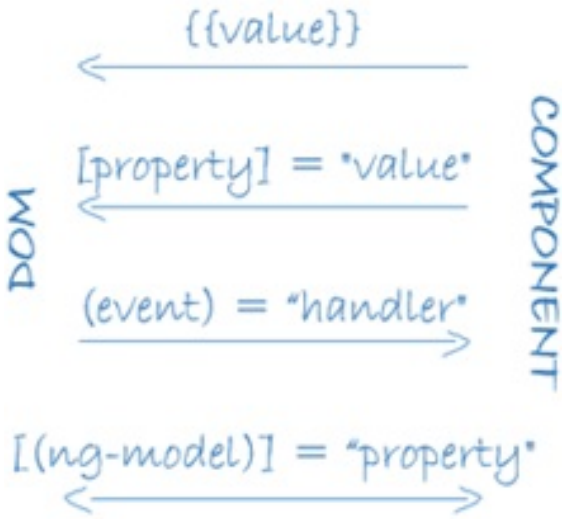

Figura 4.3: Sintassi dei tipi di data binding usati da Angular [32]

Direttive

Le direttive fanno parte della sintassi che può essere aggiunta al template, e consentono ad esso di assumere particolari comportamenti. Sono stati individuati due tipi di direttive: quelle strutturali e quelle di attributo.

Le direttive strutturali alterano la struttura del template aggiungendo, rimuovendo e sostituendo elementi. Tra queste sono utilizzate molto frequentemente **ngFor*, a cui è associata una lista di elementi e che permette di replicare nel template il tag entro cui è inserita per ogni elemento, e **ngIf*, a cui è associata un'espressione booleana e che consente di mostrare o nascondere elementi del template in base alla veridicità della condizione.

Le direttive di attributo modificano l'aspetto o il comportamento degli elementi del template. Tra queste ci sono *ngClass*, per aggiungere e rimuovere più classi contemporaneamente, e *ngStyle*, per impostare più stili inline [32].

Ciclo di vita

Ogni componente è caratterizzato da un proprio ciclo di vita, che è controllato dall'applicazione Angular. Essa si occupa infatti di creare, aggiornare e distruggere i componenti e mette a disposizione delle interfacce che permettono di intervenire nei principali momenti del ciclo di vita [30]. In particolare ci sono metodi che, se implementati, sono richiamati periodicamente:

- *ngOnInit()*

  Eseguito dopo l'inizializzazione di tutti i metodi di immissione dati.

- *ngOnChanges()*

  Eseguito dopo che una proprietà legata ai dati è stata cambiata.

- *ngOnDestroy()*

  Eseguito poco prima che Angular distrugga il componente.

4.1.5 Servizi

Generalmente un servizio è una classe che implementa una funzionalità specifica, non legata direttamente all'interfaccia utente (come il recupero di dati da un server o la convalida dell'input dell'utente). Angular distingue servizi e componenti per aumentare la modularità e la riusabilità del sistema poiché, separando la visualizzazione da altre elaborazioni, è possibile rendere più snelle ed efficienti le classi che contengono la logica dei componenti [33].

In Angular i componenti utilizzano i servizi grazie a una tecnica chiamata iniezione di dipendenza (DI): è possibile "iniettare" un servizio in un componente, dando al componente l'accesso al servizio. Perciò, per definire una classe come servizio in Angular, è necessario usare il decoratore *@Injectable*.

Iniezione di dipendenza

Alla base dell'iniezione di dipendenza vi è il concetto di iniettore, che viene prodotto da Angular secondo necessità (durante il processo di bootstrap). Un iniettore è un oggetto che mantiene un contenitore di istanze di servizi creati in precedenza, e poi li fornisce ai componenti che le richiedono.

Per qualsiasi dipendenza necessaria in un'applicazione, bisogna registrare un provider (in genere la stessa classe di servizio) in modo che l'iniettore possa utilizzarlo per creare nuove istanze. Così Angular può chiamare il costruttore del componente, inserendo tra gli argomenti i servizi che devono essere iniettati. Se nell'iniettore non è presente il servizio cercato, cercherà tra i provider registrati e provvederà a creare un'istanza della classe necessaria.

Nella figura 4.4 è riportato un esempio di iniezione di dipendenza, in cui il servizio HeroService viene iniettato nel componente HeroComponent.

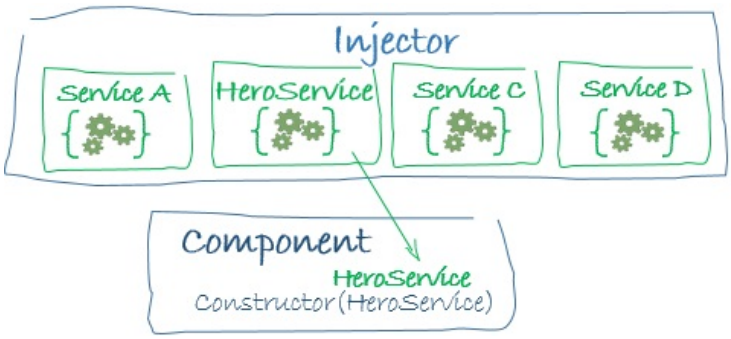

Figura 4.4: Esempio di iniezione di dipendenza in Angular [33]

Routing

Un servizio molto usato nelle applicazioni Angular è il Router, che permette di navigare tra le viste. Nelle applicazioni NativeScript create con Anguar sono disponibili due tipi di router: *router-outlet* e *page-router-outlet*.

Il *router-outlet* è il tradizionale router di Angular, che ad ogni navigazione sotituisce il componente presente nell'interfaccia utente.

Invece il *page-router-outlet* usa la navigazione tra le pagine tipica di NativeScript, e carica il componente in una nuova pagina. Questo tipo di navigazione è uniforme con quanto offerto dalle piattaforme mobile native, e perciò nello sviluppo mobile è preferibile al *router-outlet* [34].

4.2 Design dell'interfaccia utente

La fase di design è fondamentale per la realizzazione dell'interfaccia utente. Partendo dai requisiti dell'applicazione, che sono stati individuati dopo l'incontro con il dottor Albarello (vedi capitolo 2), è infatti possibile definire una struttura di massima dell'interfaccia che sarà sviluppata.

Uno strumento che è stato utilizzato in questa fase di design è il wireframe, che costituisce una bozza del lavoro da svolgere e mostra lo "scheletro" del front-end[4] dell'applicazione. Esso è un documento a bassa fedeltà, non navigabile (si tratta di un'immagine statica) e che descrive la giusta posizione degli elementi nella pagina. Attraverso il wireframe è quindi possibile comunicare l'idea iniziale del progetto e quello che sarà possibile vedere [35].

L'interfaccia grafica dovrà essere facile da usare, mettendo però a disposizione dell'utente tutte le funzionalità individuate. Un'altra caratteristica che dovrà avere è la responsiveness[5], la capacità di adattarsi al dispositivo.

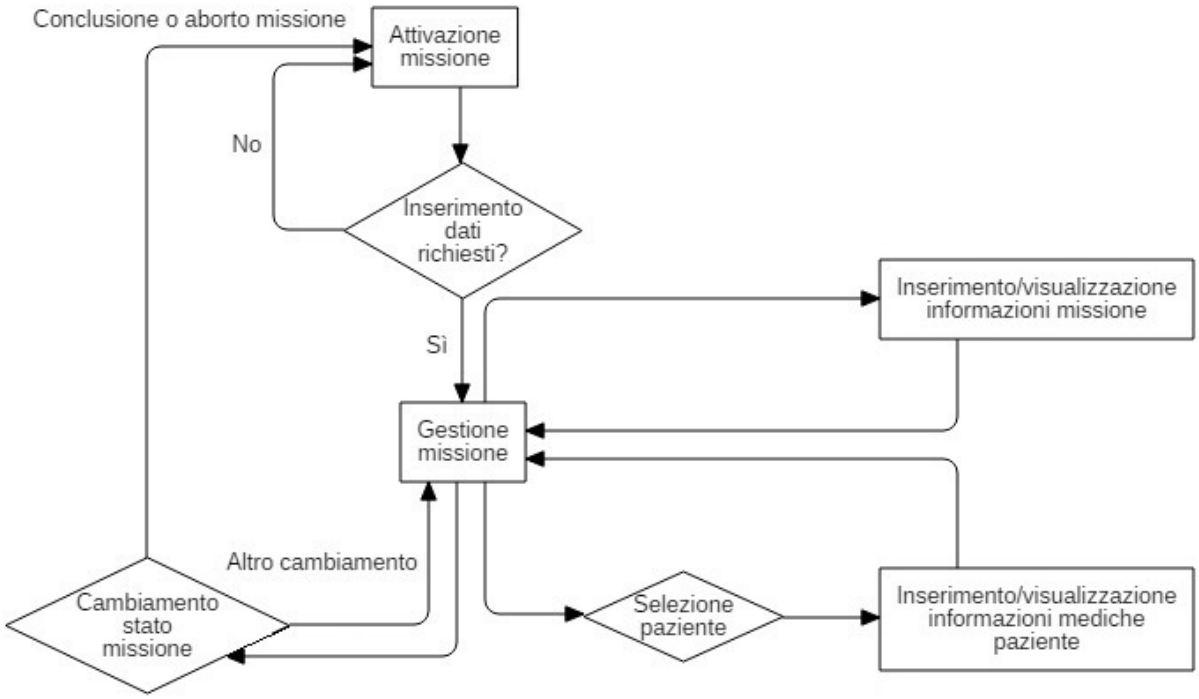

Figura 4.5: Flow-chart con la navigazione tra le principali pagine dell'app

[4]**Front-end**: Parte dell'applicazione che è responsabile dell'interazione con l'utente e dell'acquisizione dei dati in input.

[5]**Responsive design**: Approccio al design delle interfacce utente, nato in ambito web, che permette la realizzazione di applicazioni in grado di adattarsi automaticamente alle caratteristiche del dispositivo su cui vengono visualizzate.

Nella figura 4.5 riportata sopra è stato presentato un flow-chart[6], in cui sono rappresentate le principali pagine dell'applicazione e la navigazione tra esse. Come evidenziato dal diagramma, dovranno infatti essere presenti:

- Una pagina iniziale che permetta di attivare una nuova missione, obbligando l'utente ad inserire determinate informazioni per farlo (ad esempio, egli si deve identificare prima di poter procedere).
- Una pagina di gestione della missione, che metta a disposizione tutte le operazioni necessarie per il suo tracciamento. Da questa pagina sarà possibile cambiare lo stato di una missione e, in caso di conclusione o aborto, tornare alla pagina iniziale.
- Delle pagine che permettano l'inserimento e la visualizzazione delle informazioni che caratterizzano una missione, tra cui la dinamica dell'avvenimento che è stato la causa dell'intervento di soccorso e i dati generali relativi ai pazienti coinvolti. Ci sarà inoltre una pagina che consentirà di visualizzare tutti i cambiamenti di stato che si sono verificati nel corso della missione, in modo da mostrarne uno storico.
- Una serie di pagine attraverso le quali sia possibile inserire informazioni mediche relative a un determinato paziente (esame obiettivo, anamnesi, misurazioni di parametri vitali, eventuali arresti cardio-circolatori, lesioni riportate, prestazioni effettuate e farmaci o liquidi somministrati).

4.2.1 Wireframe realizzati

Come già accennato, sono stati realizzati dei wireframe relativi alle principali pagine dell'applicazione e questi sono stati utilizzati per ricevere i primi feedback da parte del personale medico. Per farlo è stato usato Balsamiq, un software per il wireframing che aiuta a lavorare in modo veloce [36].

