ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

TELEMEDICINA E WEARABLE COMPUTING A SUPPORTO DEL PERSONALE SANITARIO PER LA DIAGNOSI DELL'ICTUS: IL PROGETTO TELESTROKE COME CASO DI STUDIO

Elaborato in
SISTEMI EMBEDDED E INTERNET OF THINGS

Relatore
Prof. ALESSANDRO RICCI

Co-relatore
Dott. Ing. ANGELO CROATTI

Presentata da
ELENA YAN

Anno Accademico 2020 – 2021

PAROLE CHIAVE

Wearable Computing

Smartglasses

Telemedicina

Teleconsulto

TeleStroke

Usabilità

alla mia famiglia

Indice

Introduzione

Grazie al rapido sviluppo delle tecnologie, sono stati introdotti nuovi strumenti e opportunità a supporto del sistema sanitario. In particolare, la telemedicina ne ha trovato un ampio utilizzo negli ultimi anni, consentendo la connessione e la comunicazione fra medici, professionisti e pazienti situati in località differenti e di superare le problematiche legate alla scarsa distribuzione delle risorse nel territorio.

Infatti ospedali che si trovano nelle zone periferiche non sempre hanno a disposizione risorse e medici specializzati per gestire i pazienti con particolari patologie. La telemedicina può risultare particolarmente utile in queste situazioni, per esempio si potrebbe attivare una videoconferenza con i medici di ospedali centrali, ed effettuare una diagnosi del paziente a distanza. Questo permette di evitare perdite di tempo negli spostamenti e può portare a una riduzione dei rischi soprattutto per i pazienti con patologie a tempo dipendenti, come l'ictus celebrale, dove il tempo è un fattore molto importante.

È in tale ambito che si inserisce il progetto "TeleStroke", sviluppato dal gruppo di ricerca dell'Università di Bologna in collaborazione con l'Ospedale Bufalini di Cesena, coinvolgendo l'Unità Operativa di Neurologia e AUSL Romagna. È un sistema di telemedicina, in particolare offre il servizio di teleconsulto e telecooperazione tra neurologi e operatori sanitari durante la fase di diagnosi del paziente colpito dall'ictus celebrale. Il sistema aiuta anche a quantificare la severità dell'ictus e a determinare l'iter da seguire per la cura del paziente, basandosi su strumenti di valutazione come l'NIHSS.

Inoltre, utilizzando dispositivi *wearable*, possono portare ulteriori vantaggi al fine di migliorare l'efficienza e l'efficacia del sistema. In particolare, nel progetto di riferimento è stato scelto di utilizzare lo *smartglasses* come hardware per eseguire l'applicazione dell'operatore sanitario. Sono degli occhiali intelligenti che possono facilmente registrare flussi video e audio senza impedimento, permettendo all'operatore sanitario di inquadrare sempre il paziente. Allo stesso tempo, possono fornire informazioni utili da mostrare davanti agli occhi dell'operatore e possono essere utilizzati con una minima interazione, lasciando le mani completamente libere per poter aiutare il paziente.

Il progetto "TeleStroke" è stato sviluppato una versione prototipale e l'o-

biettivo della tesi consiste nel refactoring del progetto già esistente, con una maggiore attenzione sulla parte wearable, aggiungendo nuove funzionalità al fine di potenziare e migliorare il sistema. Dato il delicato contesto di utilizzo del progetto, l'applicazione deve fornire informazioni utili e deve essere progettata in modo da non creare ulteriori difficoltà all'operatore. Per il refactoring del progetto, si è tenuto in considerazione delle varie differenze che presentano gli *smartglasses* tra gli smartphone o i computer, cercando di rendere l'applicazione efficiente e *usabile*, ponendo l'utente sempre al centro dell'attenzione.

Durante tutta la fase di sviluppo, sono state effettuate delle prove per verificare il corretto funzionamento dei componenti sviluppati e i problemi manifestati sono stati corretti.

