ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Dipartimento di Informatica – Scienza e Ingegneria
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

STUDIO E SVILUPPO DI UN SISTEMA DI TELEREFERTAZIONE

Elaborato in
PERVASIVE COMPUTING

Relatore
Prof. ALESSANDRO RICCI

Presentata da
DAVIDE GIACOMINI

Co-relatori
Ing. ANGELO CROATTI
Dott. MARCO LONGONI

Anno Accademico 2019 – 2020

PAROLE CHIAVE

Wearable computing

Microservices

Telemedicina

Healthcare

WebRTC

Alla mia famiglia

Abstract

La presente tesi si è incentrata sull'analisi e sullo sviluppo di un sistema di telemedicina a supporto dell'intero personale medico coinvolto nella gestione di un caso di ictus. Durante questo processo i medici si avvalgono di strumenti particolari, come l'NIHSS, per valutare la gravità del paziente e determinare quale iter seguire per la cura di quest'ultimo. Essendo un'attività tempo-dipendente, dove quindi è vitale minimizzare il tempo di trattamento della malattia, la telemedicina, ovvero abilitare una comunicazione remota tramite l'utilizzo di sistemi ICT, può migliorarne sensibilmente sia l'efficacia sia l'efficienza. Inoltre, tramite l'utilizzo di dispositivi wearable, come smartglasses, è possibile abilitare questa comunicazione minimizzando l'impedimento agli operatori sanitari. Ulteriore miglioramento si ha presentando al personale un'interfaccia che struttura il percorso da seguire per la valutazione delle condizioni del paziente, riducendo la possibilità di errori e abilitandone il tracciamento. Il sistema sviluppato risulta quindi in linea con i principi della telemedicina e, sfruttando le ultime tecnologie, sia a livello architetturale sia a livello di strumentazione wearable, agevola gli operatori, permettendo di comunicare, in modo hands-free, con specialisti ai quali viene invece offerto uno strumento per facilitare la conduzione del trattamento. Il primo prototipo del sistema è stato validato assieme ad esperti del dominio, che hanno confermato l'effettivo miglioramento del processo.

Indice

Introduzione

Lo sviluppo tecnologico degli ultimi anni ha portato all'introduzione nel mercato di dispositivi portatili sempre più compatti, pur mantenendo performance elevate ed un nutrito insieme di funzionalità. Oltre ai più comuni *smartphone* e *tablet*, si sono affermati anche altri dispositivi quali *smartwatch*, *smartband* e, sicuramente i più interessanti per l'ambito di questo progetto, gli *smartglasses*. Per questi ultimi si parla quindi di **Wearable computing**, ovvero dispositivi in grado di essere indossati, che, abilitando nuovi metodi di interazione, si rendono particolarmente utili nei casi in cui l'utente non ha la possibilità di interagire fisicamente con essi. Viene perciò introdotto il termine **Hands-free computing**, proprio per specificare i dispositivi con cui è possibile interagire in modo indiretto.

Quest'ultima categoria di sistemi risulta particolarmente utile in ambito **Healthcare**, dove è vitale non causare impedimento all'utilizzatore. L'introduzione di questi dispositivi all'interno dell'ambito ospedaliero ed extraospedaliero e la loro interazione con altri tipi di sistemi *embedded* o servizi, è la base per quello che viene chiamato **Ospedale 4.0**, ovvero la digitalizzazione delle aree e dei processi ospedalieri al fine di aumentare l'efficacia ed efficienza.

Questo avviene principalmente tramite il dislocamento nell'ambiente di sistemi in grado di percepire, condividere ed elaborare informazioni, che è la concretizzazione del modello di **Ubiquitous computing**, legato anche al neologismo **Internet of Things**.

Le potenzialità che offrono questi dispositivi risultano infatti estremamente utili quando applicate ai processi in ambito medico, dove in molti casi è necessaria reattività, tempestività di intervento ed accesso immediato ad informazioni specifiche.

Ed è proprio questo l'ambito in cui si colloca questo progetto, dove, con la collaborazione del reparto di neurologia dell'ospedale M. Bufalini di Cesena, ci si pone l'obiettivo di studiare e produrre un sistema a supporto degli attori coinvolti nel processo di analisi dei casi di ictus cerebrale.

Nello specifico si vuole fare in modo che l'artefatto prodotto possa guidare e supportare il processo di valutazione dello stato del paziente, che coinvolge sia l'operatore che il neurologo. In particolare, l'obiettivo è la creazione di

un sistema di *teleconsulto* per la valutazione della gravità dell'ictus da parte del medico, senza che sia fisicamente presente, e renderlo perciò in grado di fornire supporto diretto all'operatore, minimizzare i tempi di scelta dell'iter da intraprendere e del trattamento del paziente.

Per abilitare il *teleconsulto* sarà quindi necessario fornire agli utenti dispositivi mobili in grado di registrare flussi video e audio, senza causare impedimento. La scelta è quindi ricaduta sugli *smartglasses*, che presentano tutte le caratteristiche richieste.

Il sistema, denominato "TeleStroke", verrà progettato tenendo in considerazione le problematiche, necessità e richieste specifiche degli esperti del dominio coinvolti. Si cercherà tuttavia di renderlo il più generico possibile in modo tale che possa essere applicato ad altri scenari con un flusso di lavoro simile.

Nel capitolo seguente si andrà ad introdurre il concetto generale di *telemedicina*, con particolare enfasi sul *teleconsulto*, corredato da un quadro generale sui sistemi esistenti. Successivamente si andrà a dettagliare l'analisi, progettazione e sviluppo del sistema. Infine si esporranno i risultati ottenuti dalla validazione e le conclusioni.

Capitolo 1

Telemedicina

In questo capitolo si andrà ad introdurre il concetto di *telemedicina*, con particolare interesse per la parte inerente al *teleconsulto*. Si esporrà inoltre un quadro generale sui progetti esistenti.

1.1 Definizione

Superando il tradizionale rapporto fisico tra paziente-dottore, o dottore-dottore, la telemedicina permette l'utilizzo di servizi erogati tramite *ICT* per abilitare l'interazione remota a lunga distanza. Esistono molteplici definizioni di telemedicina, una tra le più valide risulta essere quella adottata dal **WHO**, *World Health Organization*[1] e dall'**Unione Europea**[2].
Essa definisce la telemedicina come:

> "[...] l'erogazione di servizi di cura ed assistenza, in situazioni in cui la distanza è un fattore critico, da parte di qualsiasi operatore sanitario attraverso l'impiego delle tecnologie informatiche e della comunicazione per lo scambio di informazioni utili alla diagnosi, al trattamento e alla prevenzione di malattie e traumi, alla ricerca e alla valutazione e per la formazione continua del personale sanitario, nell'interesse della salute dell'individuo e della comunità."

Anche il *Ministero della Salute* italiano condivide la medesima definizione[3], specificando tuttavia che:

[1] A health telematics policy in support of WHO's Health-for-all strategy for global health development

[2] Communication from the commission: on telemedicine for the benefit of patients, healthcare systems and society

[3] Telemedicina: linee di indirizzo nazionali

> "La Telemedicina comporta la trasmissione sicura di informazioni e dati di carattere medico nella forma di testi, suoni, immagini o altre forme necessarie per la prevenzione, la diagnosi, il trattamento e il successivo controllo dei pazienti. [...] Tuttavia la prestazione in Telemedicina non sostituisce la prestazione sanitaria tradizionale nel rapporto personale medico-paziente, ma la integra per potenzialmente migliorare efficacia, efficienza e appropriatezza"

In generale, il punto chiave della definizione risulta quindi essere la possibilità di assistenza in situazioni in cui il professionista e il paziente (o altri professionisti) non si trovano nella stessa località, tramite l'utilizzo di tecnologie informatiche e di telecomunicazione.

1.2 Vantaggi

La comunicazione remota consente di eliminare quasi completamente il problema della collocazione spaziale tra specialista e paziente. Questa particolare caratteristica permette alla telemedicina di fornire i seguenti vantaggi[4]:

- Assicura l'erogazione dei servizi sanitari a chi si trova distante da un centro sanitario
- Migliora in modo completo il sistema sanitario, soprattutto per quanto riguarda i servizi di emergenza, organizzazione medica ed educazione/addestramento/aggiornamento del personale medico
- Migliora l'efficacia clinica nella cura dei malati e l'efficienza gestionale
- Concentra le risorse per la cura di malati in fase acuta o con affezioni molto gravi che richiedono terapie particolari o interventi chirurgici
- Contenimento della spesa sanitaria
- Riduzione mortalità dei pazienti affetti da malattie croniche grazie a monitoraggio costante

[4]Dizionario di Medicina, definizione telemedicina

1.3 Criticità

Oltre ai vantaggi appena esposti, questo particolare approccio comporta anche diverse criticità.

La telemedicina prevede infatti la trasmissione di informazioni e dati di carattere medico. Ciò può comportare dei rischi. Teoricamente potrebbe infatti essere violata la riservatezza dei dati, intenzionalmente o accidentalmente, sia durante la sessione di telemedicina sia durante la trasmissione delle informazioni a distanza, oppure in sede di archiviazione.

Per mitigare questi rischi, la Commissione Europea[5] e il Ministero della Salute[6] hanno emanato raccomandazioni volte a rafforzare i sistemi a protezione dei dati e per garantire la massima sicurezza per i pazienti, sia a livello di raccolta che di archiviazione ed utilizzo delle suddette informazioni.

Altri aspetti da considerare riguardano la possibilità di malfunzionamenti dell'apparecchiatura, con possibile alterazione di dati, interpretazione errata o mancata rilevazione di anomalie. Inoltre un problema grave al sistema informatico o all'infrastruttura di telecomunicazione, potrebbe impedire, parzialmente o totalmente, le operazioni di soccorso da parte gli operatori sanitari.

Infine viene anche introdotto il discorso sulla tutela del paziente. La telemedicina deve infatti ottemperare a tutti i diritti e obblighi vigenti per un qualsiasi atto sanitario, è quindi necessario informare il paziente sul tipo di servizio, i rischi collegati e gli esiti attesi, probabili e possibili[7].

1.4 Applicazioni

La telemedicina può essere applicata in diversi ambiti e settori medici, con diverse finalità. Alcuni esempi sono:

- **Telemonitoraggio**, nel caso in cui un paziente affetto da una patologia viene sottoposto ad un monitoraggio costante a distanza
- **Teleradiologia**, ovvero la trasmissione di immagini radiografiche
- **Teleriabilitazione**, servizi erogati presso domicilio o in strutture vicine a pazienti a cui viene prescritto un intervento riabilitativo
- **Teledidattica**, ovvero l'addestramento o aggiornamento di specialisti da remoto

[5] Communication from the commission: on telemedicine for the benefit of patients, healthcare systems and society

[6] Telemedicina: linee di indirizzo nazionali

[7] Norme in materia di consenso informato e di disposizioni anticipate di trattamento

Oltre all'ambito applicativo, la telemedicina può essere classificata in base alle varie modalità in cui vengono forniti i servizi e agli attori coinvolti nel processo. Nella telemedicina specialistica si fa distinzione tra:

- **Televisita**, l'atto sanitario in cui il medico interagisce direttamente con il paziente per eseguire una diagnosi
- **Teleconsulto**, attività di consulenza tra medici o operatori sanitari (molto spesso collegato alla **Telerefertazione**, ovvero la stesura di un referto medico tramite supporto informatico)
- **Telecooperazione sanitaria**, assistenza fornita da un medico o altro operatore sanitario ad un altro attore impegnato in un atto sanitario

1.5 Stato dell'arte

Stimolati dalla commissione europea, diversi paesi hanno già da tempo avviato sperimentazioni e realizzazioni concrete di servizi di telemedicina.
Si riportano in seguito alcuni esempi di applicazione o servizi di telemedicina in uso in alcuni paesi europei:

- **Germania**: Tramite il progetto *bIT4Health*[8] si è implementata la cosiddetta **HealthCard**, che conteneva foto, dati personali e amministrativi dell'assistenza sociale. È stata utilizzata per la *ePrescription*, per agevolare pazienti e farmacisti nel trasmettere le prescrizioni e aggiornare l'*electronic patient record*, ottimizzando la fornitura di farmaci. Altro progetto tedesco è stato il **LifeSensor**, sistema di registrazione avanzato che permette ai medici di accedere a dati clinici del paziente, trattamenti, terapie.
- **Paesi scandinavi**: Il progetto **RTF** (Regional Telemedicine Forum Project) prevede la collaborazione di 9 regioni europee, per favorire un utilizzo più ampio e la diffusione di servizi della telemedicina, oltre che fornire linee guida. Un secondo progetto attivo è il **Dreaming Project**, orientato principalmente alla teleassistenza domiciliare, con l'obiettivo di tenere anziani nel loro ambiente familiare fino a quando le condizioni lo consentono.
- **Francia**: Uno dei paesi europei in cui la telemedicina è maggiormente diffusa e regolamentata, grazie principalmente ad un decreto specifico (n°2010–1229) pubblicato sul *Journal officiel de la République française*.