Di seguito saranno riportati i wireframe ottenuti, che sono specifici per un target mobile. Questo perché si tratta di un'applicazione destinata a dispositivi mobile e, come visto nel capitolo 3, se venisse portata su un dispositivo non mobile sarebbe necessario replicare l'interfaccia utente.

Per quanto riguarda la pagina iniziale (wireframe a sinistra, figura 4.6) , attraverso questa un medico dovrà avere la possibilità di attivare una missione. Per farlo, però, egli dovrà inserire informazioni che permettano la sua identificazione (nella parte alta della pagina) e dati relativi alla richiesta di intervento ricevuta dalla centrale operativa (nella parte bassa della pagina).

[6]**Flow-chart**: Diagramma di flusso, che mostra le possibili sequenze di passi presenti all'interno di una procedura.

Una volta che la missione sarà attivata con successo, verrà visualizzata la pagina adibita alla sua gestione (wireframe a destra, figura 4.6). In questa troviamo due parti principali: quella superiore, che permetterà di visualizzare e modificare (anche attraverso pagine di dettaglio) le informazioni generali relative a una missione, e quella inferiore, che consentirà l'accesso a pagine per l'inserimento di informazioni cliniche relative a un paziente. Prima di accedere ad esse, sarà però necessario indicare il paziente a cui si fa riferimento. Ovviamente dovranno essere realizzate più pagine di dettaglio per le informazioni cliniche, e ognuna di esse sarà pensata in relazione ai dati da raccogliere.

Figura 4.6: Wireframe relativi alla pagina iniziale e a quella della missione

Nella figura 4.7 sono riportati i wireframe che fanno riferimento ad alcune delle pagine di dettaglio da implementare: la prima relativa alla dinamica dell'evento che ha causato l'intervento di soccorso, la seconda che permette di memorizzare le misurazioni di parametri vitali di un paziente e la terza che consente di salvare l'esame obiettivo di un paziente. In ognuna di ese si trovano due bottoni, salva e indietro, che permettono di tornare alla gestione della missione, rispettivamente salvando e cancellando i dati inseriti.

Come evidenziato nell'analisi dei requisiti non funzionali, è fondamentale che l'interfaccia sia facile da usare per l'utente. In questo senso sarà necessario

utilizzare alcuni accorgimenti, come ad esempio alcuni elementi grafici che permettano la scelta di un valore da una lista di opzioni predefinite (presenti sia nel primo sia nel terzo wireframe della figura 4.7).

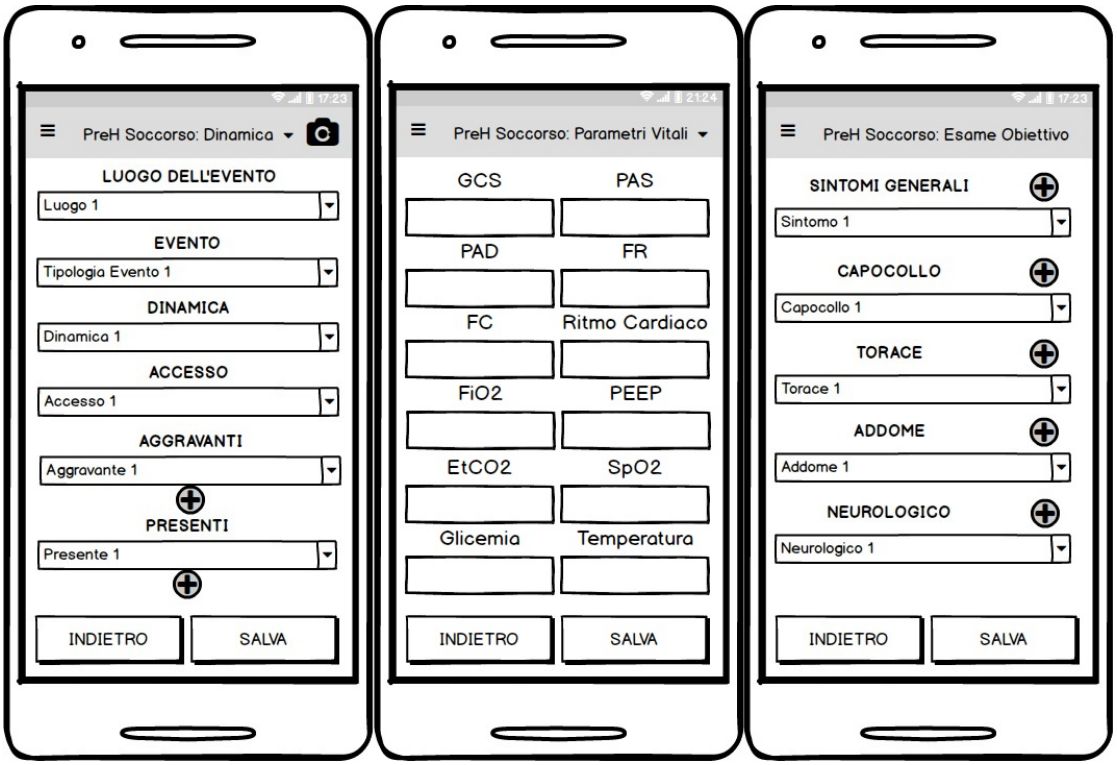

Figura 4.7: Wireframe relativi alle pagine di dettaglio

4.3 Modellazione dei dati trattati

Nell'analisi dei requisiti si parla delle informazioni che il sistema dovrà raccogliere (cosa si vuole memorizzare). Perciò ora, in fase di progettazione, si deve definire un modello che rappresenti i dati (come saranno memorizzati).

I servizi web presenti in TraumaTracker memorizzano le informazioni relative ad un determinato trauma usando MongoDB, un database NoSQL[7] orientato ai documenti. Questi sono in stile JSON[8] e adottano uno schema dinamico, che rende semplice e veloce l'integrazione dei dati alle applicazioni.

[7] **Database NoSQL**: Database che propone una modellazione e un'organizzazione dei dati completamente diversa rispetto ai classici database relazionali (SQL).

[8] **JSON**: JavaScript Object Notation, è un formato adatto all'interscambio di dati tra applicazioni client/server.

Seguendo una linea di continuità con quanto fatto in TraumaTracker, si è scelto di adottare uno schema non relazionale per gestire i dati raccolti dall'applicazione. Verrà definito un modello di dati non relazionale e indipendente dalla tecnologia utilizzata, che sarà mantenuto all'interno dell'applicazione e potrà essere inviato al web service con cui essa dovrà comunicare.

L'approccio object-oriented proposto da TypeScript si adatta molto bene al modello non relazionale introdotto, poiché consente la definizione di classi che contengono i concetti principali individuati nel dominio e che permettono di istanziare oggetti facilmente convertibili in documenti JSON.

In base ai dati da raccogliere, è stato individuato un modello per rappresentare il dominio. Nel diagramma delle classi[9] presente sotto (4.8) è mostrata una rappresentazione ad alto livello di questo modello (in cui ci sono le principali entità e le relazioni tra esse, mentre vengono trascurati gli attributi).

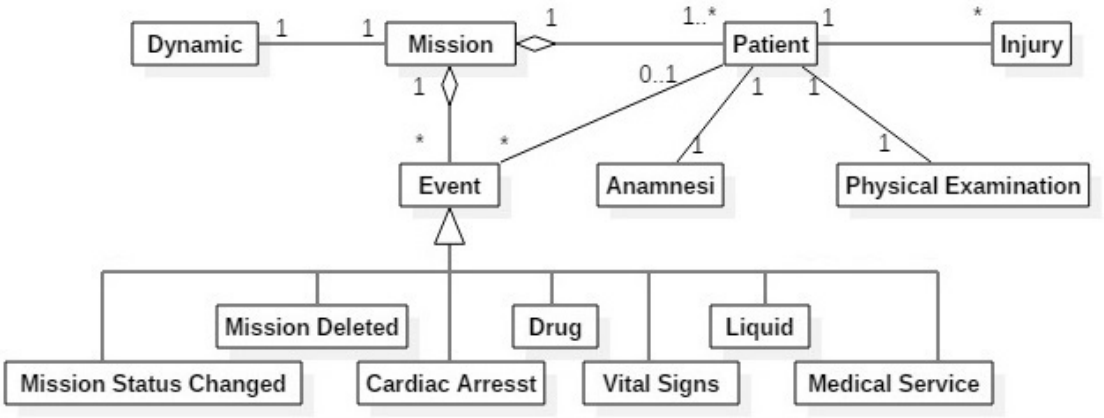

Figura 4.8: Diagramma delle classi che rappresenta in modo generico il modello

Le entità più rilevanti all'interno del modello rappresentato sono:

- **Missione**

  Rappresenta il concetto principale del dominio e contiene tutte le informazioni relative ad un intervento di soccorso. Tra gli attributi di una missione ci sono il medico responsabile, l'eventuale infermiere che lo accompagna, il mezzo di soccorso, il codice relativo alla richiesta di intervento, la data e l'ora di attivazione e altre informazioni significative. Ad ogni missione è associata un'entità, la *Dinamica*, che contiene i dati relativi all'avvenimento che ha causato la missione di soccorso. Inoltre fanno parte di una missione uno o più *Pazienti* e una serie di *Eventi*.

[9]**Diagramma delle classi**: In UML è un diagramma che consente di descrivere vari tipi di entità, con le loro caratteristiche e le eventuali relazioni tra esse.

- **Paziente**

  Rappresenta un paziente coinvolto nella missione di soccorso. Ogni paziente possiede un'identificativo all'interno della missione, e tra i suoi attributi sono presenti informazioni generali e relative al suo stato di salute (dati anagrafici, gravità della sua situazione, ...). Ad ogni paziente sono associate informazioni mediche invariabili durante la missione di soccorso: *Anamnesi*, *Esame obiettivo* e *Lesioni* riportate.

- **Evento**

  Rappresenta un tipo di avvenimento che si è verificato in una missione, registrato dal medico attraverso l'applicazione e di cui è importante conoscere data e ora. Ci sono più tipologie di evento: tra essi ne troviamo alcuni "generali" relativi alla missione e altri che fanno riferimento ad uno specifico paziente, e che sono quindi associati ad esso.

  In base alla sua tipologia l'evento è caratterizzato da attributi differenti, e i possibili tipi sono: *Aborto della missione*, *Cambiamento di stato della missione*, *Arresto cardio-cirolatorio*, *Farmaco o liquido somministrato*, *Prestazione medica effettuata* e *Misurazione di parametri vitali*.

4.4 Progettazione architetturale

Avendo introdotto sia le funzionalità che il sistema dovrà avere sia le tecnologie da usare per svilupparlo (NativeScript & Angular), è possibile definire uno schema architetturale di massima dell'applicazione.

Essendo l'architettura di riferimento quella di Angular, vista in precedenza, il sistema sarà basato su componenti e servizi. In particolare ogni pagina sarà realizzata attraverso un componente (e, eventualmente, altri componenti al suo interno), che si occuperà di gestire le interazioni dell'utente. Le funzionalità specifiche, per la gestione di una missione e per il tracciamento di tutte le informazioni necessarie, saranno implementate nei servizi.