La presente tesi è articolata in cinque capitoli:

- Nel primo capitolo vengono approfonditi i principi base che stanno dietro al progetto, soffermandosi sui concetti di telemedicina e wearable computing.
- Nel secondo capitolo viene fornita una panoramica generale dello stato attuale del sistema, e a seguire gli obiettivi della tesi e l'analisi dei requisiti.
- Nel terzo capitolo vengono discussi i principi e le linee guide per la progettazione di applicazioni wearable, con una particolare enfasi sulle indicazioni e linee guide fornite dalle varie case produttrici di smartglasses.
- Nel quarto capitolo vengono descritte le scelte progettuali e implementative delle varie funzionalità aggiunte per potenziare il sistema.
- Nel quinto capitolo tratta la validazione del sistema, testando il corretto funzionamento dei componenti, e per concludere con le osservazioni finali e sviluppi futuri.

Capitolo 1

Introduzione alla Telemedicina e al Wearable Computing

L'obiettivo di questo capitolo è di introdurre i concetti base che stanno dietro al progetto di analisi e miglioramento della tesi.

In particolare verrà illustrata in primo luogo il concetto della Telemedicina: si parte dalla sua definizione, per poi introdursi nelle opportunità che offre, ed infine i principali vantaggi che possono derivare.

Successivamente verrà esaminato il concetto di *Wearable computing*, focalizzandone in particolare la sottocategoria di dispositivi *Eyewear*. Verranno analizzati le loro caratteristiche, i principali dispositivi e per poi concludere con una panoramica sulle principali applicazioni in ambito sanitario.

1.1 Telemedicina

Il rapido sviluppo della tecnologia ha permesso di introdurre nuove opportunità nell'ambito sanitario a supporto delle modalità tradizionali. Negli ultimi anni e in particolare nel periodo dell'epidemia, è cresciuto notevolmente l'uso della telemedicina, la quale consente una connessione più facile tra pazienti, medici e professionisti che si trovano in località differenti.

1.1.1 Definizione

Attualmente ci sono differenti definizioni di Telemedicina, una abbastanza completa è quella fornita dal WHO, *World Health Organization*:

> "La telemedicina è l'erogazione di servizi sanitari quando la distanza è un fattore critico, per cui è necessario usare, da parte degli operatori, le tecnologie dell'informazione e delle telecomunicazioni,

> al fine di scambiare informazioni utili alla diagnosi, al trattamento e alla prevenzione delle malattie e per garantire un'informazione continua agli erogatori di prestazioni sanitarie e supportare la ricerca e la valutazione della cura." [9]

Anche il *Ministero della Salute* adotta una definizione simile:

> "Per Telemedicina si intende una modalità di erogazione di servizi di assistenza sanitaria, tramite il ricorso a tecnologie innovative, in particolare alle *Information and Communication Technologies* (ICT), in situazioni in cui il professionista della salute e il paziente (o due professionisti) non si trovano nella stessa località." [6]

Dalle definizioni riportate, si può affermare che la telemedicina è una nuova modalità con cui è possibile fornire servizi sanitari da remoto attraverso lo scambio di informazioni mediche. Le informazioni vengono scambiate con l'utilizzo di tecnologie di comunicazione e possono includere dati, testi, audio, immagini e video.

La telemedicina non sostituisce la modalità tradizionale, ma lo assiste fornendo nuovi approcci con l'obiettivo di migliorare la qualità e l'efficienza dell'assistenza sanitaria. Inoltre tale modalità non si limita soltanto nello scambio o nell'analisi dei dati, ma offre anche la possibilità di migliorare il protocollo di cura, in quanto i medici o professionisti di località differenti possono interagire e collaborare in tempo reale per fornire una cura adatta al paziente.