[8] E-Health and the Future of Healthcare Information Systems

Nel decreto si ufficializzano gli atti di teleconsulto, telesorveglianza e teleassistenza.

- **Spagna**: A causa del sistema sanitario molto frammentato sono nati diversi progetti. Alcuni esempi sono la **cartella clinica digitale**, un **Personal Health File**, **medical imaging** e la **prescrizione elettronica**.

- **Inghilterra**: l'**NHS**[9] ha emanato linee guida strategiche per la sanità, che prevedono l'uso della telemedicina per migliorare l'assistenza domiciliare, i servizi mobili di emergenza, videoconsulti e accesso remoto a cartelle cliniche. In particolare, **Telehealth** e **Telecare** costituiscono i sistemi più sviluppati per la gestione di malattie croniche. Il primo prevede il monitoraggio remoto di alcuni valori e parametri vitali tramite un centro che provvede in caso di anomalie di contattare il personale medico. Il secondo invece è riferito ad un sistema di allarme che permette a persone vicine al malato di essere avvertiti in caso di emergenza.

Alcuni esempi extra-europei sono:

- **USA**: Un primo esempio è costituito dalla **ATA**[10], che fornisce servizi come teleconsulto medico-paziente, monitoraggio a distanza, educazione medica per operatori sanitari e informazioni sanitarie accessibili on-line. Inoltre, grazie all'emanazione dell'**HITECH Act**, si è vista l'adozione del **EHR** (Electronic health records), ovvero cartella clinica elettronica abilitante per la ricerca, miglioramento salute pubblica e compentenza operatori sanitari.

- **Canada**: Ha incentivato sistemi di telemedicina grazie alla corporazione indipendente no-profit *Canada Health Infoway*. Promuove sviluppo e adozione di tecnologie per trasformare l'assistenza sanitaria canadese. Prevede collaborazione di pazienti, governo, docenti, ricercatori e industria tecnologica.

[9] National Health Service
[10] American Telemedicine Association

Capitolo 2

Il progetto TeleStroke

Si andrà ora ad introdurre il progetto specificando le motivazioni che hanno portato alla sua realizzazione, gli attori disponibili per la fase di sperimentazione e un esempio di scenario tipico per quanto riguarda il trattamento attuale, per comprenderne meglio le criticità ed avere un quadro generale sulla sequenza di attività eseguite.

2.1 Motivazione

Nel trattamento di casi di **ictus cerebrale** è fondamentale la velocità di diagnosi e di trattamento del paziente. Proprio per questo motivo, risulta estremamente efficace applicare la telemedicina a questo contesto.

In particolare, il progetto si concentrerà sulla gestione del *teleconsulto* e *telecooperazione*, ovvero sull'offrire un sistema in grado di abilitare uno specialista a fornire supporto *real-time* ad un operatore sul campo.

L'obiettivo preposto risulta essere il supporto sia del medico che dell'operatore durante la fase di diagnosi dell'ictus, in modo da accelerare il processo decisionale per quanto riguarda la selezione dell'*iter* da intraprendere per la cura del paziente, così da massimizzare le possibilità di recupero di quest'ultimo.

2.1.1 Vantaggi

I vantaggi principali che porterebbe l'applicazione della telemedicina in questo scenario sono:

- Velocità di esecuzione
- Qualità della prestazione, dovuto principalmente alla presenza di uno specialista

- Miglioramento per quanto riguarda la logistica, in quanto gli specialisti non hanno necessità di spostarsi e possono fornire supporto in tempi brevi
- Limitazione errori, sia da parte degli operatori che degli specialisti
- Ottimizzazione tempo specialisti, annullamento tempi morti causati dallo spostamento

2.2 Attori disponibili

Nello sviluppo del sistema diversi attori, esperti del dominio, si sono resi disponibili per la sperimentazione. Il loro contributo è risultato vitale sia per quanto riguarda la fase di progettazione che durante la fase di validazione.

In particolare, il progetto è stato supportato da:

- Neurologia e pronto soccorso dell'ospedale M. Bufalini di Cesena
- Neurologia e pronto soccorso dell'ospedale Morgagni-Pierantoni di Forlì

Il referente principale del progetto è stato il Dott. Marco Longoni, Direttore SC Neurologia e Stroke Unit di Cesena e Forlì.

2.3 Scenario di utilizzo

Grazie al contributo degli attori citati in precedenza si è riuscito a definire uno scenario tipico in cui il sistema verrà utilizzato.

Questo risulterà particolarmente utile nella fase di analisi per definire i componenti del sistema, le entità coinvolte, la loro collocazione spaziale e la loro interazione.

Di seguito sono riportate le fasi nel processo di gestione di un paziente affetto da ictus:

- Un'ambulanza preleva il paziente e lo porta in pronto soccorso
- Il paziente arriva in pronto soccorso
- Lo specialista è informato dell'arrivo del paziente
- L'operatore in pronto soccorso sposta il paziente in una stanza predisposta al teleconsulto
- Lo specialista inizia una sessione di teleconsulto dalla sua postazione verso il pronto soccorso

- Lo specialista utilizzando la **NIHSS**, National Institutes of Health Stroke Scale, valuta il paziente, supportato dall'operatore
- Lo specialista termina la valutazione e riferisce all'operatore come procedere

2.3.1 National Institutes of Health Stroke Scale

In questa sezione si andrà a descrivere la **National Institutes of Health Stroke Scale**, scala che è stato richiesto di supportare all'interno del sistema da sviluppare, già utilizzata dagli specialisti per valutare un caso di ictus.

L'**NIH Stroke Scale** (**NIHSS**) è uno strumento utilizzato in ambito healthcare per quantificare oggettivamente l'impedimento causato da un ictus.

La scala è composta da 11 elementi, ciascuno dei quali assegna un punteggio a una specifica abilità, da 0 a 4. Per ogni elemento un punteggio di 0 indica il funzionamento normale della specifica abilità, mentre un punteggio alto è indicativo di una qualche forma di impedimento. I valori degli elementi sono sommati tra di loro per calcolare il punteggio totale. Esso varia tra 0, nessun sintomo di ictus, e 42, forma molto grave di ictus.

Di seguito vengono elencati gli 11 punti dai quali è composta la scala:

- **1A: Vigilanza** - si valuta se il paziente è sveglio
- **1B: Orientamento** - si chiede al paziente mese ed età
- **1C: Esecuzione ordini** - si chiede al paziente di fare il pugno e di chiudere gli occhi
- **2: Deviazione dello sguardo** - si chiede al paziente di muovere gli occhi sul piano orizzontale
- **3: Campo visivo** - si valuta assenza di risposta allo stimolo visivo mostrato consecutivamente in uno dei quattro quadranti del campo visivo
- **4: Deviazione rima faciale** - si chiede al paziente di chiudere/aprire gli occhi e mostrare i denti/sorridere
- **5A: Forza arto superiore sinistro** - il paziente deve sollevare dal piano del letto singolarmente l'arto superiore sinistro mantenendolo in posizione per 10 secondi
- **5B: Forza arto superiore destro** - stessa procedura, ma per arto superiore destro

- **6A: Forza arto inferiore sinistro** - il paziente deve sollevare dal piano del letto singolarmente l'arto inferiore sinistro mantenendolo in posizione per 5 secondi
- **6B: Forza arto inferiore destro** - stessa procedura, ma per arto inferiore destro
- **7: Atassia** - si chiede al paziente di eseguire la prova indice-naso e tallone-ginocchio singolarmente prima con un un lato del corpo e poi con l'altro, con gli occhi aperti
- **8: Sensibilità** - si somministra al paziente uno stimolo tattile ed uno doloroso ad entrambi gli emisomi
- **9: Afasia** - si fa ripetere al paziente una frase si chiede di leggere una serie di parole e di descrivere un oggetto
- **10: Disartria** - si valuta insieme all'afasia
- **11: Estinzione/trascurare** - si chiede al paziente di localizzare lo stimolo visivo o tattile che viene somministrato prima in un singolo lato e poi contemporaneamente in entrambi, si chiede inoltre al paziente di riconoscere il proprio lato corporeo

2.4 Progetti esistenti

Prima di procedere con la descrizione del progetto si andranno ad elencare alcuni riferimenti di progetti esistenti, inerenti allo stesso ambito:

- **Telbios**, un'azienda nel milanese, ha collaborato con alcuni ospedali lombardi e ha avviato sperimentazioni per l'utilizzo di robot, collegato direttamente con il reparto di Neurologia, per fornire consulto in presa diretta[1][2]
- **Dipartimento di Neurologia Mayo Clinic** negli Stati uniti ha eseguito uno studio di fattibilità per l'utilizzo di **NIHSS** in ambulanza[3]

[1]Telbios, Telemedicina per gli ospedali di Magenta, Legnano, Abbiategrasso e Cuggiono
[2]Salute: Legnano primo ospedale pubblico con telemedicina
[3]Ambulance-based assessment of NIH Stroke Scale with telemedicine: A feasibility pilot study

Capitolo 3

Analisi

L'obiettivo del progetto è quello di creare un sistema di *teleconsulto* e *telecooperazione* per assistere specialista ed operatore durante la fase di analisi della gravità di un paziente affetto da ictus. Dovrà inoltre registrare e salvare le operazioni eseguite dallo specialista, fungendo quindi anche come piattaforma di *telerefertazione*.

3.1 Requisiti

In questa sezione verranno elencati i requisiti del sistema, ottenuti durante il colloquio con gli esperti del dominio che si sono offerti per la sperimentazione del sistema.

I requisiti principali sono:

- Specialista e operatore devono poter comunicare tramite il sistema in real-time
- Lo specialista deve poter visualizzare il paziente gestito dall'operatore in real-time. La qualità della visualizzazione deve essere tale da poter permettere allo specialista di vedere chiaramente il paziente
- Il sistema deve essere veloce e reattivo, evitando di rallentare il processo di diagnosi
- Allo specialista deve essere presentata un'interfaccia che mostri una versione virtuale della **NIHSS** con la quale possa interagire
- Il sistema deve guidare il processo di diagnosi e minimizzare il rischio di errori
- Il sistema deve minimizzare l'impedimento causato all'operatore

- L'accesso al sistema da parte dello specialista potrebbe avvenire tramite dispositivo fisso o mobile

- Il sistema deve fornire sicurezza adeguata poiché vengono trattati dati sensibili di pazienti

- Il sistema deve tracciare le operazioni eseguite per finalità di addestramento, tracciabilità e valutazione

Oltre ai requisiti ottenuti tramite la discussione con gli esperti, è stato fornito anche un documento che elencava più nel dettaglio tutte le fasi di una valutazione tramite **NIHSS**.

Di seguito sono riportate le varie fasi, ciascuna corredata da un elenco di operazioni:

- **0. Raccolta ABCD e parametri vitali** – Fase eseguita dal personale di soccorso:
  - Parametri vitali (Pressione arteriosa, saturazione, frequenza cardiaca, stick glicemico)
  - Rilevazione eventuali alterazioni neurologiche esordite improvvisamente da meno di 24h (deficit linguaggio, motorio, di sensibilità, della coscienza, visivo, mancanza di equilibrio)

- **1. Raccolta informazioni su paziente** – Richieste neurologo a operatore:
  - Età
  - Genere
  - Ora esordio sintomi
  - Disabilità precedenti (calcolo m-RS[1])
  - Malattie precedenti
  - Farmaci assunti

- **2. Esecuzione NIHSS** – L'operatore si pone di fronte al paziente

- **3. Analisi risultato e suggerimento iter**

[1]Modified Rankin Scale

3.2 Analisi requisiti

Dopo aver esplicitato i vari requisiti ottenuti, si andrà ora a riportare la loro analisi.