Ogni servizio viene creato per assolvere particolari compiti ma, per semplificare la logica inserita nei componenti, sarebbe ideale se questi potessero ricorrere ad un unico servizio (chiamando un suo metodo) anche quando devono eseguire un'operazione articolata. Ad esempio, se un componente dovesse attivare una missione, non dovrebbe preoccuparsi di eseguire una chiamata al web service ma dovrebbe essere il servizio ad effettuarla automaticamente (non appena il componente chiama il metodo per l'attivazione) e, se il componente volesse cambiare lo stato della missione, dovrebbe limitarsi a chiamare il metodo di un servizio senza richiedere la posizione GPS a un altro servizio.

Questo si potrebbe realizzare con un servizio che implementa tutte le funzionalità richieste dalla view (e quindi dai componenti), ma sarebbe una pessima scelta dal punto di vista della modularità. Una possibile soluzione è costituita dall'utilizzo del Facade Pattern, un pattern di progettazione software che fornisce una semplice interfaccia attraverso la quale accedere a sottosistemi che implementano funzionalità complesse e che è spesso usato in Angular [37]. In questo caso specifico, applicare il Facade Pattern equivale a creare più servizi che implementino compiti specifici e un servizio "di facciata" che utilizzi le loro funzionalità fornendo una semplice interfaccia ai componenti.

Nella figura 4.9 è riportato uno schema architetturale di massima dell'applicazione. Sono presenti i componenti che realizzeranno le pagine descritte in precedenza (4.2), tra cui si nota il *MissionComponent* (relativo alla pagina principale di gestione di una missione). *ExamsComponent* e *EventsComponent* fanno riferimento a tutte quelle pagine che permettono, rispettivamente, di registrare informazioni mediche di un paziente e eventi in cui è coinvolto.

Per quanto riguarda i servizi tutti i componenti dell'applicazione utilizzeranno il *MissionService*, che sarà iniettato in essi e costituirà il servizio "di facciata". Infatti, seguendo il Facade Pattern, a sua volta il *MissionService* userà altri servizi che implementeranno funzionalità specifiche.

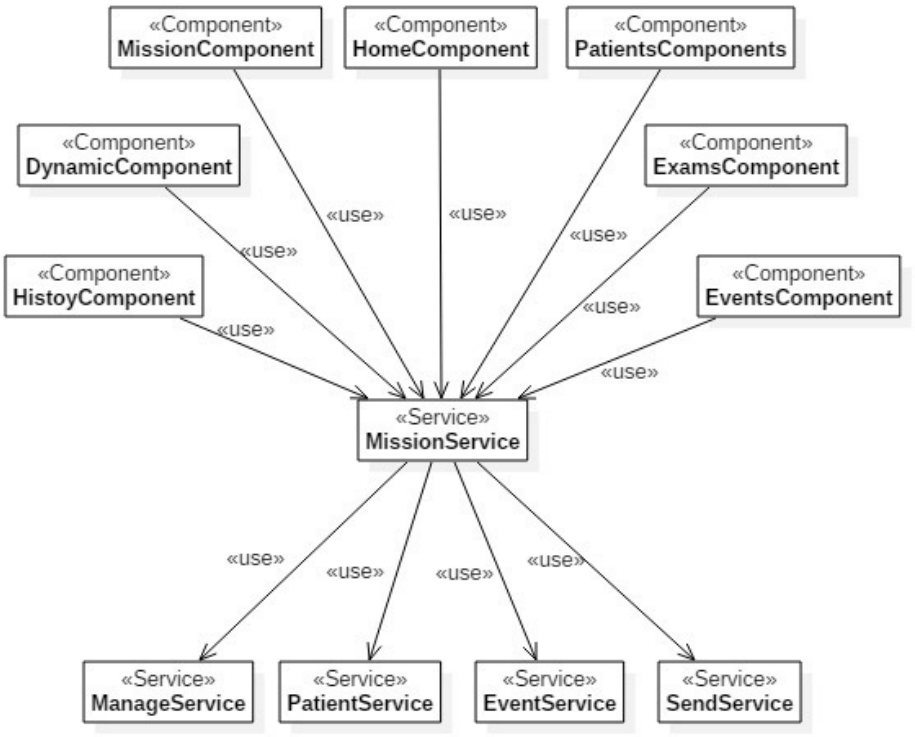

Figura 4.9: Schema architetturale dell'applicazione PreH Soccorso

4.5 Sistema implementato

Nell'implementazione del sistema sono state seguite le linee guida per lo sviluppo presenti nella documentazione ufficiale di Angular, che contengono convenzioni sulla sintassi e sulla struttura dell'applicazione [38].

Di seguito sarà presentata l'implementazione svolta, descrivendo nell'ordine i seguenti punti: organizzazione del progetto, realizzazione dell'interfaccia utente, gestione dei dati raccolti e risultato ottenuto.

4.5.1 Organizzazione del progetto

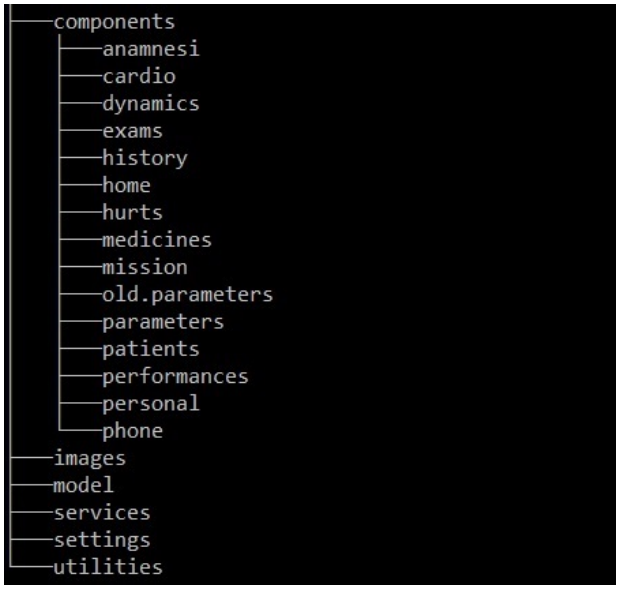

Figura 4.10: Organizzazione in directory del codice implementato

Il progetto realizzato segue la struttura di riferimento vista per NativeScript & Angular (4.1). Tutto il codice implementato, ad esclusione di piccoli accorgimenti usati per configurare il progetto (dichiarazione dei componenti, registrazione dei servizi, ...), è stato inserito nella cartella *app*. Al suo interno, come mostrato dalla figura 4.10, sono state inserite varie sottodirectory:

- *Components*, in cui sono presenti i componenti implementati. Per ognuno di essi è stata creata un'apposita cartella, contenente un file TypeScript per la logica, un file XML per il template e un file CSS per lo stile.
- *Services*, che contiene tutti i servizi in cui è stata sviluppata la logica dell'applicazione.

- *Model*, in cui si trovano le definizioni delle classi TypeScript usate per organizzare i dati gestiti dal sistema secondo il modello individuato nella fase di progettazione.

- *Settings*, contenente dei moduli TypeScript in cui sono specificate opzioni usate dai componenti per mettere a disposizione dell'utente una serie di possibili scelte predefinite.

- *Utilities*, che contiene dei moduli TypeScript che implementano funzionalità legate all'interfaccia utente, e quindi non incapsulabili in un servizio, ma ricorrenti nell'applicazione (come, ad esempio, la creazione di finestre di dialogo molto simili tra loro).

- *Images*, in cui si trovano immagini e icone usate nell'applicazione.

4.5.2 Interfaccia utente

Come già ampiamente introdotto, l'interfaccia utente dell'applicazione è stata realizzata attraverso una serie di componenti. A causa della diversa natura delle pagine da implementare, che prevedono l'inserimento o la visualizzazione di informazioni differenti le une dalle altre, si è scelto di realizzare ogni pagina attraverso un componente. Non è stato creato nessun componente generico, perché questo si sarebbe limitato a contenere la barra di stato superiore e, solo in alcuni casi, qualche tasto di navigazione.

Nelle classi che implementano i componenti è stata inserita la logica che gestisce le interazioni con l'utente. Per realizzare comportamenti specifici, sono stati utilizzati gli eventi del ciclo di vita dei componenti Angular.

Per ogni componente è stato inoltre realizzato un template XML, inserito nella stessa cartella della classe TypeScript e a cui essa fa riferimento con il metadato *templateUrl*, che specifica la struttura della pagina che sarà realizzata. In questi template (pur potendo ricorrere a elementi grafici specifici delle piattaforme Android e iOS) sono stati utilizzati soltanto gli elementi grafici messi a disposizione da NativeScript, nell'ottica di scrivere codice completamente indipendente dalla piattaforma [39]. La realizzazione di questi template ha goduto anche dei benefici portati dal data binding e dalle direttive di Angular, che hanno permesso rispettivamente di associare elementi di modello alla *view* e di creare pagine caratterizzate da una struttura dinamica.

Inoltre ad ogni componente sono stati associati uno o più file CSS, attraverso il metadato *styleUrls*, usati per definire lo stile della pagina. Gli elementi grafici messi a disposizione da NativeScript supportano un set limitato delle proprietà CSS che si possono utilizzare nel web, ma comunque sufficiente per

ottenere una buona interfaccia utente [40]. Nella scrittura dei CSS, per realizzare un'applicazione responsive, la dimensione degli elementi grafici è stata impostata in proporzione alle dimensioni del dispositivo.

Nell'applicazione sono stati realizzati diversi componenti e di seguito, nel frammento di codice 4.1, sarà riportata una parte di un template XML (quello relativo alla pagina di inserimento dei parametri vitali) che costituisce un esempio di quanto fatto. Esso mostra i grandi vantaggi portati dal data binding di Angular, usato in modo ricorrente nei componenti implementati.

Nel template ci sono infatti diversi campi di testo, che permettono l'inserimento dei parametri misurati ad un paziente. Ad ognuno di essi è associato, grazie all'attributo *[(ngModel)]*, un campo di un oggetto (proprietà della classe del componente) che contiene tutti i parametri e istanzia una classe del *Model*.