A tal proposito, in Italia con la conferenza Stato-Regioni svolta nel dicembre del 2020 è stato deciso che tutte le attività di telemedicina siano riconosciute ufficialmente dal Ministero della Salute come prestazioni del Servizio Sanitario Nazionale. [7] Questo significa che le prestazioni erogate a distanza sono considerate a pari livello di quelle erogate in modalità tradizionale.

1.1.2 Opportunità e ambiti di applicazione

Sono presenti vari campi e settori medici in cui la telemedicina risulta utile, come per esempio la teleassistenza, la teleriabilitazione, il teleconsulto e la teledidattica, ecc. Di seguito ne vengono descritte le principali applicazioni di telemedicina. [7, 9]

- **Emergenze sanitarie**. I soccorritori possono gestire direttamente pazienti arrivati al pronto soccorso anche in assenza di specialisti, mettendosi in contatto con medici e professionisti di altri ospedali. Questo permette di evitare perdite di tempo negli spostamenti e di poter intervenire tempestivamente. Sono aspetti molto importanti soprattutto per pazienti affetti da patologie *tempo-dipendenti*.

A tale scopo, molte regioni dell'Italia stanno adottando un modello organizzativo detto *hub-spoke*, dove nei centri *hub* sono presenti medici e professionisti specializzati, mentre gli ospedali del territorio, chiamati *spoke*, hanno risorse limitate. Specialisti e professionisti dei centri *hub*, forniscono servizi di supporto come il teleconsulto o la televisita specialistica agli *spoke*, con l'obiettivo di ottimizzare i tempi.

- **Patologie rilevanti**. La telemedicina è utile anche per la gestione delle patologie rilevanti e/o casi clinici. Infatti permette di limitare lo spostamento da parte dei pazienti fragili e di offrire servizi per il controllo dei parametri vitali dei pazienti, per poi analizzarli allo scopo di fornire cure adatte.

- **Accessibilità ai servizi diagnostici a distanza**. È possibile erogare servizi sanitari anche per pazienti che si trovano nelle zone distanti dal centro urbano, come isole, montagne, mari, ecc. consentendo di usufruire servizi equamente distribuiti da tutte le zone.

- **Controllo e monitoraggio a distanza**. Oggi sono presenti molti dispositivi a supporto della telemedicina, che includono funzionalità di monitoraggio dei parametri vitali del paziente con lo scopo di prevenire o di evitare il peggioramento delle malattie.

1.1.3 Classificazione dei servizi

La telemedicina può offrire tantissimi servizi che possono essere raggruppati in diverse tipologie. A tal fine le linee guida proposte dal Ministero della Salute [6] forniscono la seguente classificazione, in base al tipo di servizio fornito e ai possibili impieghi.

Telemedicina specialistica

La telemedicina specialistica comprende tutte le modalità di erogazione dei servizi sanitari a distanza. Possono coinvolgere pazienti, medici o altri professionisti. In base alle modalità e agli attori interessati, si possono individuare le seguenti sottocategorie di telemedicina specialistica:

- **Televisita**: si tratta di un'attività medica, che consiste in una visita effettuata a distanza tra il medico e il paziente con sistemi di videochiamata. La televisita può portare alla prescrizione di farmaci o di cure.

- **Teleconsulto**: si tratta di un'attività in cui un professionista si confronta a distanza con altri medici per discutere e collaborare sulla situazione clinica del paziente, riferendosi ai dati e informazioni sanitari condivisi. Il teleconsulto può essere svolto in tempo reale anche con la presenza del paziente e in questo caso si fa riferimento a un approccio multidisciplinare. L'obiettivo del teleconsulto è la formulazione di una diagnosi e/o scelta della terapia.

- **Telecooperazione specialistica**: è un'attività di assistenza con cui vengono coinvolti medici e operatori sanitari. In particolare, consiste nell'attività di supporto fornito da uno specialista a un altro operatore in un atto sanitario.