3.2.1 Personale coinvolto

Tramite i requisiti è possibile classificare il personale coinvolto nell'utilizzo del sistema nelle seguenti categorie:

- **Specialista**: il neurologo competente e specializzato nel trattamento di ictus
- **Operatore**: l'operatore sanitario in pronto soccorso che potrà interagire direttamente con il paziente

Tendenzialmente, un operatore, in un dato momento, gestirà un singolo paziente e sarà collegato ad un singolo specialista.

Un altro assunto che è possibile fare riguarda la collocazione spaziale dell'operatore e lo specialista. Infatti si assume che non siano mai nello stesso luogo durante l'utilizzo del sistema. Questo perché, dopotutto, lo scopo di questo progetto è esattamente quello di abilitare una cooperazione remota tra queste due entità. Ciò nonostante, in caso di presenza di entrambe le entità davanti al paziente, il sistema risulterà comunque utile per compilare il referto e guidare il processo di analisi dell'ictus.

3.2.2 Comunicazione real-time

In un sistema di *teleconsulto* e *telecooperazione*, il concetto di comunicazione tra medico e personale medico è da intendersi come videoconferenza audio-video real-time. È stato quindi necessario ricercare un protocollo in grado di permettere videoconferenza real-time tra dispositivi mobili.

La ricerca si è conclusa con la selezione del protocollo **WebRTC**[2]. Si tratta di un progetto open source, standardizzato dal *W3C*[3] e supportato dalla maggior parte dei browser presenti sul mercato. Offre inoltre API native per le maggiori piattaforme mobile (Android e iOS).

Risulta quindi il candidato perfetto per l'implementazione della comunicazione real-time all'interno del sistema.

[2]WebRTC
[3]WebRTC 1.0: Real-time Communication Between Browsers

3.2.3 Interfacciamento con il sistema

In questa sezione verrà analizzato più nel dettaglio in che modo ciascun utente dovrebbe interagire con il sistema.

Operatore

L'interfacciamento dell'operatore con il sistema deve essere studiato per minimizzare l'impedimento. Come già riportato, dispositivi di **Wearable computing**, in particolare **Eyewear**, ovvero *smartglasses*, risultano molto efficaci sotto questo aspetto. In accordo con gli esperti è stato quindi intrapreso un percorso di ricerca di un dispositivo che presentasse tutte le caratteristiche richieste.

La scelta è ricaduta sul dispositivo **Vuxiz Blade**[4], in quanto possiede tutte le qualità richieste, elencate in seguito:

- impedimento minimo, in quanto hanno un ingombro molto simile a comuni occhiali da vista
- presentano una videocamera e un microfono ad alta risoluzione in grado di registrare flussi audio e video
- hanno la possibilità di riprodurre flussi audio
- presentano la possibilità di visualizzare informazioni (in questo caso proiettate direttamente nella lente destra)
- è possibile sviluppare applicazioni installabili all'interno del dispositivo (il sistema operativo *onboard* è *Android OS*)

Figura 3.1: Figura che mostra il dispositivo Vuzix Blade Smart Glasses.

[4]Vuzix blade smart glasses

Specialista

Per quanto concerne lo specialista, i dispositivi che potrà utilizzare sono molteplici. Infatti è possibile che utilizzi un dispositivo fisso (computer desktop) nel proprio studio, oppure un dispositivo portatile, per esempio un *tablet* o *smartphone*, qualora sia in movimento.

Si assume comunque che il dispositivo sia equipaggiato adeguatamente per:

- registrare la voce dello specialista
- riprodurre il flusso video e audio remoto dell'operatore collegato
- visualizzare adeguatamente sia il flusso video che l'interfaccia per la refertazione

3.2.4 Interfaccia specialista

L'interfaccia mostrata allo specialista, oltre ad essere intuitiva e facile da utilizzare, dovrà guidare il processo di valutazione mostrando una versione virtuale della **NIHSS**. Inoltre dovrà riprodurre, durante la compilazione, i flussi audio e video registrati tramite il dispositivo dell'operatore.

Struttura

Il processo potrebbe prevedere diverse fasi, questo indica che sarebbe opportuno modellare la struttura dell'interfaccia in termini più generali, senza concentrarsi solamente su una specifica vista.

I dettagli di questa struttura saranno spiegati accuratamente nel capitolo successivo, per ora comunque risulta opportuno introdurre i seguenti termini:

- **Template**: Lista ordinata di *checklist*
- **Checklist**: Lista ordinata di *step*
- **Step**: Raggruppamento di *field*
- **Field**: Rappresenta un campo compilabile

Ciascuno di questi termini avrà una propria rappresentazione visuale all'interno dell'interfaccia.

Adattamento

Oltre alla struttura gerarchica dei controlli, sarebbe opportuno che l'interfaccia adattasse il proprio contenuto e layout in base allo spazio disponibile, ovvero rispetto alla dimensione dello schermo del dispositivo. Inoltre si dovrà cercare di mantenere ben visibile sia la visualizzazione del flusso video, sia il *template*; potenzialmente facendo in modo che occupino ciascuno la metà dello spazio disponibile.

3.2.5 Casi d'uso

Si andranno ora ad illustrare i vari casi d'uso di utilizzo del sistema per entrambe le categorie di utenti.

Gestione sessione

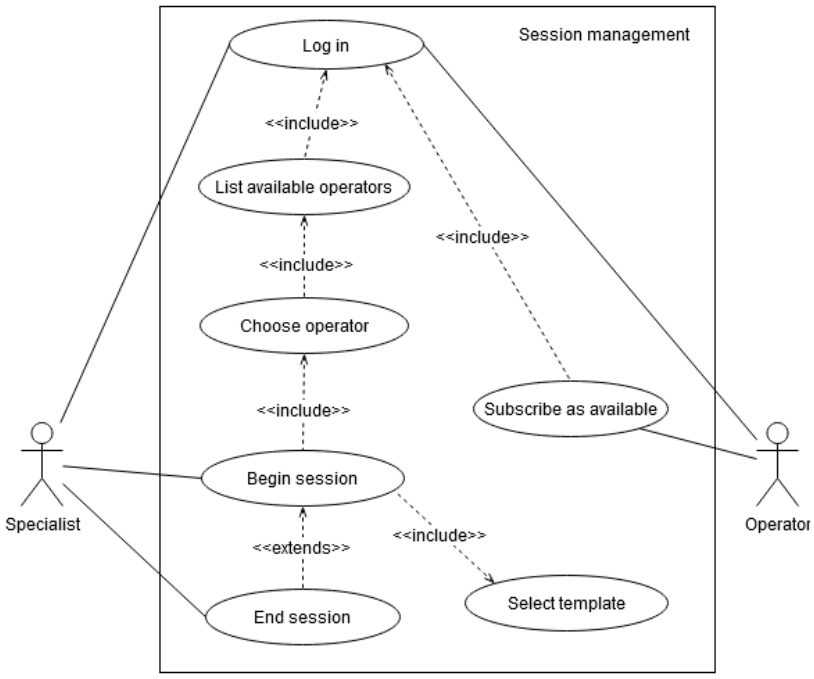


Figura 3.2: Figura che mostra il diagramma dei casi d'uso per gli utenti utilizzatori del sistema durante la gestione di una sessione.

In Figura 3.2 viene mostrato un esempio di creazione di *sessione* di *teleconsulto*. Una *sessione* è collegata al trattamento di un singolo paziente. Tiene traccia di tutte le azioni eseguite dallo specialista e viene conclusa quando

lo specialista ha terminato la compilazione del referto, analizzato il risultato e suggerito l'iter da intraprendere per la cura del paziente. La creazione di una *sessione* richiede la scelta di un operatore e di un *template*, ovvero della sequenza di fasi mostrate nell'interfaccia.

Nel prossimo capitolo sarà descritto più accuratamente cosa significa e cosa comprende.

Compilazione referto

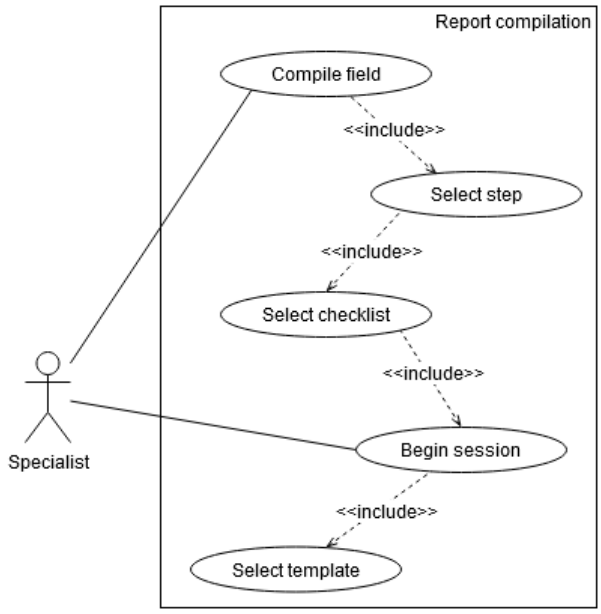


Figura 3.3: Figura che mostra il diagramma dei casi d'uso inerente all'interazione tra specialista e sistema.

In Figura 3.3 si riporta invece una tipica interazione tra specialista ed interfaccia durante la compilazione delle varie fasi. Il sistema si occuperà inoltre di tracciare tutte le azioni svolte dallo specialista, ovvero l'aggiunta di un valore in uno specifico *field*, di uno specifico *step* all'interno di una *checklist*.

Interazione utenti

Infine, nel diagramma in Figura 3.4 vengono mostrati tutte le possibili interazioni tra specialista e operatore.

All'avvio di una nuova sessione da parte dello specialista, il sistema si occuperà di stabilire immediatamente una videoconferenza con l'operatore selezionato, il quale avrà necessariamente comunicato la propria presenza e sarà

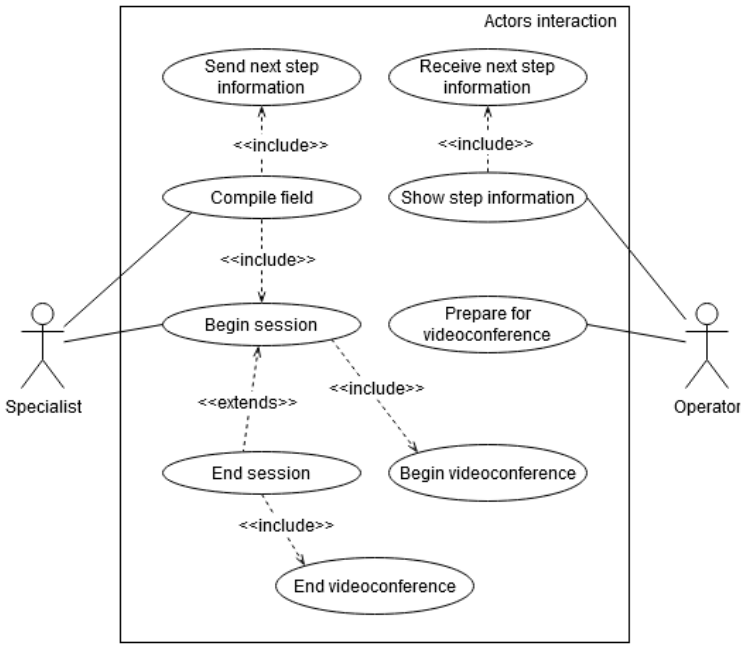


Figura 3.4: Figura che mostra il diagramma dei casi d'uso inerente all'interazione tra specialista e operatore.

già in attesa di ricevere una richiesta per la creazione di una nuova videoconferenza. Essa verrà poi terminata automaticamente una volta che la sessione sarà conclusa.

Inoltre, durante la sessione, ogniqualvòlta lo specialista compilerà un campo, il sistema dovrà inviare le informazioni sullo *step* successivo, così facendo ne potranno essere mostrati i dettagli all'operatore.

3.2.6 Scenario di validazione

Grazie alle informazioni fornite dagli specialisti è stato possibile definire uno scenario di test, che verrà utilizzato per validare il sistema.