Il binding bidirezionale di Angular fa sì che, ogni volta che l'utente modifica il valore di un campo di testo, l'oggetto contenente tutti i parametri vitali venga aggiornato. In questo modo, quando l'utente vuole salvare i parametri vitali inseriti, è sufficiente chiamare l'apposita funzione implementata in un servizio passando come argomento l'oggetto appena descritto.

```
<!-- ... -->
<ScrollView orientation="vertical">
    <StackLayout>
        <!-- ... -->
        <StackLayout class="gcsInput">
            <Label text="GCS(1-15) [O(1-4) V(1-5) M(1-6)]"></Label>
            <StackLayout orientation="horizontal">
                <TextField [(ngModel)]="currentVitalSigns.gcsEyes"
                    hint="O" (blur)="onTextChange($event)"
                    keyboardType="number" maxLength="1"></TextField>
                <TextField [(ngModel)]="currentVitalSigns.gcsVerbal"
                    hint="V" (blur)="onTextChange($event)"
                    keyboardType="number" maxLength="1"></TextField>
                <TextField [(ngModel)]="currentVitalSigns.gcsMotor"
                    hint="M" (blur)="onTextChange($event)"
                    keyboardType="number" maxLength="1"></TextField>
                <TextField [(ngModel)]="currentVitalSigns.gcsTotal"
                    hint="TOT" (blur)="onTextChange($event)"
                    editable="false" class="gcsTotal"></TextField>
            </StackLayout>
        </StackLayout>
        <StackLayout orientation="horizontal" class="paInput">
            <StackLayout class="sxCol">
                <Label text="PAS (mmHg)"></Label>
                <TextField [(ngModel)]="currentVitalSigns.pas"
```

```
                hint="PAS" keyboardType="number"
                maxLength="3"></TextField>
        </StackLayout>
        <StackLayout class="dxCol">
            <Label text="PAD (mmHg)"></Label>
            <TextField [(ngModel)]="currentVitalSigns.pad"
                hint="PAD" keyboardType="number"
                maxLength="3"></TextField>
        </StackLayout>
    </StackLayout>
    <StackLayout orientation="horizontal" class="cardioInput">
        <StackLayout class="sxCol">
            <Label text="FC (bpm)"></Label>
            <TextField [(ngModel)]="currentVitalSigns.fc"
                hint="FC" keyboardType="number"
                maxLength="3"></TextField>
        </StackLayout>
        <StackLayout class="dxCol">
            <Label text="Ritmo Cardiaco"></Label>
            <TextField
                [(ngModel)]="currentVitalSigns.heartRhythm"
                hint="Ritmo" class="cardioField"></TextField>
        </StackLayout>
    </StackLayout>
    <StackLayout orientation="horizontal" class="fInput">
        <StackLayout class="sxCol">
            <Label text="FR (atti/min)"></Label>
            <TextField [(ngModel)]="currentVitalSigns.fr"
                hint="FR" keyboardType="number"
                maxLength="2"></TextField>
        </StackLayout>
        <StackLayout class="dxCol">
            <Label text="FiO2 (l/min)"></Label>
            <TextField [(ngModel)]="currentVitalSigns.fio2"
                hint="FiO2" keyboardType="number"
                maxLength="2"></TextField>
        </StackLayout>
    </StackLayout>
    <StackLayout orientation="horizontal" class="pressionInput">
        <StackLayout class="sxCol">
            <Label text="PEEP (cmH20)"></Label>
            <TextField [(ngModel)]="currentVitalSigns.peep"
                hint="PEEP" keyboardType="number"
                maxLength="2"></TextField>
```

```
            </StackLayout>
            <StackLayout class="dxCol">
                <Label text="EtCO2 (mmHg)"></Label>
                <TextField [(ngModel)]="currentVitalSigns.etco2"
                    hint="EtCO2" keyboardType="number"
                    maxLength="2"></TextField>
            </StackLayout>
        </StackLayout>
        <StackLayout orientation="horizontal" class="sgInput">
            <StackLayout class="sxCol">
                <Label text="SpO2 (%)"></Label>
                <TextField [(ngModel)]="currentVitalSigns.spo2"
                    hint="SpO2" keyboardType="number"
                    maxLength="3"></TextField>
            </StackLayout>
            <StackLayout class="dxCol">
                <Label text="Glicemia (mg/dl)"></Label>
                <TextField [(ngModel)]="currentVitalSigns.glycaemia"
                    hint="Glicemia" keyboardType="number"
                    maxLength="3"></TextField>
            </StackLayout>
        </StackLayout>
        <StackLayout orientation="horizontal" class="tempdolInput">
            <StackLayout class="sxCol">
                <Label text="Temperatura"></Label>
                <TextField
                    [(ngModel)]="currentVitalSigns.temperature"
                    hint="Temp" keyboardType="number" maxLength="4"
                    step="0.01"></TextField>
            </StackLayout>
            <StackLayout class="dxCol">
                <Label text="Dolore (0-10)"></Label>
                <TextField [(ngModel)]="currentVitalSigns.pain"
                    hint="Dolore" keyboardType="number"
                    maxLength="2"></TextField>
            </StackLayout>
        </StackLayout>
        <!-- ... -->
</ScrollView>
<!-- ... -->
```

Listato 4.1: Parte del template XML relativo al *ParametersComponent*

Navigazione tra le viste

Per realizzare l'interfaccia utente, è necessario gestire anche la navigazione tra le varie pagine. NativeScript consente di farlo usando il router di Angular, che è iniettabile nei componenti come un qualunque altro servizio. In questo progetto è stata scelta la configurazione fornita dal *page-router-outlet*, poiché mette a disposizione una navigazione specifica per il target mobile.

Per configurare il router è necessario specificare, nel componente root dell'applicazione (ossia quello contenuto nel file *app.component.ts*, che appare sullo schermo quando viene lanciata l'applicazione) il template *"<page-router-outlet></page-router-outlet>"*. Inoltre nel progetto va inserito un file, *app.routing.ts*, che contiene la dichiarazione di tutti i componenti accessibili dal router e un percorso associato ad essi, che permette di raggiungerli. Il componente che contiene un percorso vuoto sarà quello mostrato inizialmente.

Dopo aver configurato il router di Angular in questo modo, è emersa una problematica: il router fa sì che, premendo il tasto indietro messo a disposizione del sistema operativo (piccolo e disponibile in varie zone dello schermo a seconda del dispositivo) e senza chiedere alcuna conferma all'utente, si torni alla pagina visitata in precedenza. Questo comportamento è decisamente inadatto ad un'applicazione come quella progettata, in cui lo spostamento alla pagina precedente indica una scelta ben precisa dell'utente (come l'aborto di una missione o l'annullamento di alcune informazioni mediche inserite).

Si è quindi deciso di eliminare questo comportamento, e di consentire all'utente la navigazione solo attraverso i bottoni messi a disposizione dall'interfaccia utente. Per farlo è stata usata la classe *RouterExtension*, un'estensione del classico router di Angular creata appositamente per NativeScript, che consente di specificare alcune opzioni (tra cui la cancellazione della cronologia).

Inoltre, dovendosi trattare di un'applicazione robusta, è necessario che venga mantenuta la pagina corrente in caso di chiusura. Per farlo è stato usato il *localStorage*, un plugin installabile tramite NPM e creato per riprodurre su NativeScript le funzionalità fornite dal *localStorage* messo a disposizione da HTML5. *LocalStorage* permette di salvare informazioni localmente sulla memoria del dispositivo, attraverso delle associazioni chiave-valore [41].

Di seguito (4.2) è riportato il codice relativo al *NavigationService*, il servizio creato nell'applicazione e utilizzato al posto del classico router di Angular. Il metodo *initialize()* viene richiamato all'interno del *ngOnInt()* dell'*AppComponent*, in modo da riportare sempre l'applicazione all'ultimo componente visitato. Nel codice si fa riferimento ad *ApplicationPage*, una classe enumerata contenente i percorsi relativi a tutte le pagine presenti nell'applicazione. Il *RouterExtension* descritto in precedenza è stato iniettato nel servizio creato, in modo che questo possa utilizzarlo per implementare la navigazione.

```
import { Injectable } from "@angular/core";
import { ApplicationPage } from "../model/application.page";
import { RouterExtensions } from "nativescript-angular/router";
const localStorage = require( "nativescript-localstorage" );

const CURRENT_PAGE = "currentPage";
@Injectable()
export class NavigationService {
    public constructor(private router: RouterExtensions) {}
    private getCurrentPage(): ApplicationPage {
        var page = localStorage.getItem(CURRENT_PAGE);
        if(page == null) {
            return ApplicationPage.HOME;
        } else {
            return page;
        }
    }
    public initialize(): void {
        var page = this.getCurrentPage();
        this.navigate(page);
    }
    public navigate(page: ApplicationPage): void {
        localStorage.setItem(CURRENT_PAGE, page);
        this.router.navigate([page], { clearHistory: true });
    }
}
```

Listato 4.2: *NavigationService* creato per gestire la navigazione tra le viste

4.5.3 Raccolta e gestione dei dati

Nell'applicazione realizzata tutte le operazioni più complesse, che permettono la raccolta e la gestione dei dati, sono implementate con dei servizi.

In particolare, seguendo l'architettura definita per il sistema (4.4), è stato usato il Facade Pattern ed è stato creato un servizio (*MissionService*) che mette a disposizione dei componenti tutti gli attributi e le proprietà di cui hanno bisogno (diagramma 4.11). Il *MissionService* non implementa direttamente nessuna delle funzionalità offerte, che sono realizzate da altri servizi.

Ad esempio, nel *MissionService* sono presenti due proprietà, *currentMission* e *currentPatientIndex*, che contengono rispettivamente l'oggetto che rappresenta la missione corrente e l'indice del paziente selezionato (se, per esempio, dall'applicazione ci si sposta in una pagina di dettaglio che permette l'inseri-

mento di informazioni mediche, è necessario selezionare un certo paziente). Grazie a queste due proprietà i componenti hanno la possibilità di accedere direttamente ad un qualsiasi campo della missione in corso.

Figura 4.11: Diagramma delle classi relativo al *MissionService* implementato

Saranno ora descritti i "sotto-servizi" realizzati e usati dal *MissionService*:

- **ManageService**

  ManageService è il servizio che implementa le operazioni di gestione di una missione come, ad esempio, la sua attivazione, il suo cambiamento di stato o la sua cancellazione (aborto missione).

Al suo interno è utilizzato il plugin *localStorage*, già visto in precedenza, che permette di salvare localmente la missione in corso (codificata come stringa JSON). Così, anche in caso di chiusura dell'applicazione, vengono conservati i dati inseriti fino a quel momento.

Altri due plugin che vengono usati in questo servizio sono *nativescript-camera* [42] e *nativescript-geolocation* [43], che consentono di accedere rispettivamente alla fotocamera e al sensore GPS del dispositivo (per scattare foto relative alla dinamica della missione e per memorizzare le coordinate rilevate quando si verifica un cambiamento di stato).

Nello snippet[10] 4.3 è riportato un metodo d'esempio presente nel *ManageService*, ossia quello che consente di cambiare lo stato di una missione. Da esso si può notare come, attraverso le API messe a disposizione dal plugin di NativeScript, sia semplice accedere alla posizione rilevata dal sensore GPS. Se ciò non è possibile il servizio ritorna comunque un cambiamento di stato, senza però inserire i dati relativi alla posizione.

L'oggetto ritornato è del tipo *Promise<MissionStatusChanged>* perché l'accesso al GPS prevede un'operazione asincrona. Di conseguenza, il cambiamento di stato della missione sarà restituito non appena sarà stato completato l'accesso al GPS (per questo si parla di "promessa").

```
public changeMissionStatus(mission: Mission):
    Promise<MissionStatusChanged> {
    var change: string = this.getNextChange(mission);
    if(this.getNextStatus(mission) == MissionStatus.END) {
        localStorage.removeItem(CURRENT_MISSION);
    }
    return new Promise((resolve, reject) => {
        getCurrentLocation({
            desiredAccuracy: 3, updateDistance: 10, maximumAge:
                20000, timeout: 20000
        }).then(loc => {
            resolve(new MissionStatusChanged(change,
                loc.latitude, loc.longitude, loc.altitude));
        }, error => {
            resolve(new MissionStatusChanged(change, null, null,
                null));
        });
    });
}
```

Listato 4.3: Metodo che realizza il cambiamento di stato della missione

[10]**Snippet**: Frammento di codice ritenuto significativo.

- **PatientService**

  PatientService è il servizio che implementa tutte le operazioni relative ad un paziente come, ad esempio, l'aggiunta di un nuovo paziente, la selezione di un paziente, la memorizzazione delle lesioni che ha riportato o lo scatto di una fotografia che lo rappresenta.

  Nel *PatientService* sono usati i plugin *localStorage*, per memorizzare il paziente selezionato, e *nativescript-camera*, per accedere alla camera del dispositivo. Di seguito è riportato uno snippet (4.4) con il metodo del *PatientService* che permette di fotografare un paziente. Si può notare come, all'interno della missione, venga mantenuto il percorso locale in cui viene salvata l'immagine (utilizzabile per accedere alla foto).

```
public takePatientImage(mission: Mission, index: number):
    Promise<Mission> {
    return new Promise((resolve, reject) => {
        requestPermissions().then(() => {
            takePicture({keepAspectRatio: true, saveToGallery:
                false, cameraFacing: "front"}).then((imageAsset:
                ImageAsset) => {
                var path: string = null;
                if(isAndroid) {
                    path = imageAsset.android;
                } else if(isIOS) {
                    path = imageAsset.ios;
                }
                mission.patients[index].images.push(path);
                resolve(mission);
            }).catch((error: Error) => {
                resolve(null);
            });
        });
    });
}
```

Listato 4.4: Metodo che permette di scattare fotografie ad un paziente

- **EventService**

  EventService è il servizio che si occupa della memorizzazione e della visualizzazione di tutti gli *Eventi* verificatisi durante una missione. *EventService* si basa sui diversi tipi di *Evento* definiti in fase di modellazione: misurazioni di parametri vitali, farmaci o liquidi somministrati, prestazioni effettuate e eventuali arresti cardio-circolatori.

- **SendService**

  SendService è il servizio, definito e inserito nell'applicazione ma i cui metodi non sono ancora stati implementati, che avrà il compito di inviare i dati raccolti al servizio web descritto nella fase di analisi dei requisiti, in modo che tutti i dati tracciati possano essere comunicati in tempo reale alla centrale operativa che coordina il soccorso.

  Il servizio web dovrà mettere a disposizione due semplici API a cui l'applicazione farà riferimento: una per la creazione di una nuova missione e una per l'aggiornamento di una missione. In entrambi i casi sarà inviata una rappresentazione JSON della missione. Perciò il *SendService* dovrà implementare due metodi, *createMission()* e *updateMission()*, che saranno chiamati dal *MissionService* per ogni creazione o aggiornamento.

4.5.4 Risultato ottenuto

In questa sezione sarà descritto il sistema realizzato e il suo funzionamento, mostrando alcuni screenshot relativi alle pagine dell'applicazione.

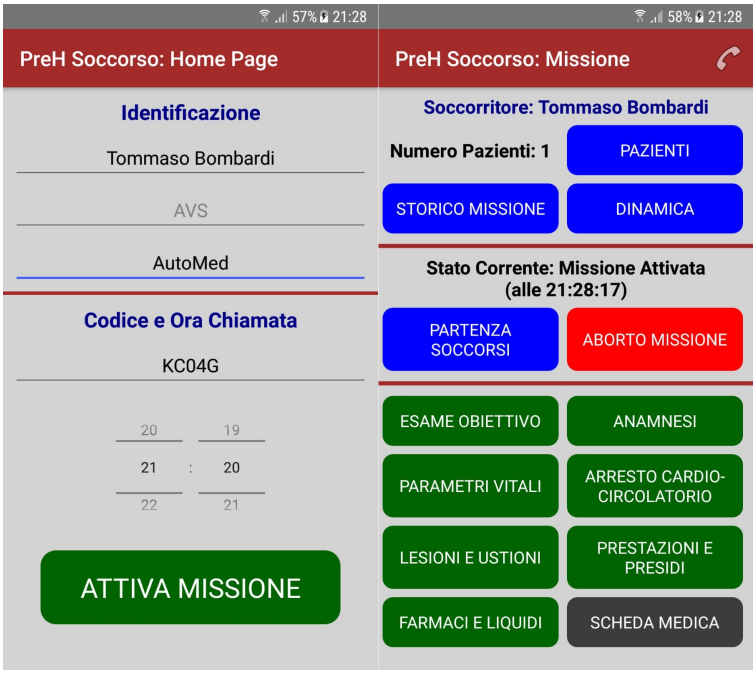

Figura 4.12: Attivazione e gestione della missione

Inizialmente l'applicazione si posiziona nella home page (4.12), dalla quale il medico ha la possibilità di attivare una nuova missione. Per farlo deve obbligatoriamente identificarsi (selezionando il suo nominativo e il mezzo di soccorso

da una lista di valori predefiniti) e inserire il codice ricevuto al momento della chiamata. Inoltre, può opzionalmente specificare il nome di un AVS (infermiere) che lo accompagna e l'orario in cui è stato richiesto l'intervento di soccorso (se diverso da quello di apertura dell'applicazione).

Una volta che è stata attivata una missione, viene visualizzata la pagina principale dell'applicazione: quella che si occupa della gestione di una missione (4.12). Questa, come si può notare dallo screenshot riportato, è divisa in tre parti: quella superiore che permette di accedere a pagine di dettaglio relative alla missione, quella centrale che consente di tracciare i cambiamenti di stato della missione e quella inferiore attraverso la quale è possibile accedere alle pagine mediche di dettaglio relative ad un certo paziente.

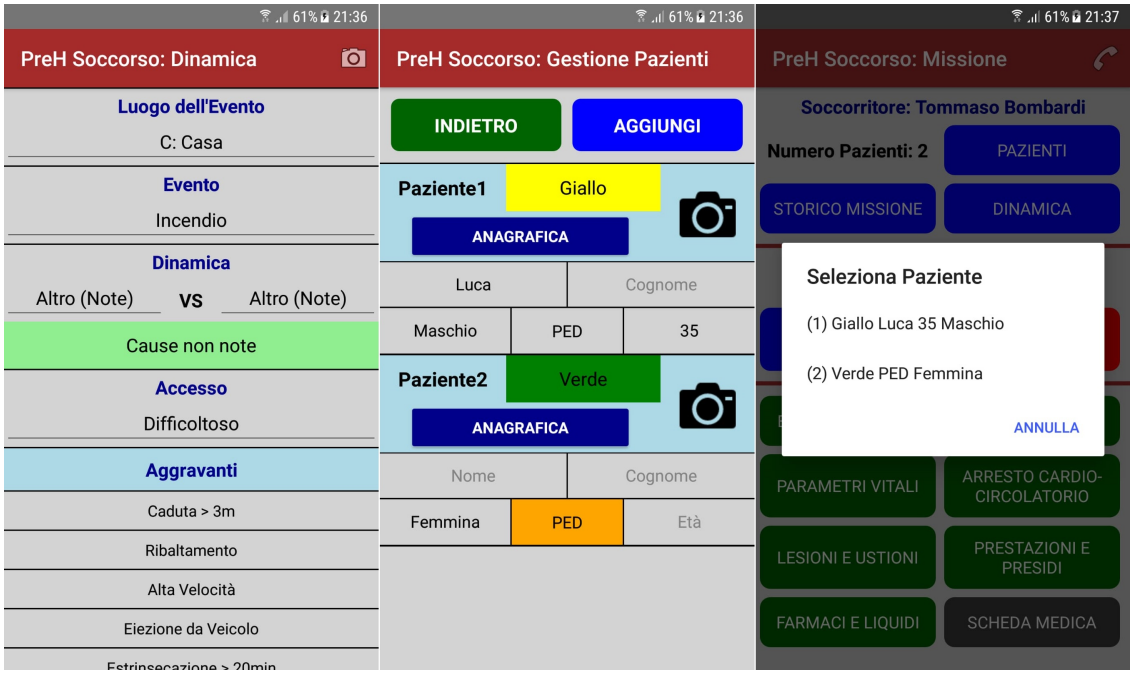

Figura 4.13: Gestione della dinamica dell'evento e dei pazienti

Tra le pagine di dettaglio relative alla missione, vi è quella che consente di inserire la dinamica dell'evento (4.13). Come la quasi totalità delle pagine dell'applicazione, è previsto uno scrolling verticale che permette di visualizzare tutti i contenuti. Nei vari campi è possibile scegliere tra una lista di valori predefiniti e, dove necessario, è presente il campo "altro" per inserire manualmente un valore. Grazie all'apposita icona posizionata sulla *ActionBar* della pagina, è anche possibile scattare una o più foto relative alla dinamica.

Un'altra pagina di fondamentale importanza è quella adibita alla gestione dei pazienti coinvolti in una missione (4.13). Essa permette di aggiungere un

nuovo paziente, di inserire semplici informazioni utili all'identificazione dei pazienti presenti (gravità, nome, cognome, sesso, età e se si tratta di un paziente pediatrico) e, eventualmente, di scattare loro delle fotografie.

Inizialmente la missione prevede la presenza di un solo paziente ma, essendo possibile inserirne altri, l'applicazione dovrà essere in grado di gestire più pazienti contemporaneamente. Perciò, quando dalla pagina principale della missione si tenta di accedere a una pagina adibita all'inserimento o alla visualizzazione di informazioni mediche relative a un paziente, se i pazienti coinvolti nella missione sono più di uno il sistema obbliga l'utente a indicare quello a cui si sta facendo riferimento (con una finestra di dialogo, vedi 4.13).

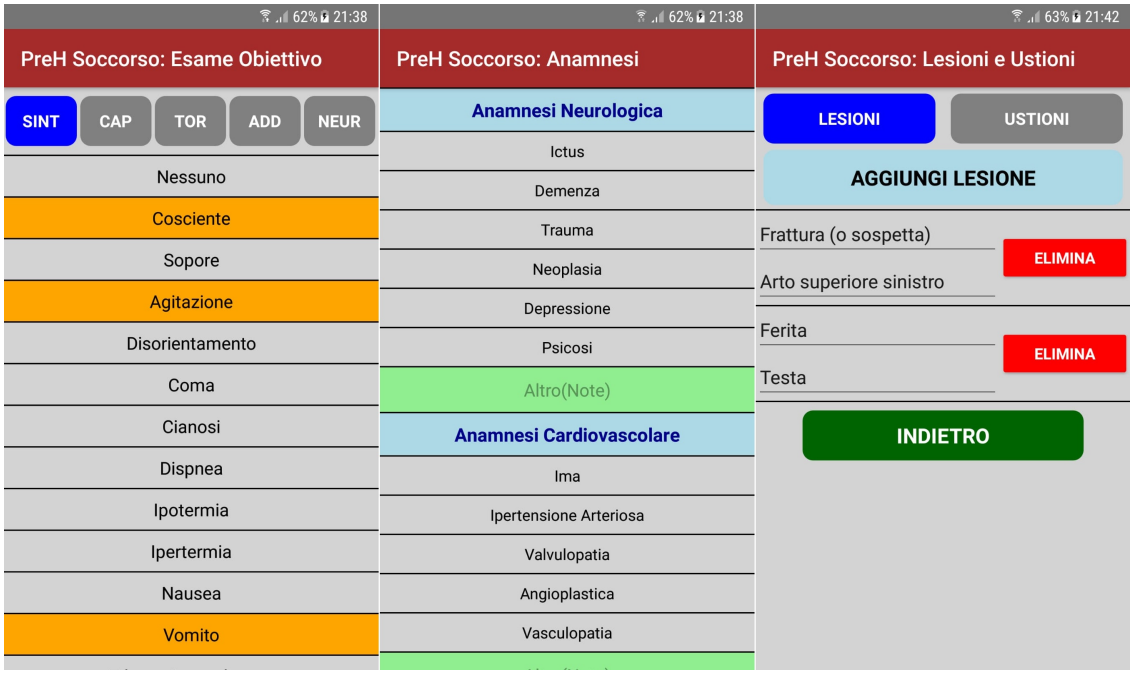

Figura 4.14: Inserimento e visualizzazione di informazioni mediche

Tra le pagine di dettaglio che consentono di inserire e visualizzare informazioni mediche relative ad un paziente, ne è stata realizzata una relativa all'esame obiettivo, una dedicata all'anamnesi e una per lesioni e ustioni (4.14). Accedendo a queste pagine è possibile visualizzare e modificare i valori inseriti in precedenza, poiché si tratta di informazioni che, nel corso di una missione, sono indipendenti dal tempo e vengono memorizzate una sola volta.