Di seguito è riportata la sequenza stabilita di azioni da eseguire:

- L'operatore attiva il proprio dispositivo e si rende disponibile per una nuova *sessione*
- Lo specialista accede al sistema
- Lo specialista seleziona l'operatore disponibile e il *template* da utilizzare

- Il sistema deve mostrare la corretta successione di fasi e riprodurre contemporaneamente il flusso audio-video generato dal dispositivo collegato all'operatore selezionato
- Il dispositivo dell'operatore deve riprodurre il flusso audio generato dal dispositivo dello specialista a cui è collegato
- Lo specialista inizia a compilare i vari campi mostrati nell'interfaccia
- Lo specialista termina la *sessione*

3.2.7 Assunzioni fatte

Per terminare la fase di analisi dei requisiti, si elencano le assunzioni fatte per garantire il corretto funzionamento del sistema.

- Entrambi i dispositivi coinvolti nel processo devono poter comunicare tra di loro
- Non essendo contemplato dal sistema il supporto a notifiche, si presuppone che l'operatore notifichi tramite un mezzo secondario lo specialista riguardo ad un nuovo paziente da valutare

Capitolo 4

Progettazione

In questo capitolo si andrà ad introdurre la progettazione per determinati componenti del sistema. Successivamente si andrà a fornire un quadro generale sull'architettura, individuando e descrivendo i principali artefatti che lo compongono.

4.1 Modellazione template

Come già rilevato durante la fase di analisi, sarebbe opportuno rendere il sistema in grado di supportare la configurazione dinamica dell'interfaccia dello specialista. Per far ciò è stata quindi pensata una struttura gerarchica per rappresentare i vari elementi all'interno dell'interfaccia, in grado di supportare una vasta gamma di layout.

Questa struttura comprende i seguenti componenti:

- **template**: Rappresenta una particolare configurazione, selezionabile dallo specialista. Comprende una serie ordinata di *checklist*.
- **checklist**: Singola fase di un particolare processo di consulto, rappresenta una lista di *step* che sono in qualche modo collegati.
- **step**: Elemento di una *checklist*, contiene un gruppo di *field*.
- **field**: Ultimo componente nella gerarchia, rappresenta un campo compilabile dall'utente.

Ad ogni *checklist* possono essere associati uno o più *risultati*. Il **risultato** non è altro che un valore che viene calcolato in base al valore di determinati *field* all'interno della *checklist*.

In Figura 4.1 è mostrata in dettaglio la gerarchia dei componenti e le loro associazioni.

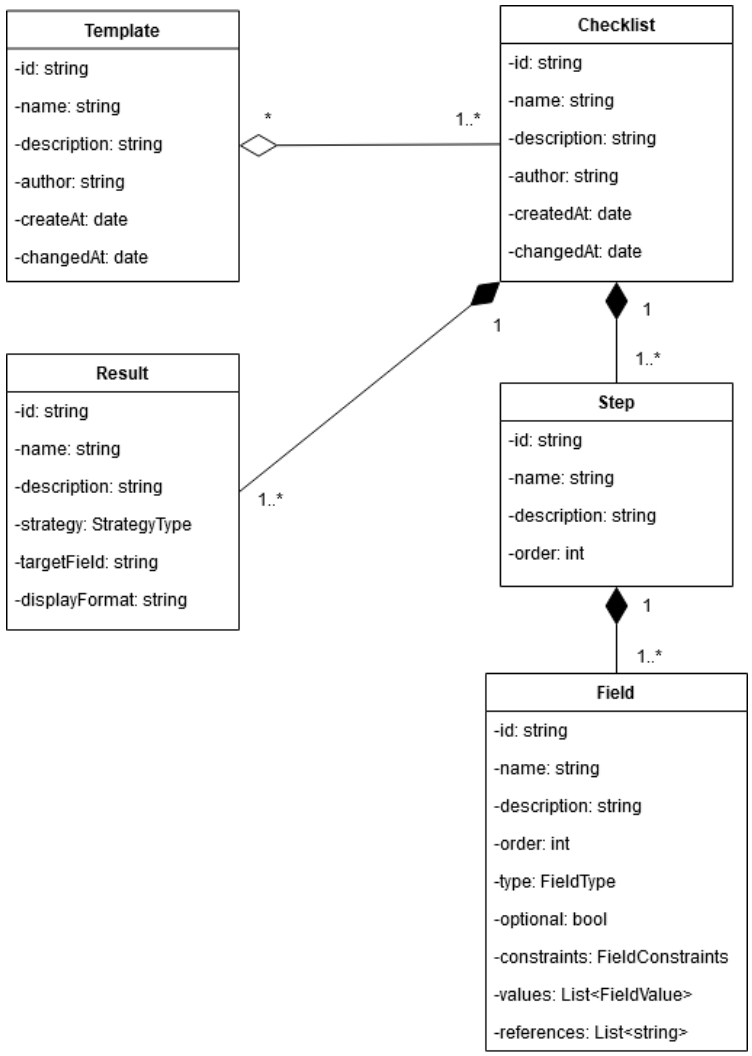


Figura 4.1: Diagramma delle classi che mostra la struttura gerarchica del modello di template e checklist

4.1.1 Configurazione field

Ad ogni *field* sono associate diverse proprietà:

- **optional**: definisce se la sua compilazione è necessaria o opzionale
- **type**: Definisce il tipo di *field*, che può essere:
  - *value*: se il componente accetta un singolo valore, che deve essere immesso manualmente dall'utente

  - *select*: il componente accetta un set ristretto di valori predefiniti, all'utente verrà presentato un controllo per la selezione di un singolo valore

- **constraints**: proprietà che permette di definire limitazioni al valore che può essere immesso dall'utente. Oltre al valore massimo e minimo, è possibile configurare il tipo di valore, tra i seguenti:

  - **text**: stringa
  - **multiline-text**: stringa multilinea
  - **number**: numero
  - **datetime**: data con orario

- **references**: lista di identificatori (*targetField*) che determina per quali *risultati* il valore del *field* deve essere utilizzato nel calcolo

A seconda del tipo di valore il componente sarà reso graficamente in modo diverso.

4.1.2 Rendering checklist

In Figura 4.3 si mostra in che modo vengono resi graficamente i modelli citati in precedenza nel *frontend* dello specialista.

Nell'immagine sono evidenziati i vari controlli tramite rettangoli colorati in modo diverso:

- **verde**: checklist, ciascuna resa come una *tab*[1]
- **rosso**: step, elemento di uno *stepper*[2]
- **magenta**: risultato, viene mostrato utilizzando il formato definito all'interno del modello
- **ciano**: field, in questo caso evidenziato un controllo di tipo *select* (alla sua destra uno di tipo *value*)

[1] Angular Material: Tabs
[2] Angular Material: Stepper

Figura 4.2: Immagine che mostra la resa grafica di un template, dove sono evidenziati con colori differenti i vari elementi che lo compongono

4.2 Modellazione sessione

Come accennato in precedenza, una sessione rappresenta l'intero processo di valutazione delle condizioni del paziente. Il suo modello presenta le seguenti proprietà:

- **id**: identificativo univoco
- **specialist**: il riferimento allo specialista
- **operator**: il riferimento all'operatore
- **startDate**: l'istante in cui è stata creata
- **endDate**: l'istante in cui si è conclusa
- **template**: il riferimento al *template* utilizzato

Inoltre, ad ogni sessioni sono collegate una serie di fasi, che dipendono dal *template* scelto, ciascuna identificata da:

- **checklist**: identificatore della *checklist*
- **actions**: lista di *azioni* eseguite dall'utente fino ad ora per questa specifica *checklist*

- **results**: lista di *risultati* con indicato l'identificativo del risultato e il valore corrente

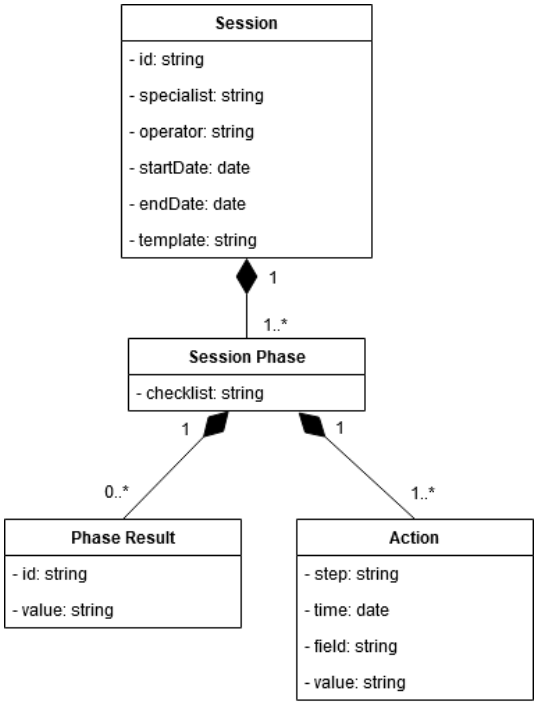


Figura 4.3: Diagramma delle classi che mostra il modello di una sessione

4.2.1 Gestione sessione

Lo specialista tramite il proprio frontend ha la possibilità di creare una nuova sessione selezionando un *template* e un operatore attivo (se presente). La creazione prevede la scrittura di un nuovo record, dove la proprietà *endDate* risulta ancora non assegnata. Dopodiché può decidere se chiuderla o abbandonarla.

La chiusura necessita che tutti i campi non opzionali siano stati compilati correttamente. In caso affermativo verrà aggiornata la data di terminazione determinando il completamento della sessione, altrimenti sarà riportato un errore. Se invece lo specialista decide di abbandonarla, la sessione rimarrà in sospeso e potrà essere ripresa in seguito.

4.2.2 Gestione operatore

Il sistema conserva la lista di tutti gli operatori attivi, ovvero quelli che hanno comunicato la loro presenza sottoscrivendosi al sistema.

Per evitare che più specialisti possano selezionare lo stesso operatore, ciascuno presenta l'identificativo della sessione alla quale sono collegati (*currentSession*), assegnato durante la creazione di una nuova sessione. Qualora questo campo non fosse valorizzato, l'operatore viene considerato libero.

Infine, il rilascio dell'operatore avviene quando uno specialista decide di chiudere o abbandonare una sessione.

4.3 Gestione azioni

Per azione si intende la compilazione di uno specifico *field*. Il frontend genera azioni ogni volta che lo specialista passa ad un altro *step* o cambia *checklist*. Il salvataggio di quest'ultima avviene tramite una chiamata al backend, il quale si occupa di aggiornare il record relativo alla specifica sessione all'interno del database. Prima di eseguire il salvataggio, il backend controlla che il valore rispetti i limiti configurati per lo specifico *field*.

Un'azione presenta le seguenti proprietà:

- **step**: identificativo dello *step*
- **field**: identificativo del *field*
- **time**: istante in cui è stata generata l'azione
- **value**: il valore impostato nel *field*

4.4 Interfaccia specialista

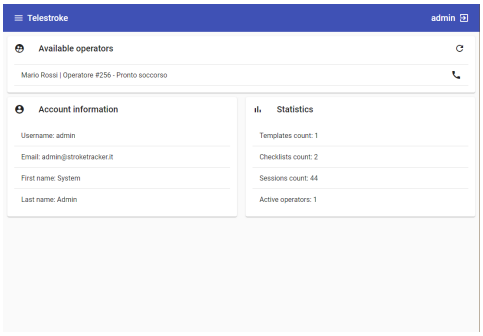
L'interfaccia dello specialista è accessibile solo dopo aver confermato l'identità dello specialista, che deve quindi immettere le credenziali di accesso ed effettuare il login.

La progettazione di quest'ultima è stata svolta seguendo le linee guida e i principi del **Material Design**[3], ovvero un particolare linguaggio visuale che sintetizza la classica concezione di buon design, che si traduce anche in un notevole miglioramento nella cosiddetta **User Experience** (*UX*). Inoltre, utilizzando un *framework* per la costruzione di *web-app*, quale **Angular**, basato proprio su questi principi, si è riuscito a concepire molto velocemente la struttura dell'interfaccia, favorito anche dalla vasta gamma di componenti nativi a disposizione.