Per quanto riguarda anamnesi e esame obiettivo, è possibile selezionare uno o più valori da una lista predefinita e, per facilitare l'utente, i possibili valori sono stati suddivisi in categorie predefinite (indicate dai medici).

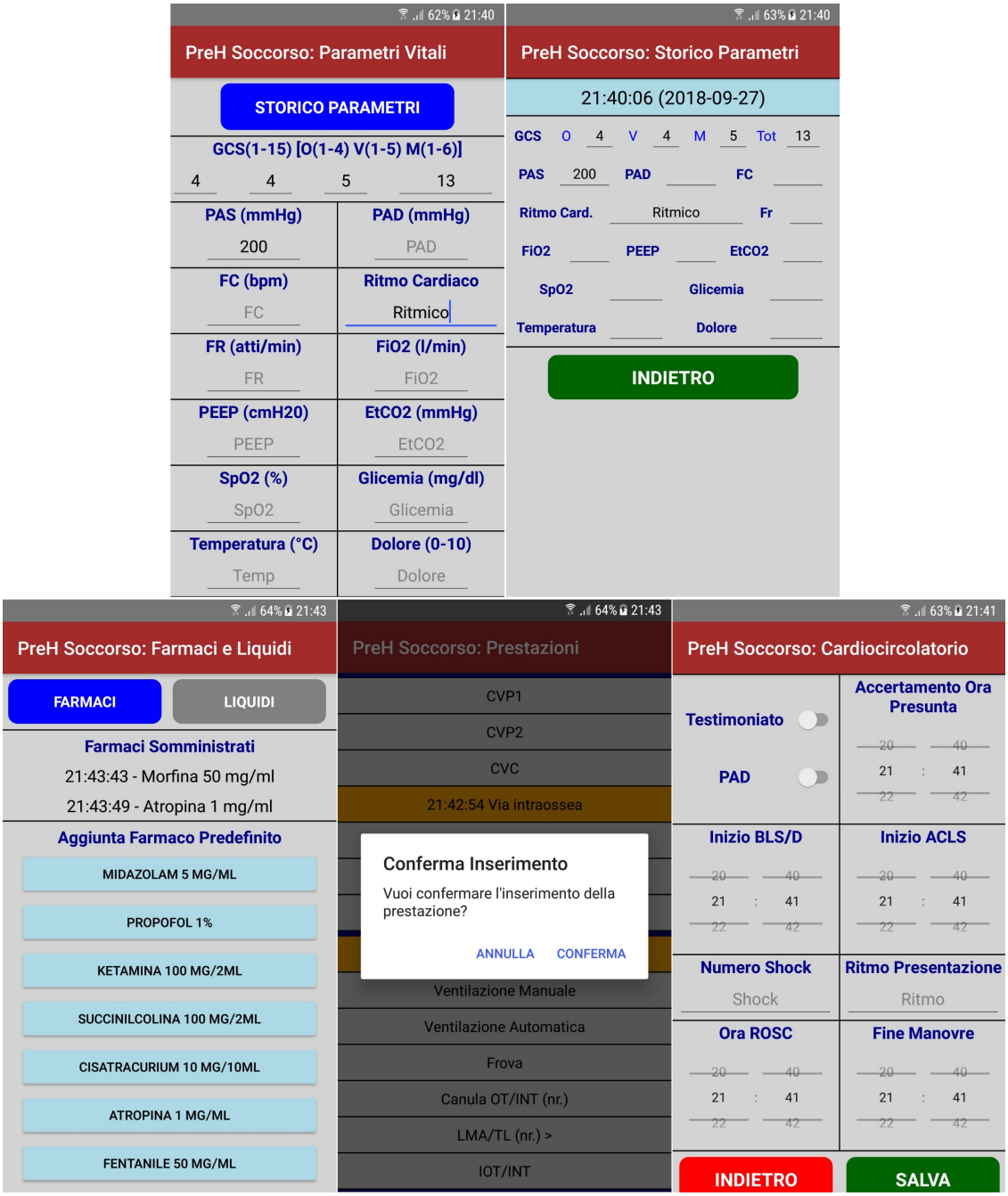

Figura 4.15: Inserimento e visualizzazione di eventi medici

Oltre alle pagine per l'inserimento e la visualizzazione di informazioni mediche, sono state create diverse pagine per il tracciamento dei cosiddetti *Eventi* medici (per i quali è importante memorizzare l'orario in cui si verificano).

È presente una pagina per l'inserimento delle misurazioni di parametri vitali relative a un paziente e, da essa, è possibile accedere a una pagina in cui sono mostrate, a partire dalla più recente, le misurazioni precedenti (4.15).

Inoltre, è stata inserita una pagina che permette di tracciare tutti i farmaci e i liquidi somministrati al paziente nel corso della missione. Questa permette di visualizzare le sostanze somministrate in precedenza e di registrarne altre, attraverso una lista di farmaci o liquidi predefiniti oppure specificando la descrizione della sostanza, la sua quantità e l'unità di misura (4.15).

Sono infine state realizzate altre due pagine, che consentono di tracciare rispettivamente le prestazioni effettuate su un determinato paziente e un eventuale arresto cardiocircolatorio di cui è stato vittima (4.15).

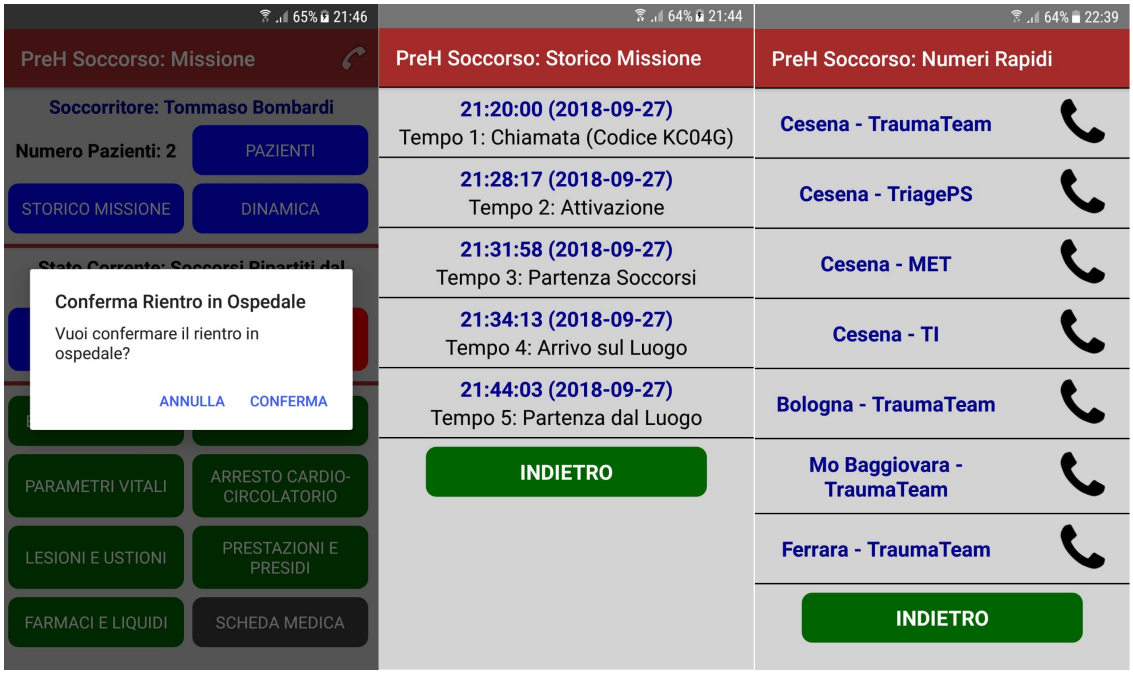

Figura 4.16: Gestione dello stato della missione e numeri rapidi

Come visto in precdenza, dalla pagina principale della missione è possibile registrare il suo cambiamento di stato (cliccando l'apposito tasto e poi confermando attraverso una finestra di dialogo, vedi 4.16).

Inoltre sono state create due pagine, anch'esse accessibili da quella di gestione della missione, contenenti rispettivamente lo storico della missione (costituito dai cambiamenti di stato che si sono verificati e dalle informazioni temporali associate) e una serie di collegamenti che consentono di contattare telefonicamente alcuni dipartimenti degli ospedali della zona (4.16).

Capitolo 5

Validazione e discussione del lavoro svolto

Dopo aver descritto la progettazione e lo sviluppo del sistema (nel capitolo 4), in questa parte dell'elaborato saranno riportate tutte le attività che sono state svolte per accertare che il software realizzato rispettasse i requisiti nel modo dovuto e sarà evidenziato quanto è emerso.

Nei casi in cui queste attività abbiano evidenziato delle mancanze o dei difetti nell'applicazione sviluppata, saranno descritte le soluzioni che sono state adottate per risolvere le problematiche sottolineate.

5.1 Feedback ricevuti durante la realizzazione dell'applicazione

Il modello di sviluppo che ha portato alla produzione del sistema presentato in questa tesi di laurea non è un modello lineare (come ad esempio quello a cascata, in cui l'output di ogni fase costituisce l'input della successiva), ma è un modello incrementale basato sulla prototipazione evolutiva.

Infatti, prima di giungere alla versione presentata, sono stati prodotti una serie di prototipi[1] (a partire dai wireframe dell'interfaccia utente, fino ad arrivare a versioni del sistema contenenti diverse funzionalità), che hanno consentito di valutare quanto era stato fatto fino a quel momento. L'evoluzione di questi prototipi ha permesso di ottenere il sistema che è stato presentato.

Questi prototipi hanno sì consentito di giudicare le funzionalità del sistema, ma sono stati utili principalmente nella valutazione e nel miglioramento della sua usabilità. Come evidenziato dall'analisi dei requisiti non funzionali, il

[1] **Prototipo**: Modello parziale del sistema da sviluppare, che simula o esegue delle funzionalità del sistema finale e che è realizzato per valutarne le caratteristiche [6].

sistema avrebbe infatti dovuto essere semplice da usare per facilitare quanto più possibile i medici impegnati nella missione di soccorso.

La valutazione dei prototipi ha consentito di migliorare l'usabilità del sistema poiché è stata fatta confrontandosi direttamente con il personale medico, seguendo un approccio allo sviluppo che si può definire user-centered[2].

In particolare nel corso dell'estate sono stati organizzati due incontri con il dottor Albarello e alcuni suoi collaboratori (avvenuti in seguito a quello svolto per definire i requisiti), nei quali sono stati presentati loro due diversi prototipi dell'applicazione (uno successivo all'altro) e sono stati raccolti feedback che hanno permesso di rendere il sistema più adatto alle loro esigenze.

Nell'ottica di realizzare un sistema "su misura" dell'utente, sono stati molto utili anche i feedback che ho ricevuto confrontandomi col dottor Croatti. Egli, avendo lavorato a stretto contatto con i medici del Trauma Center per sviluppare TraumaTracker, è stato in grado di darmi consigli molto utili e di aiutarmi a creare un sistema con buone caratteristiche di usabilità.

I feedback ricevuti si sono tradotti principalmente in modifiche dell'interfaccia utente. In particolare, i contenuti sono stati raggruppati in modo da poter essere individuati più facilmente, sono stati inseriti elementi grafici per l'interazione con gli utenti dove questi si aspettavano di trovarli e sono stati usati determinati colori per distinguere gli elementi presenti nelle pagine.

5.2 Analisi delle performance

L'applicazione, come messo in luce nell'analisi dei requisiti non funzionali, deve sia implementare tutte le funzionalità necessarie sia farlo in modo efficiente. La scelta di NativeScript è stata fatta proprio in quest'ottica poiché si tratta, come visto, di un framework che consente di realizzare applicazioni mobile con prestazioni di poco inferiori rispetto a quelle native.

Per valutare l'efficienza del sistema realizzato, può essere molto utile un'analisi delle sue performance. In particolare NativeScript, con la versione 3.1, ha introdotto un nuovo strumento: il *timeline*. Esso consente di accedere alla console dei metodi di runtime, realizzando un profilo contenente istanti significativi (tempi di avvio, tempi di estrazione delle risorse, tempo trascorso in una chiamata, ...). *Timeline* può essre abilitato inserendo l'opzione *"profiling": "timeline"* nel file *package.json* presente nella cartella *app*.

In questo modo le informazioni temporali sono visibili sulla console ma, essendo mostrate insieme al resto dell'output, risultano difficili da leggere e analizzare. Perciò NativeScript offre anche *timeline-view*, uno strumento in-

[2]**User-centered design**: Modello usato nella progettazione di sistemi software che mette al primo posto le esigenze dell'utente.

stallabile via NPM e utilizzabile aggiungendo l'opzione *–timeline-view* al comando usato per lanciare l'applicazione (*tns run –platform*). *Timeline-view* genera una rappresentazione HTML dei dati ottenuti [44].

Le performance del sistema realizzato sono state valutate con *Timeline* e *Timeline-view* e, da una prima analisi, è emerso quanto era già stato notato attraverso prove sperimentali: mentre la velocità di navigazione tra le pagine era buona, il tempo necessario per avviare l'applicazione era decisamente eccessivo. In particolare, eseguendo la misurazione dopo aver lanciato il sistema su Huawei P9 (uno smartphone Android di fascia media), questo tempo si collocava in un intervallo compreso tra i 10 e i 12 secondi.

Ovviamente NativeScript non permette di realizzare applicazioni che si avviino con la stessa velocità di quelle native ma, soprattutto nella situazione di emergenza analizzata, non è accettabile una latenza di questo tipo.

5.2.1 Ottimizzazioni eseguite

Avendo riscontrato il problema descritto, ossia l'eccessiva lentezza dell'applicazione nell'avvio, sono state cercate soluzioni in grado di risolverlo.

Dalla documentazione NativeScript, è emerso che una delle principali cause dell'avvio lento delle applicazioni è costituita dal grande numero di operazioni di File I/O necessarie all'avvio. Infatti, ogni volta che nell'applicazione viene richiesto un certo modulo, NativeScript deve recuperare un file dal file system dedicato all'applicazione e, successivamente, eseguirlo attraverso la macchina virtuale JavaScript utilizzata da NativeScript. Questa operazione è particolarmente costosa nei casi in cui sono richiesti molti moduli.

Il modo migliore per ridurre l'I/O di file consiste nel posizionare tutto il codice JavaScript in un numero limitato di file e, in NativeScript, è possibile farlo abilitando il plugin *Webpack*. Esso fornisce uno strumento di compilazione attraverso il quale è possibile ottimizzare le applicazioni [45].

Per utilizzare *Webpack* è necessatio installare il plugin via NPM e, facendolo, viene automaticamente aggiunto al progetto un file *webpack.config.js*, in cui è contenuta la configurazione di questo strumento. Per compilare l'applicazione con il *Webpack*, basta specificare l'opzione *–bundle* nel comando usato per eseguire l'applicazione (*tns run –platform*). Nell'applicazione realizzata si è quindi deciso di utilizzare il *Webpack* e questo, come sarà evidenziato in seguito, ha portato miglioramenti notevoli delle performance.

5.2.2 Risultati raggiunti

L'utilizzo del *Webpack* nella compilazione dell'applicazione ha portato notevoli miglioramenti da un punto di vista delle performance. Questi posso-

no essere notati anche attraverso delle prove sperimentali, poiché il tempo necessario per avviare l'applicazione appare decisamente inferiore.

Compilando con il *Webpack*, le performance sono state valutate di nuovo attraverso *Timeline* e *Timeline-view* e essi hanno confermato i miglioramenti ottenuti. Infatti, sempre su Huawei P9, l'applicazione ha impiegato circa 4,5 secondi per avviarsi e circa 0,2 secondi per effettuare la navigazione tra due pagine. Nella figura 5.1 è riportata una parte della pagina HTML, generata grazie al *Timeline-view*, che mostra i risultati appena descritti.

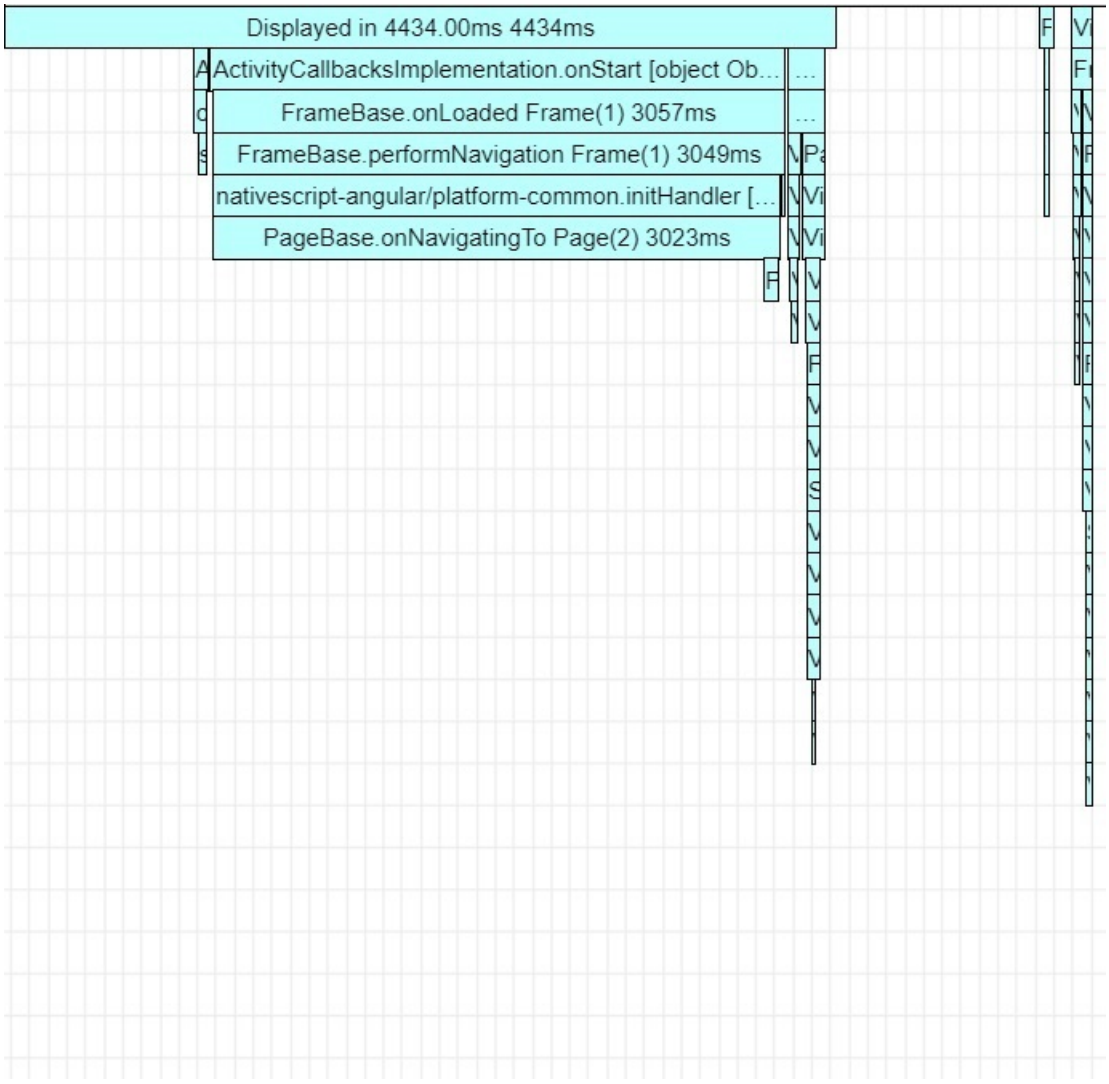

Figura 5.1: Profilo delle performance dell'applicazione ottenuto con *Timeline*

5.3 Test sperimentali

Per verificare il corretto funzionamento del sistema realizzato, esso è stato visionato e testato su diversi dispositivi mobile. In particolare, è stato fatto il deployment dell'applicazione sui seguenti dispositivi:

- Huawei P9, con display da 5,2 pollici.
- Samsung Galaxy A5, con display da 5,2 pollici.
- Samsung Galaxy A6 Plus, con display da 6 pollici.

Il sistema realizzato si è ben comportato su ognuno di questi, e non è stato riscontrato alcun difetto dal punto di vista delle funzionalità.

Dal punto di vista dell'interfaccia utente, eseguire il deployment su dispositivi caratterizzati da schermi di dimensioni diverse ha permesso di notare come l'applicazione sappia adattarsi a dispositivi differenti e presenti le caratteristiche di responsiveness che le sono richieste.

Conclusioni e sviluppi futuri

Questa tesi si prefiggeva l'obiettivo di progettare e sviluppare un sistema mobile per il tracciamento nell'ambiente preospedaliero, che consentisse di sfruttare le tecnologie informatiche attualmente disponibili per fornire un valido aiuto ai medici impegnati nella fase di soccorso preospedaliera.

Il lavoro progettuale è stato portato avanti individualmente, sotto l'attenta supervisione del professor Ricci e del dottor Croatti, e dal mio punto di vista si è rivelato estremamente interessante sia per il contesto applicativo sia per quello tecnologico in cui si colloca.

Per quanto riguarda il dominio applicativo, sono state evidenziate tutte le difficoltà che esso presenta ed è stata necessaria una fase di analisi molto dettagliata (vedi capitolo 2). Questa ha permesso, grazie al confronto diretto con i medici (utenti del sistema), di chiarire quali fossero le reali necessità dell'applicazione da realizzare. Inoltre le conoscenze acquisite nel corso di Ingegneria del Software mi hanno consentito di sviluppare un sistema che rispettasse, per quanto possibile, i requisiti definiti in fase di analisi.

Dal punto di vista tecnologico, la necessità di creare un sistema portabile ha fatto sì che venisse scelto un approccio cross-platform. Come è stato visto, si tratta di un tipo di sviluppo per cui non esiste uno "standard" (linguaggio di programmazione e librerie proprietarie, che caratterizzano lo sviluppo nativo). Perciò è stata necessaria un'esplorazione dei framework cross-platform, e questa ha sicuramente arricchito il mio bagaglio di conoscenze.

In questo senso la scelta di utilizzare NativeScript `&` Angular è stata molto stimolante, perché mi ha permesso di riutilizzare le nozioni conseguite nel corso di Tecnologie Web e di approfondirle studiando, anche se in un ambito non propriamente web, Angular (un framework molto utilizzato nello sviluppo web, ma che non viene trattato nel corso di questa laurea triennale).