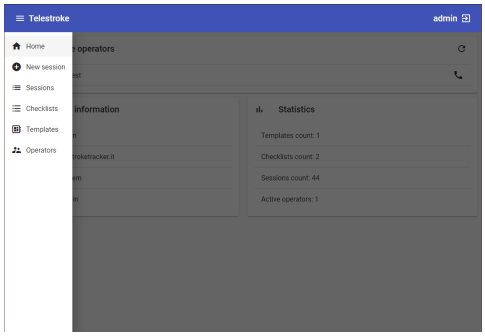
[3]Material: design

Come mostrato in Figura 4.4a e 4.4b, essa si presenta con una *toolbar*[4] superiore, all'interno della quale vi è l'indicazione dello username correntemente collegato e un pulsante di logout. È inoltre presente una *sidenav*[5] a scomparsa, la quale permette di accedere alle pagine in cui è possibile mostrare liste paginate inerenti a: sessioni, template, checklist e operatori attivi.

Importante notare come già dalla pagina iniziale sia mostrato l'elenco di operatori attivi, dove cliccando in un elemento si può immediatamente iniziare una nuova sessione, dopo aver selezionato il *template* da utilizzare. Questo permette di diminuire il numero di passaggi necessari per avviare una nuova sessione.



(a) Schermata iniziale frontend specialista.



(b) Sidenav frontend specialista.

Per quanto riguarda invece la gestione della pagina adibita alla visualizzazione del flusso video dell'operatore e delle *checklist* da compilare, si sono utilizzate *media queries*[6] per poter modificare la disposizione dei controlli al variare della dimensione orizzontale dello schermo. In questo modo rimarrà agevole la compilazione dei campi, evitando di rimpicciolire la dimensione del componente dedicato alla riproduzione del video.

Infine, in Figura 4.6 e 4.5 sono mostrate entrambe le configurazioni al variare della dimensione dello schermo. Nello specifico, il layout viene modificato quando la pagina rileva una risoluzione orizzontale maggiore di 1100 pixels.

[4]Angular Material: toolbar
[5]Angular Material: Sidenav
[6]CSS: Media queries

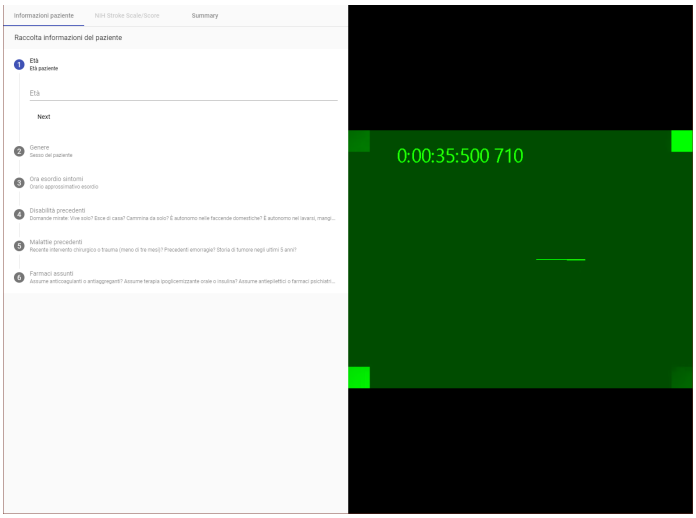


Figura 4.5: Pagina di compilazione sessione su schermi con grandezza maggiore di 1100px

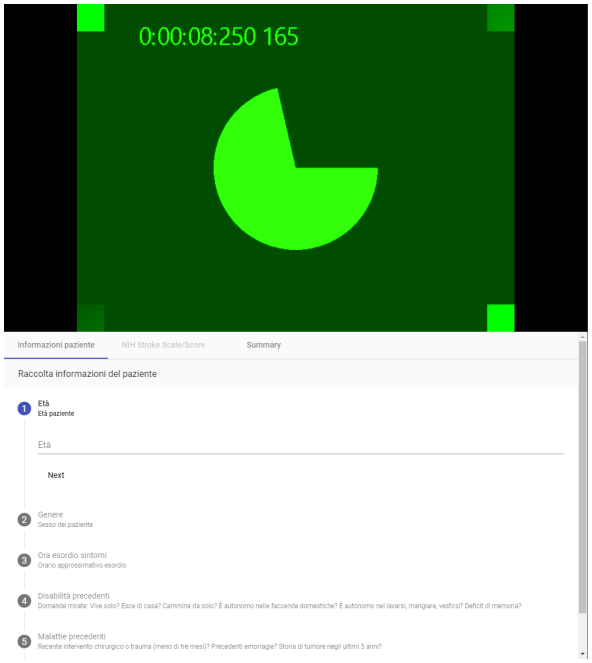


Figura 4.6: Pagina di compilazione sessione su schermi con grandezza minore di 1100px

4.5 Interfaccia operatore

L'interfaccia dell'operatore risulta essere molto minimale, proprio perché l'utente non dovrebbe essere distratto da essa e non è previsto alcun tipo di interazione. Anche in questo caso si è cercato di progettarla seguendo le regole del *Material Design*, tenendo però presente le linee guida fornite dal produttore degli *smartglasses* selezionati per questo progetto.

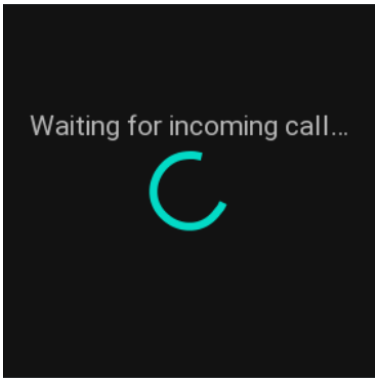


Figura 4.7: Schermata frontend client Android in attesa di ricevere una nuova richiesta.

Inizialmente l'interfaccia mostrerà all'operatore una *progress bar* in stato *undetermined* e un testo, per segnalare all'utente lo stato corrente dell'applicazione. Una volta completate le fasi preliminari, verrà modificato il testo visualizzato, segnalando che l'applicazione è in attesa di ricevere richieste, come mostrato in Figura 4.7.

Inoltre, essendo possibile trasmettere anche dati grezzi all'interno del canale *WebRTC*, si è deciso, ogni volta che lo specialista termina la compilazione di uno *step*, di inviare le informazioni sul successivo.

Queste informazioni, che comprendono principalmente il titolo e la descrizione dello *step*, saranno poi mostrate al centro della schermata, come mostrato in Figura 4.8. Si specifica inoltre che nonostante sia possibile mostrare il flusso video dello specialista, non è stato considerato necessario visualizzarlo per i seguenti motivi:

- Potrebbe distrarre l'operatore
- Il *rendering* di un video comporta un consumo di risorse più elevato di risorse ed energia
- Il display dei dispositivi *wearable* è solitamente molto piccolo e non molto efficace per la riproduzione di video

Figura 4.8: Schermata frontend android durante una chiamata.

4.6 Architettura

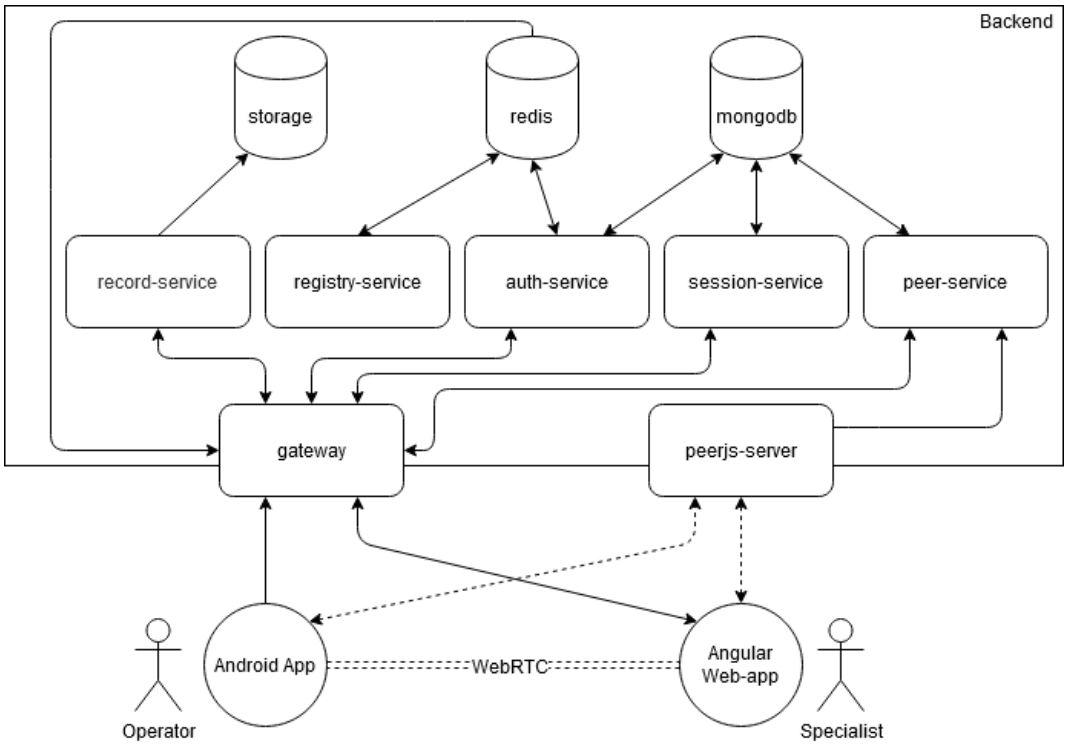


Figura 4.9: Figura che mostra una panoramica dell'architettura del sistema

Come rilevato durante l'analisi dei requisiti, il sistema risulta chiaramente essere di natura distribuita. Si è quindi suddiviso in 3 principali componenti, a seconda della loro collocazione spaziale:

- **Frontend - operatore**: Parte del sistema con cui si interfaccia l'operatore. È rappresentato dal software in esecuzione all'interno del dispositivo utilizzato dall'operatore, per esempio gli *smartglasses*.
- **Frontend - specialista**: Rappresenta la parte del sistema con cui si interfaccia lo specialista. Essendo molto eterogeneo l'insieme dei dispositivi dai quali può essere fruita è stato deciso di utilizzare una *web-app*. Questo garantisce un supporto molto esteso a diverse tipologie di apparati.
- **Backend**: Parte del sistema adibita alla gestione e mantenimento delle informazioni. Supporta i *frontend* esponendo funzionalità per effettuare operazioni, ottenere dati o supportare l'interazione fra di essi. È stato progettato utilizzando un approccio a microservizi.

In Figura 4.9 è rappresentata graficamente l'architettura del sistema, con gli artefatti che la compongono e le loro interazioni.

4.6.1 Backend

Questo componente rappresenta la parte più importante del sistema. Fornisce infatti funzionalità ad entrambi i *frontend* abilitandoli a svolgere tutte le attività per i quali sono stati concepiti. Si occupa inoltre del controllo e della gestione delle informazioni, incluso il loro salvataggio. Integra inoltre protocolli di sicurezza.

In puro approccio microservizi, ciascuna funzionalità sarà incapsulata in un servizio dedicato, le quali verranno esposte tramite *API REST*. L'interazione con il sistema, o con altri servizi interni, avverrà tramite chiamate *HTTP* alle *API*.

Si andranno ora ad introdurre più nel dettaglio tutti i microservizi che compongono il backend.

Redis

In architetture a microservizi è sempre consigliato un approccio *stateless*. Questo significa che lo stato dovrebbe sempre essere mantenuto da un servizio specifico dedicato alla sola memorizzazione dei dati, facilitandone la gestione. Per non pregiudicare le performance del sistema, lo stato dovrebbe essere mantenuto su un supporto che garantista prestazioni elevate.

È stato quindi scelto Redis[7], un *key-value store* che risiede in memoria, standard *de-facto* per questa specifica applicazione. Presenta infatti come peculiarità principale la velocità di accesso e scrittura dei dati, oltre ad avere molte funzionalità che ne permettono una facile integrazione in una architettura a microservizi. È stato utilizzato nello specifico per:

- Mantenere i record inerenti al *service discovery*
- Mantenere i dati inerenti all'autenticazione

MongoDB

Oltre ad un supporto per la gestione dello stato, il sistema richiede anche un database nel quale salvare in modo permanente i dati. Anche in questo caso, per non pregiudicare le performance del sistema, è opportuno selezionare una tecnologia che garantista performance elevate. In questo caso la scelta è ricaduta su MongoDB[8], DBMS non relazionale orientato ai documenti. Utilizzando una struttura a documenti è risultato particolarmente utile per la gestione dei modelli dei vari componenti dell'interfaccia e delle azioni dello specialista. Anch'esso risulta essere facilmente integrabile in una architettura a microservizi. Più nello specifico, lo si è utilizzato per salvare:

- Informazioni sugli utenti
- Informazioni sugli operatori disponibili
- I dati inerenti alle sessioni eseguite ed in esecuzione
- I modelli che rappresentano i *template* e le *checklist* utilizzabili per configurare l'interfaccia dello specialista

Peerjs-server

Come già detto in precedenza, per l'implementazione della comunicazione real-time è stato deciso di utilizzare *WebRTC*. Tuttavia le *API* offerte risultano essere molto a basso livello. È stato quindi deciso di utilizzare **PeerJS**[9], una libreria che semplifica notevolmente l'utilizzo delle primitive *WebRTC* e ne permette una facile integrazione offrendo un più alto livello di astrazione.

Oltretutto, per fare in modo che un client *WebRTC*, denominato *peer*, possa contattarne altri, vi è la necessità di implementare un meccanismo di coordinazione (*signaling*) tra di essi, tramite l'utilizzo di un server dedicato.

[7] Redis
[8] MongoDB
[9] PeerJS

Lo standard *WebRTC* infatti non contempla questa parte della comunicazione e perciò non impone uno specifico protocollo. Fortunatamente *PeerJS* offre già il proprio protocollo di *signaling* e un server per la gestione dei *peer*.

Il server citato è stato quindi integrato nel backend e le sue *API* esposte ad entrambi i *frontend*.

Gateway

Il servizio *gateway*, conosciuto meglio come *API gateway*, rappresenta la reificazione di un pattern comune in ambito microservizi per esporre un singolo *entrypoint* con cui comunicare con l'intera infrastruttura, facilitandone l'interazione ed esponendo una *API* ridotta e più strutturata. Essendo inoltre il servizio che gestisce tutte le richieste in ingresso, rappresenta il luogo ideale dove aggiungere logica inerente alle richieste.

Per quanto concerne questo progetto, esso è stato sfruttato per:

- Abilitare HTTPS
- Autorizzare le richieste
- Tracciare le richieste (*logging*)
- Facilitare l'interazione con il *backend*

Registry-service

Il microservizio *registry-service* viene utilizzato per la gestione del *discovery* dei servizi. Per *service discovery* si intende quella operazione per cui il sistema rileva automaticamente servizi in esecuzione nello stesso ambiente. Serve principalmente per ridurre la configurazione dei microservizi da parte dell'utente.

Questo servizio offre le seguenti funzionalità:

- Espone metodi per eseguire *query* sui servizi attivi
- Si occupa di gestire i *record* inerenti al discovery all'interno di *Redis*, che vengono creati da ogni nuova istanza di microservizio all'avvio, eliminando quelli appartenenti a servizi che non sono più disponibili (offline). Lo stato di uno specifico microservizio viene determinato attraverso un meccanismo di *heartbeat*.

Auth-service

Servizio dedicato all'autenticazione ed autorizzazione degli utenti e delle loro richieste. L'autenticazione, ovvero l'identificazione degli utenti, avviene tramite un sistema di account, i quali sono mantenuti all'interno di *MongoDB*. L'autorizzazione delle richieste viene invece gestita tramite **JWT**[10], standard molto utilizzato in questo tipo di architetture, in quanto fornisce buona sicurezza senza risultare complesso da integrare e da gestire.

Più precisamente questo servizio si occupa della generazione dei *token* **JWT**, mentre il controllo delle richieste viene eseguito dal *gateway*, il quale controlla che il *token* fornito sia presente su *Redis*. Si è scelto di utilizzare questo tipo di tecnologia per il salvataggio di suddetti dati per i seguenti motivi:

- Offre un tempo di lettura molto basso, questo fa si che il *gateway* riesca a controllare le richieste velocemente
- Offre un meccanismo nativo per definire un *TTL*[11] per le informazioni salvate su di esso (molto comodo per gestire la scadenza dei token)

Session-service

Rappresenta il microservizio che espone l'intera logica di gestione delle *sessioni*. Nello specifico permette di:

- Ottenere informazioni su una o più sessioni attive o terminate
- Ottenere *template* e *checklist*
- Creare una nuova sessione
- Terminare o concludere una sessione
- Aggiungere una nuova azione per una determinata sessione

Tutte le funzionalità sopra descritte sono disponibili tramite *API*, divise logicamente per risorsa, in puro stile *REST*.

Questo servizio si occupa anche di effettuare tutti i controlli preliminari richiesti per l'esecuzione di alcune operazioni esposte. Integra inoltre la logica di gestione di possibili errori a livello applicativo. Infine si specifica che tutte le informazioni gestite da questo servizio risiedono su *MongoDB*, così facendo è possibile limitare il rischio di perdita di dati.

[10] Json Web Token
[11] Time to live

Peer-service

È il microservizio che si occupa di mantenere aggiornata la lista di operatori attivi, ovvero dei *peer WebRTC*, in quanto il dispositivo di un operatore è identificato dal sistema come un *peer*. In particolare, il servizio confronterà periodicamente l'elenco dei *peer* sottoscritti al *peerjs-server* e l'elenco degli operatori su *MongoDB*, mantenendo allineata quest'ultima.

Offre inoltre una *API* per fare in modo che il dispositivo di un operatore possa sottoscriversi al sistema, comunicando l'identificativo del *peer* da cui è rappresentato, permettendo così allo specialista di contattarlo alla creazione di una nuova *sessione*.

Record-service

Microservizio dedicato al salvataggio delle registrazioni dei flussi audio-video al termine di una *sessione*, le quali saranno salvate in uno *storage* persistente. Si specifica che questo servizio non effettuerà la registrazione, ma sarà il *frontend* dello specialista ad occuparsene. Questa scelta è dettata dal fatto che *WebRTC* si basa su un'architettura *peer-to-peer* (*p2p*) per diminuire quanto più possibile la latenza, eliminando la presenza di un server. Questo rende quindi complicato l'accesso da parte del microservizio ai flussi multimediali, cosa che invece risulta banale all'interno del *frontend*.

4.6.2 Frontend: specialista

In fase di analisi è già stato individuato che questa parte del sistema potrà essere fruita da dispositivi eterogenei. Si è quindi scelto un approccio basato su *web-app*. Come riportato in precedenza, il *framework* scelto per la creazione della suddetta è **Angular**[12] (versione 8) e **Angular Material**[13].

Le caratteristiche principali che hanno portato a questa scelta sono:

- Performance
- Cross-platform
- Supporto a toolkit per rendere la pagina *responsive*, ovvero adattare il layout pagina alle dimensioni dello schermo
- Presenza di molti componenti utilizzabili per la creazione dell'interfaccia (prototipazione veloce)
- Facilità di integrazione con libreria *PeerJS*

[12] Angular
[13] Angular Material

4.6.3 Frontend: operatore

Al contrario del frontend dello specialista, la parte dedicata all'operatore sarà installata in un insieme molto più ristretto di dispositivi. Si è quindi deciso di implementarlo come applicazione nativa *Android*.

Le motivazioni principali sono:

- Ottimizzazione performance (molto spesso dispositivi con risorse limitate)
- Consumo energetico
- Stabilità
- Presenza di libreria nativa *WebRTC*

Il problema principale è che nonostante esista una libreria nativa *WebRTC* per *Android*, essa ovviamente non ha supporto al *signaling*. Si rende quindi necessario studiare il protocollo *PeerJS* ed implementarlo all'interno dell'applicazione nativa.

4.6.4 Aspetti non funzionali

Di seguito si discutono alcuni aspetti non funzionali considerati poi in fase implementativa:

- **Scalabilità**: l'architettura proposta offre la possibilità di scalare orizzontalmente, infatti essendo tutti i microservizi *stateless* è possibile aumentare il numero di repliche per microservizio per far fronte ad un incremento delle richieste al sistema.
- **Fault tolerance**: con la combinazione di *discovery* e *registry-service* si ha già una buona base per poter implementare meccanismi avanzati di fault tolerance. Ogni servizio presenta infatti di base una *API* per ottenere il proprio stato e quello dei servizi da cui dipende (health check). In questo modo si rende possibile implementare meccanismi di rilevazione e gestione degli errori.
- **Sicurezza**: per quanto riguarda la sicurezza del sistema, nonostante sarà inizialmente servito all'interno della intranet dell'ospedale, una rete protetta ed isolata, offre già meccanismi di autenticazione e autorizzazione. Qualora si volessero aggiungere altri meccanismi sarà possibile farlo molto agilmente incapsulandoli nel *gateway*.

Capitolo 5

Sviluppo e Validazione

In questo capitolo si descriveranno alcune scelte implementative fatte durante lo sviluppo del primo prototipo. Si andrà successivamente a riportare in che modo è stata eseguita la sua validazione e quali risultati sono stati ottenuti.

5.1 Implementazione

5.1.1 Tecnologie

In questa prima sezione si andrà a fare una panoramica generale sulle tecnologie utilizzate per lo sviluppo del prototipo.

Vert.x

La quasi totalità del backend è stata sviluppata attraverso *Vert.x*[1], framework event-driven basato sulla JVM che permette la realizzazione di microservizi altamente performanti e leggeri. Esso supporta una grande varietà di linguaggi e, grazie alla sua architettura modulare, risulta particolarmente semplice integrare e coordinare microservizi diversi. Questi ultimi sono stati sviluppati in Kotlin[2] utilizzando il modello actor-like di Vert.x basato su *verticle*[3].
Si aggiunge che, per le parti in Kotlin, si è sfruttata anche la buona integrazione del framework con le feature specifiche del linguaggio. Esempio degno di nota è quello delle *coroutine*, che hanno decisamente facilitato la gestione di tutte le operazioni asincrone all'interno dei *verticle*.

[1] Vert.x
[2] Kotlin
[3] Verticle in Vert.x

Si aggiunge inoltre che si sono utilizzati diversi componenti secondari per aggiungere funzionalità ai microservizi o abilitare interconnessione con altri servizi in modo agevole. I moduli secondari utilizzati sono riportati nel seguente elenco:

- Redis client
- MongoDB client
- Vert.x Service Discovery
- Vert.x Config
- JWT auth
- MongoDB auth
- Vert.x Health Check

WebRTC (PeerJS, RecordRTC)

WebRTC[4] è un protocollo *peer-to-peer* basato su *Websockets*[5], permette abbastanza agilmente di aggiungere funzionalità di comunicazione real-time all'interno di applicazioni web o mobili. Supporta flussi audio, video e dati generici che possono essere inviati tra *peers* (client). Uno dei maggiori pregi è dovuto al fatto che si basa su tecnologie implementate come *open web standard*, disponibili tramite semplici *API Javascript* in tutti i maggiori browser. Presenta inoltre *API* native per *iOS* ed *Android*.

Per quanto concerne *PeerJS*[6], è una libreria scritta in *Typescript* utilizzabile in contesti web *client-side* che facilita ulteriormente l'integrazione di *WebRTC*, esponendo una *API* a più alto livello. Integra inoltre un protocollo per il *signaling* tra i *peer* (parte non considerata nello standard *WebRTC*) e fornisce anche un'implementazione completa del server per la gestione di quest'ultimo.

Infine, per quanto riguarda la gestione della registrazione dei flussi multimediali, si è utilizzata la libreria *RecordRTC*[7]. Essa espone le funzionalità del componente *WebRTC MediaRecorder*, sempre tramite una *API* molto comoda ed intuitiva, il quale abilita il client a registrare e aggregare i vari flussi che i *peer* si scambiano durante una videoconferenza.

[4] WebRTC
[5] Websockets
[6] PeerJS
[7] RecordRTC

Angular

Angular[8] è un framework open source per lo sviluppo di applicazioni web, manutenuto principalmente da Google. Una delle caratteristiche principali è l'utilizzo di *template* per la definizione delle viste, approccio che permette una facile prototipazione delle viste. Possiede inoltre diverse astrazioni che permettono di separare in modo efficace il codice.