NativeScript si è rivelato un ottimo framework, poiché mette a disposizione molti strumenti utili nello sviluppo, e mi ha permesso infatti di realizzare un software con buone caratteristiche di qualità. I feedback ricevuti dai medici in merito all'ultima versione del sistema (quella presentata) sono stati molto positivi ed essa rispetta nel modo dovuto i requisiti definiti nell'analisi, perciò mi ritengo soddisfatto del lavoro svolto.

A causa della grande mole di funzionalità che potrebbero essere implementate in un sistema come quello realizzato (completamente nuovo e affrontato per la prima volta in questa tesi di laurea) e dei possibili utilizzi dei dati raccolti dall'applicazione, sono diversi gli sviluppi futuri a cui essa potrebbe essere sottoposta. Inoltre la scelta tecnologica fatta, e in particolare l'uso di Angular (che costruisce applicazioni modulari), consente di estendere facilmente l'applicazione. Di seguito, saranno riportati gli sviluppi futuri più significativi.

- **Implementazione della comunicazione con il servizio web**

  Non appena sarà realizzato un servizio web in grado di comunicare con l'applicazione, sarà sufficiente completare l'implementazione dell'apposita parte del sistema realizzato (attualmente è già presente un servizio Angular, non implementato, che ha il compito di comunicare con il servizio web) per potergli inviare i dati raccolti.

- **Integrazione con TraumaTracker**

  Il progetto PreH Soccorso è nato con l'obiettivo di realizzare un sistema autonomo rispetto a TraumaTracker, ma l'integrazione con quest'ultimo può portare grandi vantaggi. Attualmente, in TraumaTracker i dati relativi alla fase preospedaliera sono più generici rispetto a quelli raccolti dall'applicazione realizzata e vengono inseriti manualmente.

  L'integrazione dei due sistemi permetterebbe a TraumaTracker di ricevere in maniera automatica le informazioni tracciate nella fase preospedaliera, semplificando l'attività di documentazione del trauma.

- **Interazione con altri sistemi usati nell'ambiente ospedaliero**

  Trattandosi di un sistema che raccoglie informazioni relative ad avvenimenti generici, queste potrebbero essere comunicate a sistemi specifici (di cui TraumaTracker costituisce un esempio) usati nell'ambiente ospedaliero, per attivare più velocemente, e quindi con maggiore efficacia, determinati protocolli di emergenza.

- **Collegamento del sistema a una wearable camera**

  Per produrre una documentazione multimediale della missione di soccorso (aggiungendo dei video alle foto che il sistema è in grado di scattare), potrebbe essere implementata la comunicazione tra il dispositivo e una wearable camera. NativeScript non mette a disposizione nessun plugin che permetta di comunicare, ad esempio, con dispositivi USB (la camera potrebbe essere collegata via cavo al device). Perciò, se si volesse sviluppare questa parte, con ogni probabilità sarebbe necessario ricorrere alle API native delle piattaforme che si vogliono supportare.

- **Esportazione, in formato digitale, dei dati raccolti**

  Un'altra funzionalità che potrebbe essere implementata per arricchire il sistema realizzato è costituita dall'esportazione in formato digitale di tutti i dati raccolti relativamente ad un paziente nell'ambito di una missione di soccorso. Ad esempio l'applicazione potrebbe generare automaticamente un file PDF caratterizzato da una struttura predefinita, che sostituisca la scheda cartacea attualmente utilizzata e che sia facile da stampare o condividere.

- **Porting dell'applicazione su dispositivi non mobile**

  Se fosse necessario riutilizzare il sistema su dispositivi non mobile, sfruttando il fatto che si tratta di un'applicazione Angular, sarebbe possibile effettuare il suo porting in ambiente web. Questo comporterebbe però alcune limitazioni, ossia l'obbligo di replicare il codice dell'interfaccia utente e l'impossibilità di riutilizzare i moduli destinati in modo specifico alle piattaforme mobile [27].

Ringraziamenti

Tengo a ringraziare innanzitutto la mia famiglia, che nel corso di questo percorso (e di tutta la mia vita) è sempre stata al mio fianco e che mi ha permesso, supportandomi in ogni modo possibile, di arrivare fin qui.

Grazie a mia mamma, Paola, che ha saputo trasmettermi l'importanza dello studio e la passione per le materie scientifiche (raramente il figlio di una prof di Matematica studia Filosofia... e io non faccio eccezione). Grazie a mio babbo, Alberto, che mi ha insegnato l'importanza del lavoro quotidiano, elemento indispensabile nel raggiungimento di qualsiasi traguardo. Insieme, costituiscono per me un esempio di vita che cercherò sempre di seguire.

Grazie ai miei nonni, che mi hanno accompagnato durante la mia infanzia e la mia adolescenza e che, nonostante moltissime difficoltà, sono stati con me anche durante questo percorso. Grazie a Giancarlo, sempre disponibile e affettuoso nei miei confronti e che mi ha spesso ospitato in giornate completamente dedicate allo studio, e a Sante, a cui va un grande in bocca al lupo.

Dedico un ringraziamento speciale a mia nonna, Pina, mancata durante questo percorso ma che è stata per me un importante punto di riferimento.

Per me è importante ringraziare anche Maria che, da semplice conoscente, in questi tre anni è diventata prima compagna di studi e poi compagna di vita. Con lei ho condiviso gli impegni, le preoccupazioni e le soddisfazioni di questo percorso. Siamo stati in grado di aiutarci l'un l'altro e la nostra alchimia ci ha permesso di acquisire una maggiore consapevolezza delle nostre possibilità. Sono estremamente contento di condividere con lei questo traguardo e, insieme, siamo pronti ad intraprendere un nuovo percorso.

Grazie anche ai miei amici e alle mie amiche, spesso messi da parte con un "mi dispiace ma devo studiare", con cui festeggerò questo traguardo.

Infine ringrazio il professor Alessandro Ricci e il dottor Angelo Croatti, per avermi dato la possibilità di contribuire a questo progetto e per la disponibilità con cui hanno seguito il mio lavoro nel corso di questi mesi, e il dottor Vittorio Albarello, per la disponibilità mostrata nei vari incontri.

Bibliografia

[1] Alessandro Ricci. *Corso di sistemi embedded e IoT, Modulo 1.1: Introduzione ai sistemi embedded*. Università di Bologna, a.a. 2017/2018.

[2] Paolo Colli Franzone. *The Healthcare Digital Revolution*. PKE, 2018.

[3] Bertalan Meskó. *The Guide to the Future of Medicine: Technology and the Human Touch*. Webicina Kft, 2014.

[4] Angelo Croatti, Sara Montagna, Alessandro Ricci. *A Personal Medical Digital Assistant Agent for Supporting Human Operators in Emergency Scenarios*. In *Agents and Multi-Agent Systems for Health Care*. Springer International Publishing, 2017.

[5] Angelo Croatti, Sara Montagna, Alessandro Ricci, Emiliano Gamberini, Vittorio Albarello, Vanni Agnoletti. *Personal Medical Digital Assistant Agents for Trauma Tracking and Assistance*. Computer Science and Engineering Department - University of Bologna, 2017.

[6] Stefano Rizzi. *Corso di ingegneria del software, Modulo 4: Ingegneria del Software*. Università di Bologna, a.a. 2017/2018.

[7] Stefano Rizzi. *Corso di ingegneria del software, Modulo 1: Il ciclo di vita dei sistemi informatici*. Università di Bologna, a.a. 2017/2018.

[8] *ISO/IEC 9126*. `https://it.wikipedia.org/wiki/ISO/IEC_9126`.

[9] *Dispositivo mobile*. `http://www.treccani.it/enciclopedia/dispositivo-mobile_(Enciclopedia-Italiana)/`.

[10] *Mobile computing*. `https://it.wikipedia.org/wiki/Mobile_computing`.

[11] *Le 10 invenzioni del terzo Millennio che hanno cambiato il mondo*. `http://www.askanews.it/cronaca/2016/01/19/le-10-invenzioni-del-terzo-millennio-che-hanno-cambiato-il-mondo-pn_20160119_00185/`.

[12] *Digital in 2018 Report*. https://wearesocial.com/it/blog/2018/01/global-digital-report-2018.

[13] *Smartphone OS Market*. https://www.idc.com/promo/smartphone-market-share/os.

[14] *Mobile Development: Choosing Between Native, Web, and Cross-Platform Applications*. https://steelkiwi.com/blog/how-choose-correct-platform-mobile-app-development/.

[15] *Come scegliere app Native, Ibride o Web App? Ecco le differenze*. https://it.yeeply.com/blog/scegliere-app-native-ibride-o-web-app/.

[16] *Android Developers*. https://developer.android.com/.

[17] *Apple Developer*. https://developer.apple.com/.

[18] Sarah Allen, Vidal Grauper, Lee Lundrigan. *Pro Smartphone Cross-Platform Development*. Apress, 2010.

[19] *Top 10 Cross Platform Mobile App Development Tools*. https://dzone.com/articles/top-10-cross-platform-mobile-app-development-tools.

[20] *Apache Cordova, Overview*. https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

[21] *Apache Cordova, Platform Support*. https://cordova.apache.org/docs/en/8.x/guide/support/index.html#core-plugin-apis.

[22] *Xamarin, Building Cross Platform Applications Overview*. https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/overview.

[23] *Xamarin, Understanding the Xamarin Mobile Platform*. https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/understanding-the-xamarin-mobile-platform.

[24] *Xamarin, Setting Up A Xamarin Cross Platform Solution*. https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/setting-up-a-xamarin-cross-platform-solution.

[25] *Create native iOS and Android apps with JavaScript.* https://www.nativescript.org/.

[26] *How NativeScript Works.* https://docs.nativescript.org/core-concepts/technical-overview#how-nativescript-works.

[27] *Code Sharing Introduction.* https://docs.nativescript.org/angular/code-sharing/intro#introduction.

[28] *Getting Started With NativeScript and Angular.* https://docs.nativescript.org/angular/start/introduction.

[29] *Application Architecture.* https://docs.nativescript.org/angular/core-concepts/angular-application-architecture.

[30] *NativeScript application architecture and lifecycle.* https://docs.nativescript.org/angular/core-concepts/application-lifecycle#nativescript-application-architecture-and-lifecycle.

[31] *Architecture overview.* https://angular.io/guide/architecture.

[32] *Introduction to components.* https://angular.io/guide/architecture-components.

[33] *Introduction to services and dependency injection.* https://angular.io/guide/architecture-services.

[34] *Navigation.* https://docs.nativescript.org/angular/core-concepts/angular-navigation.

[35] *La differenza tra wireframe e mockup.* https://gsite.ch/la-differenza-tra-wireframe-e-mockup/.

[36] *Balsamiq.* https://balsamiq.com/.

[37] *Façade Pattern in TypeScript.* https://fullstack-developer.academy/facade-pattern-in-typescript/.

[38] *Style Guide.* https://angular.io/guide/styleguide.

[39] *User Interface Widgets.* https://docs.nativescript.org/ui/components.

[40] *Styling.* https://docs.nativescript.org/ui/styling.

[41] *nativescript-localstorage*. https://www.npmjs.com/package/nativescript-localstorage.

[42] *Camera*. https://docs.nativescript.org/hardware/camera#installation.

[43] *Location*. https://docs.nativescript.org/hardware/location#location.

[44] *Deep Dive into NativeScript 3.1 Performance Improvements*. https://www.nativescript.org/blog/deep-dive-into-nativescript-3.1-performance-improvements.

[45] *Using Webpack to Bundle Your Code*. https://docs.nativescript.org/performance-optimizations/bundling-with-webpack#nativescript-cli-commands.