MongoDB

MongoDB[9] è un DBMS non relazionale, orientato ai documenti. È di tipo NoSQL e presenta una struttura formata da documenti in stile JSON con schema dinamico, rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce. È un software open source e si integra molto bene con architetture a microservizi (tramite il supporto a *sharding* e *bilanciamento dei dati*).

Redis

Redis[10] è un *key-value store* open source che risiede in memoria, che può essere utilizzato come database, cache o message broker. Ha supporto nativo a replicazione, persistenza su disco, *high availability* (uptime garantito più alto rispetto alla norma) e partizionamento automatico. Sfruttando la memoria di sistema, risulta essere estremamente veloce nelle operazioni di lettura e scrittura dei dati.

Docker

Docker è un progetto open source che automatizza il *deployment* di applicazioni all'interno di contenitori software, fornendo un'astrazione aggiuntiva ottenuta tramite virtualizzazione a livello di sistema operativo di Linux.

È stato utilizzato per facilitare il deployment di tutto il backend e del frontend dello specialista. Il supporto a livello di progetto è stato aggiunto tramite la scrittura di *Dockerfile*, un documento di testo che contiene tutti i comandi che l'*engine docker* interpreta per assemblare l'immagine, la quale sarà successivamente utilizzata per creare container. Altro accorgimento è stato quello di rendere tutti i microservizi *stateless*, implementare meccanismi di *service discovery*, gestire dinamicamente le configurazioni e utilizzare servizi già orientati a questo tipo infrastruttura come MongoDB e Redis.

[8] Angular: features
[9] MongoDB
[10] Redis

Infine si specifica che è stato creato anche un file *docker-compose* grazie al quale è possibile eseguire il *deploy* di tutti i servizi tramite un singolo comando. Questo file, sempre interpretabile tramite Docker, permette di definire tramite una sintassi specifica i servizi di cui si intende fare il *deployment* e le loro relazioni di dipendenza.

5.1.2 Backend

Il *backend* è stato scritto interamente in *Kotlin*, utilizzando il *framework Vert.x* e tutti i moduli secondari riportati in precedenza. Grazie ad uno di questi ultimi, è stato possibile aggiungere supporto diretto all'utilizzo di *coroutine*[11], ovvero componenti nativi di questo linguaggio per abilitare l'implementazione di funzioni asincrone non bloccanti.

Si è utilizzato inoltre il tool *Gradle*[12] sia per la gestione delle dipendenze sia per l'automazione del *build* e *deploy* del sistema. Inoltre si è scelto di suddividere il progetto in moduli, uno per ogni microservizio, oltre a quelli condivisi (funzionalità comuni), per favorire appunto la modularità e il riuso del codice. Tutti i verticle estendono infatti da un verticle di base implementato nel package *common*, che fornisce già integrazione con *discovery*, *health check*, *gestione configurazione* ed esecuzione di server http embedded per la gestione dell' *API REST* esposta.

Service discovery

Il *service discovery* è stato implementato tramite l'utilizzo del package nativo *Vert.x Service Discovery*, utilizzando *Redis* come backend, ovvero database temporaneo per il salvataggio dei servizi attivi (record, secondo la nomenclatura Vert.x). Questo, combinato con il *registry-service*, permette al sistema di funzionare anche all'interno di *Docker* e, più in generale, di reagire alla comparsa e gestione di nuove istanze di servizi o alla scomparsa di quelle già presenti.

Autenticazione ed autorizzazione

Come già detto in precedenza l'autenticazione del sistema è demandata al package *MongoDB auth* il quale salva le credenziali degli account all'interno di MongoDB (seguendo gli ultimi standard in termini di sicurezza). L'autorizzazione è invece implementata tramite package *JWT auth*, ovvero tramite JWT

[11] Kotlin: coroutine overview
[12] Gradle

token, ed è supportata dall'*auth-service*. *Redis* viene utilizzato come database temporaneo dove salvare i token ancora validi.

Per migliorare la *user experience* dello specialista è stato aggiunto il supporto ai refresh token, per fare in modo che una volta scaduto il token di autorizzazione, il frontend possa automaticamente richiederne uno nuovo senza forzare l'utente ad effettuare un nuovo login.

Gestione configurazione

La gestione della configurazione dei microservizi Vert.x è stata estesa tramite package *Vert.x Config*. Questo ha abilitato la parametrizzazione, tramite variabile d'ambiente (VERTX_PROFILE), della selezione della configurazione da caricare all'avvio del microservizio, il che si è rilevato particolarmente utile per quanto riguarda il *deploy* su Docker, abilitando la selezione della configurazione specifica direttamente all'interno del file *docker-compose*, senza agire sull'implementazione dei microservizi.

5.1.3 Frontend Angular

Per quanto riguarda il frontend *Angular*, l'utilizzo di questo framework ha reso molto agevole la sua implementazione. Il *templating* è risultato particolarmente utile ed efficace per la gestione delle viste e la modularità del codice offerta tramite i cosiddetti componenti ha permesso di gestire abbastanza agevolmente il rendering dinamico delle checklist e di tutti i controlli al loro interno.

Si specifica come si sia fatto largo uso di *servizi Angular*[13] per la gestione di tutte le chiamate al backend, suddividendoli in categorie, una per ogni risorsa esposta. Sono stati utilizzati anche per incapsulare la logica di gestione di *PeerJS* e *RecordRTC*.

Infine, per la gestione dell'autenticazione, si è fatto invece uso di *http interceptor*[14], *guard*[15] e *LocalStorage*[16].

5.1.4 Frontend Android

Il frontend Android è stato sviluppato completamente in Kotlin. Per favorire il riuso è stato deciso di creare una libreria a parte per il supporto a *WebRTC* e *PeerJS*, che è stata inclusa solo successivamente al progetto dell'applicazione vera e propria. La libreria offre un client *WebRTC*, con supporto

[13] Angular: architecture services
[14] Http interceptor
[15] Route guards
[16] LocalStorage

a *signaling* tramite il protocollo definito in *PeerJS*. Il client ha pieno supporto all'instaurazione di canali per flussi audio/video o dati. Supporta inoltre la ricezione e la gestione di richieste da parte di altri peer. È anche in grado di gestire il rendering dei flussi video e audio remoti o locali. Per facilitare l'integrazione espone queste funzionalità tramite funzioni asincrone, ovvero implementate tramite coroutine.

Sottoscrizione operatore

In questo primo prototipo l'applicazione ha già assegnate a tempo di compilazioni credenziali accettate dal sistema, con le quali ha la possibilità di eseguire richieste verso il backend. Oltre alle credenziali, che si specificano essere collegate ad un account con un ruolo con privilegi minimi per eseguire le operazioni richieste, l'applicazione conosce già il nome dell'operatore e ha già impostata una descrizione del dispositivo per facilitare la selezione da parte dello specialista.

Così facendo l'applicazione una volta avviata si occupa automaticamente di eseguire tutte le operazioni preliminari senza la necessità di interazione con l'utente.

5.2 Validazione

Sarà ora introdotto il processo di validazione del sistema e i risultati ottenuti.

5.2.1 Valutazione componenti

La valutazione dei componenti è stata seguita senza la presenza degli esperti del dominio. Lo scopo di questa fase era quello di valutare i singoli componenti del sistema per evidenziare eventuali criticità prima della validazione vera e propria.

Backend

Per la valutazione dei singoli microservizi è stato utilizzato *Postman*[17], un software con interfaccia visuale molto comodo per eseguire chiamate con diversi protocolli di rete. Dopo aver definito l'interfaccia *REST* di tutti i microservizi è stata generata una richiesta *Postman* per ogni *endpoint* esposto da ciascun servizio. Questo ha permesso di facilitare e velocizzare lo *unit testing* di ciascun microservizio.

[17] Postman

Essendo disponibile anche il deployment su *Docker*, una volta stabilizzati i microservizi, è stato eseguita la stessa procedura descritta in precedenza, ma utilizzando i microservizi all'interno di *container* e non come processi in esecuzione nella macchina locale.

L' *integration testing*, ovvero la valutazione delle interazioni tra le componenti del backend, è stato eseguito sempre con la stessa procedura, ma effettuando le chiamate al *gateway*, piuttosto che ai servizi singoli, in questo modo si è potuto valutare la gestione dell'*autenticazione*, *discovery* e del *routing* delle richieste.

Si specifica infine come *Postman* abbia supporto a diversi standard di autenticazione (incluso JWT) e presenti la possibilità di definire variabili condivise tra più richieste ed eseguire script prima o dopo una richiesta. Tutte queste funzionalità si sono rivelate molto utili per gestire automaticamente la fase di login e i vari token per l'autenticazione delle richieste.

Applicazione Android

L'applicazione Android, sviluppata come frontend per l'operatore, non ha richiesto *unit testing* particolare. Questo perché l'applicazione sfrutta le funzionalità esposte dalla libreria creata per la gestione del client WebRTC, che era già stata a sua volta testata estensivamente durante la fase di sviluppo tramite la creazione di un sistema *ad-hoc*, totalmente avulso dal sistema finale.

Questo sistema comprendeva le seguenti componenti:

- una semplice pagina web che sfruttava la libreria PeerJS per ricevere/effettuare chiamate WebRTC, con un'interfaccia minimale per visualizzare/inviare dati e mostrare i flussi audio-video locali e remoti
- un'istanza peerjs-server
- un'applicazione Android specifica per il testing della libreria sviluppata che esponeva le stesse funzionalità offerte dalla pagina web

Grazie a questo sistema si è quindi potuto effettuare il testing dei singoli componenti e delle loro interazioni per le funzionalità offerte da PeerJS, ovvero: gestione chiamate audio/video, scambio dati e gestione connessione a signaling server.

Web-app Angular

Il testing del frontend dello specialista si è effettuato durante tutto lo sviluppo, utilizzando principalmente le funzionalità di *live rebuilding* di Angular, che permette di interagire con l'applicazione anche durante la fase di svi-

luppo ed eseguire compilazione e aggiornamento alla modifica dei sorgenti, particolarmente utile per validare i template e la visualizzazione dei dati.

Si è inoltre eseguito un testing specifico per la parte di gestione dei token di autenticazione e chiamate al backend (passando attraverso il gateway), interagendo manualmente con il sistema e simulando alcune situazioni specifiche, come per esempio la scadenza del token di autenticazione.

5.2.2 Valutazione sistema

La valutazione completa del sistema, ovvero l'*acceptance testing*, è stato eseguito agendo sul frontend specialista tramite una sequenza predefinita di azioni e analizzando il comportamento del sistema per rilevare eventuali problemi. La sequenza è stata studiata in modo tale da dimostrare che fossero rispettati tutti i punti definiti nello scenario di validazione (Sezione 3.2.6).

Il sistema è stato valutato sia utilizzando processi nella macchina locale, sia eseguendo il deploy su Docker del backend e del frontend dello specialista, utilizzando un'*immagine Nginx*[18] come *web-server* per servire l'applicazione Angular.

Questa valutazione è stata effettuata prima senza la presenza dello specialista, in un ambiente controllato, ovvero utilizzando una rete locale wireless dedicata e una macchina desktop per ospitare i container Docker. Inoltre si è effettuato il deploy dell'applicazione dell'operatore sia su uno *smartphone* Android sia sugli *smartglasses*, principalmente per rilevare eventuali problemi causati da livello diverso di performance.

Quest'ultimo passo è stato fondamentale per individuare alcuni problemi legati alle performance degli *smartglasses*, che ha richiesto la modifica del formato di cattura del flusso video. In particolare il dispositivo non era in grado di gestire la cattura di un video a risoluzione 1280x720 pixel a 30 frame al secondo. Dopo una serie di prove è stata individuata una configurazione che non risultasse in rallentamenti, ovvero 640x480 pixel a 25 frame al secondo.

Questa modifica ha causato un calo per quanto riguarda la qualità del flusso video, verrà quindi valutato insieme allo specialista, qualora non risultasse soddisfacente, la modifica ulteriore della configurazione.

Valutazione con esperto del dominio

Successivamente si è passati alla validazione del sistema in presenza di un esperto del dominio. Grazie alla disponibilità del dottor Longoni si è fissato un incontro nel suo studio, all'interno dell'Ospedale M.Bufalini di Cesena,

[18] Nginx docker image

per mostrare il prototipo, validarlo e ricevere eventuali feedback per il suo miglioramento.

Si specifica come in questo incontro il *deployment* sia stato effettuato in modo diverso. Si è infatti utilizzato la funzionalità *hotspot personale* di uno *smartphone* (iPhone 8) per la creazione di una rete dedicata senza limitazioni, alla quale sono stati collegati un *notebook* (Macbook Pro Late 2013), che ospitava il backend all'interno di un cluster Docker, e gli *smartglasses* (Vuxiz Blade).

Nonostante il diverso scenario il comportamento del sistema non ha subito notevoli variazioni, se non per un incremento nella latenza del flusso dati WebRTC, legato probabilmente all'utilizzo di una rete decisamente meno performante.

Inizialmente al dott. Longoni è stato presentato il frontend dello specialista, ed è stato richiesto di navigare la web-app, esplorando le varie funzionalità del sistema mentre venivano descritte. Dopodiché si è passati alla presentazione degli *smartglasses* e del frontend dell'operatore.

È stato quindi richiesto al dottore di indossare il dispositivo, grazie al quale ha potuto visualizzare l'applicazione Android direttamente tramite il display integrato nella lente.

Infine si è mostrato l'intero processo di gestione simulando una sessione di refertazione, facendo in modo che il dottore provasse il sistema impersonando entrambi i ruoli previsti: specialista ed operatore.

5.2.3 Risultati

Nonostante il sistema sia ancora in fase prototipale, l'esito della validazione è stato comunque positivo, sia perché non si sono verificati problemi durante l'utilizzo sia perché sono stati rispettati i requisiti.

Inoltre, dopo aver provato il sistema, il dottor Longoni ha fornito alcuni commenti positivi sul prototipo, elencati in seguito:

- gli *smartglasses* risultano molto efficaci, principalmente per il ridotto impedimento che provocano e per la facilità con cui possono essere indossati e configurati
- il frontend dell'operatore mostrando le domande da chiedere al paziente facilita decisamente le attività e velocizza il processo
- risulta positivo che l'operatore non debba interagire con il dispositivo indossato
- il frontend dello specialista risulta molto efficace durante la valutazione perché velocizza le operazioni e previene errori dello specialista

- questo sistema abilita ad una più veloce valutazione di quanto si discosta la scelta dello specialista rispetto ai suggerimenti ottenuto tramite l'applicazione rigorosa dell'algoritmo

Oltre a questo il dottore ha individuato alcuni possibili miglioramenti da effettuare principalmente per quanto riguarda il frontend dello specialista, che sono riepilogati qui sotto:

- sarebbe utile ingrandire la dimensione del testo per quanto riguarda il titolo e la descrizione dei singoli step
- sarebbe comodo visualizzare i risultati delle varie fasi nella sezione di riepilogo
- potrebbe essere utile mostrare sempre il cronometro in sovrimpressione, e non solo nella schermata di riepilogo
- all'interno dello step potrebbe essere utile prevedere un componente di tipo *checkbox*, così facendo alcuni step della fase di raccolta delle informazioni del paziente potrebbero essere renderizzate come un insieme di checkbox al posto di un campo libero compilabile
- per alcuni *step* con selezione (per esempio lo *step* inerente alla *Modified Rankin Scale*) potrebbe essere utile visualizzare un *dialog* per notificare/-ricordare allo specialista che una particolare selezione possa presupporre un approccio diverso per la cura del paziente
- in caso di *step* con *field* di tipo *datetime*, in alcuni casi potrebbe non essere necessario richiedere l'orario, ma solo la data

Mentre per quanto riguarda il frontend dell'operatore, è stata avanzata l'ipotesi di permettere la visualizzazione dei flussi video dello specialista direttamente all'interno dello schermo degli *smartglasses*, questo per rendere l'interazione più naturale, potendo non solo sentire l'interlocutore remoto, ma anche vederlo.

Il dottore ha tuttavia accettato l'implementazione corrente, dopo che gli è stato esposto il motivo per cui non è stato implementato in questo modo. Si è infatti deciso di non mostrare il flusso video principalmente perché avrebbe comportato un consumo più elevato della batteria e poiché, essendo lo schermo decisamente piccolo, avrebbe occupato buona parte dello schermo, rendendo più difficile la collocazione e visualizzazione delle informazioni testuali per lo step da eseguire.

Per concludere, a riprova del fatto che il sistema sviluppato abbia dato riscontri positivi, si fa notare come le modifiche richieste siano confinate ad

aspetti minori del frontend dello specialista, nello specifico la configurazione grafica, aspetto prettamente soggettivo e modificabile velocemente.

5.2.4 Criticità rilevate

Durante le varie fasi di validazione del sistema sono state riscontrate le seguenti criticità, collegate direttamente o indirettamente con il sistema:

- Il sistema tratta dati sensibili dei pazienti, deve quindi garantire un certo livello di sicurezza
- Per poter utilizzare il sistema è richiesto il consenso firmato del paziente, qualora sia cosciente e nella condizione di poterlo esprimere
- Potrebbe risultare limitante l'autonomia degli *smartglasses*. Infatti, nonostante il *frontend* dell'operatore sia stato ottimizzato per limitare il consumo di energia, sia a causa della dimensione e capacità della batteria sia per il tipo di operazioni eseguite (registrazione e trasmissione di flussi video) potrebbe essere necessario ricaricare il dispositivo al termine di ciascuna sessione

Capitolo 6

Conclusioni

La telemedicina risulta essere un tassello molto importante per migliorare *efficienza* ed *efficacia* dei processi sanitari. In particolare, dispositivi con funzionalità avanzate, in grado di essere indossati, supportati da un' architettura robusta di servizi dietro le quinte, permette di velocizzare notevolmente le attività e, allo stesso tempo, di minimizzare gli errori.

Per quanto riguarda l'ambito specifico di questo sistema, le attività di valutazione di pazienti affetti da ictus, essendo *tempo-dipendenti*, risultano un caso perfetto di applicazione di questa concezione. Sia per ridurre il tempo di esecuzione delle attività sia per diminuire gli errori, guidando l'utente durante tutto il processo e abilitando la cooperazione remota con personale più qualificato.

Tuttavia, un ostacolo a questo nuovo approccio è dato dallo scetticismo da parte del personale medico sul reale beneficio acquisito tramite l'utilizzo di questi sistemi, poiché non viene identificato come un aiuto, ma piuttosto ostacolo o un'attività aggiuntiva da eseguire. Durante la progettazione di questo sistema è stata posta particolare enfasi sullo studio dell'interazione *uomo-sistema* e, ove possibile, di prediligere la facilità d'uso rendendolo più *intuitivo*. Si è quindi cercato di raggiungere questo obiettivo riducendo il più possibile l'interazione con il sistema.

Al termine della fase sviluppo e di validazione, si può affermare che il sistema abbia rispettato i requisiti richiesti, considerando anche i limiti e le esigenze dell'utilizzatore, pur fornendo un supporto notevole al processo di refertazione e consulto.

6.1 Sviluppi futuri

Per concludere, saranno elencate alcune possibilità di sviluppi futuri che potrebbero portare ad un ulteriore miglioramento del sistema:

- Esecuzione di una validazione più articolata del sistema seguendo la prassi relativa ai protocolli di sperimentazione in ambito sanitario, che coinvolga operatori sanitari ed eventualmente pazienti
- Integrazione con sistemi esistenti per abilitare la visualizzazione delle **informazioni sul paziente**
- Integrazione di tecniche di **Machine Learning** o **Computer Vision** per guidare il processo di suggerimento dell'iter da intraprendere in base ai dati raccolti o analizzando il comportamento del paziente
- Per ridurre in modo ancora più drastico l'interazione necessaria da parte dell'utilizzatore, il sistema potrebbe integrare il supporto ad **assistenti virtuali** per abilitare l'interazione tramite comandi vocali
- Identificare l'operatore tramite la scansione di un *qr-core*
- Esportare il sistema all'esterno, ovvero nell'ambito pre-ospedaliero, esercizio di carattere prettamente sistemistico in quanto il sistema è stato già pensato per gestire componenti distribuiti. Potrebbe portare ad una ulteriore e notevole riduzione dei tempi di trattamento di un paziente

Ringraziamenti

A conclusione di questo lavoro desidero ringraziare profondamente tutti coloro che hanno contribuito, direttamente o indirettamente, a questo lavoro.

Innanzitutto la mia famiglia per il supporto che mi hanno dato durante l'intero percorso di studi. Successivamente vorrei ringraziare il prof. Ricci per avermi proposto questo progetto molto stimolante sia per quanto riguarda gli aspetti puramente tecnici sia per l'ambito al quale verrà applicato. Inoltre lo ringrazio per la passione e la dedizione mostrata sia durante le sue lezioni sia per quanto riguarda i progetti di tesi che ho avuto il piacere di svolgere insieme.

Ringrazio inoltre l'ing. Croatti per il grande supporto, la disponibilità e i consigli durante tutto l'arco di tempo nel quale mi son dedicato a questo progetto. Specialmente per quanto riguarda la coordinazione con tutte le persone ed enti coinvolti.

Un ringraziamento speciale al dott. Longoni per l'interesse mostrato verso questo progetto, per la disponibilità nel fissare i vari incontri svolti in ospedale, per il supporto fornito che ha permesso di capire più a fondo il dominio applicativo e per i suggerimenti dati in fase di validazione, che hanno permesso di migliorare ulteriormente il sistema.

Per concludere, vorrei ringraziare anche tutti i miei colleghi e amici che mi hanno sostenuto durante questo lungo percorso di studi.

Bibliografia

[1] WHO Group Consultation *Telemedicine: Opportunities and developments in Member States*, 2010.

[2] WHO Group Consultation on Health Telematics *A health telematics policy in support of WHO's Health-for-all strategy for global health development: report of the WHO Group Consultation on Health Telematics*, Geneva, 1997.

[3] European Commission *Market study on telemedicine*, 2018.

[4] European Commission *eHealth Action Plan 2012-2020 – innovative healthcare for the 21st century*, 2012.

[5] European Commission *On telemedicine for the benefit of patients, healthcare systems and society*, 2008.

[6] Ministero della salute *Telemedicina: Linee di indirizzo nazionali*, 2008.

[7] Kevin M. Barrett and Michael A. Pizzi and Vivek Kesari and Sarvam P. TerKonda and Elizabeth A. Mauricio and Scott M. Silvers and Ranya Habash and Benjamin L. Brown and Rabih G. Tawk and James F. Meschia and Robert Wharen and William D. Freeman *Ambulance-based assessment of NIH Stroke Scale with telemedicine: A feasibility pilot study*, 2017.

[8] Marc Krolczyk, Mike Telek *Vuzix Blade^{TM} Dev Kit User Experience (UX) Design Guidelines* 2018.

[9] S. Jhajharia, S. K. Pal, S. Verma *Wearable Computing and its Application*, 2014.

[10] M. Chan, D. Est'eve, J.-Y. Fourniols, C. Escriba, E. Campo *Smart wearable systems: Current status and future challenges* Elsevier, 2012.

[11] P. Lukowicz, T. Kirstein, G.Troster *Wearable Systems for Health Care Applications*, Schattauer GmbH, 2004.

[12] W. Barfield *Fundamentals of Wearable Computers and Augmented Reality*, CRC Press, 2016.

[13] W. Seongmin *Inside Vert.x. Comparison with Node.js*, 2013.

[14] J. Kim *Understanding Vert.x Architecture - Part II*, 2013.

[15] Johnston, Alan B. and Burnett, Daniel C. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, 2012.

[16] F. Fund, C. Wang, Y. Liu, T. Korakis, M. Zink, S. S. Panwar *Performance of DASH and WebRTC Video Services for Mobile Users*, 2013.

[17] W3C *WebRTC 1.0: Real-time Communication Between Browsers - W3C Candidate Recommendation*, https://www.w3.org/TR/webrtc/, Accessed February 2020.

[18] Sam Dutton *Getting Started with WebRTC*, https://www.html5rocks.com/en/tutorials/webrtc/basics/, Accessed December 2019.

[19] Sam Dutton *WebRTC in the real world: STUN, TURN and signaling*, https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/, Accessed December 2019.

[20] Chris Richardson, *Microservices*, https://microservices.io/, Accessed February 2020.