Telesalute

La telesalute si riferisce a un insieme di servizi che permettono la gestione del paziente, soprattutto quelli che presentano patologie croniche. La telesalute può includere sistemi di monitoraggio dei parametri vitali del paziente. Sono sistemi che consentono di raccogliere e trasmettere continuamente dati per poter essere interpretati e analizzati da medici e specialisti.

Teleassistenza

La teleassistenza è un insieme di servizi per il supporto eseguito prevalentemente a domicilio rivolto soprattutto alle persone anziane o fragili. Consiste nell'inserimento di allarmi, di sistemi di emergenza, di dispositivi e applicazioni per la condivisione dei dati alle strutture sanitarie. La teleassistenza è un sistema che rientra più nell'ambito socio-assistenziale, ma garantisce comunque una continua assistenza sanitaria con lo scopo della prevenzione.

1.1.4 Vantaggi

La telemedicina offre varie opportunità con l'obiettivo di superare i limiti della modalità tradizionale.

- Il vantaggio maggiore che si potrà trarre dalla telemedicina consiste nell'erogazione di servizi a distanza, consentendo l'accesso a servizi qualificati e risorse mediche migliori anche per i pazienti che vivono in aree distanti dal centro urbano.

- Medici e specialisti di diversi ospedali possono consultarsi e collaborare insieme per definire una cura adatta per il paziente.

- Si possono offrire servizi anche direttamente a casa ottimizzando i tempi nello spostamento. Inoltre, si possono integrare servizi di telemonitoraggio e telecontrollo per tenere sotto controllo lo stato di salute del paziente e includere sistemi di allarme.

- La telemedicina riduce i tempi morti causati dagli spostamenti, ottimizzando i tempi e le risorse.

- Dati e informazioni possono essere condivisi e reperibili facilmente, rendendo il sistema sanitario più efficiente.

1.1.5 Considerazioni

Un aspetto da tenere in considerazione è che le informazioni trasmesse nei servizi di telemedicina sono dati rilevanti per il paziente. Pertanto, le tecnologie devono garantire l'integrità e la qualità delle informazioni durante le trasmissioni.

Inoltre, considerando che le informazioni fanno riferimento a dati sanitari e sensibili, le tecnologie devono anche rispettare le normative della sicurezza e della tutela della privacy. I servizi devono fare il meglio possibile per garantire la sicurezza dei dati, sia durante la trasmissione sia nella conservazione.

Per il trattamento di questi dati, il paziente deve fornire il consenso ed essere informato sul tipo di servizio fornito, sulle modalità e tempi di conservazione.

1.2 Wearable Computing

Con l'innovazione e lo sviluppo delle tecnologie, i dispositivi stanno diventando sempre più potenti e pervasivi, entrando a far parte della nostra vita di tutti i giorni. Dai *computer* siamo arrivati ai dispositivi mobile come gli *smartphone* e ora si stanno diffondendo dispositivi sempre più piccoli, che possono essere anche indossabili.

Per *Wearable Computing* si intende tutte le ricerche, le tecnologie, gli studi per realizzare dispositivi intelligenti che possono essere indossati. Sono dispositivi che presentano quindi una capacità computazionale con la quale possono eseguire varie applicazioni e che possono essere programmabili, proprio come gli *smartphone* e i *computer*. Si differenzia dagli ultimi soprattutto per il fatto di essere sempre in esecuzione, permettendo di visualizzare informazioni velocemente e in qualsiasi momento.

I dispositivi *wearable* sono sempre connessi tramite WiFi o Bluetooth, al fine di ricevere e trasmettere informazioni. Inoltre sono caratterizzati da nu-

merosi sensori che permettono di raccogliere dati dall'esterno e che possono essere elaborati e condivisi. Possiamo riassumere che i dispositivi *wearable* sono sempre attivi, disponibili e connessi.

Dispositivi wearable più diffusi sono gli *smartwatch* e gli *smartband*, orologi intelligenti in grado di mostrare informazioni su un piccolo schermo, indossato nel polso. Altri dispositivi indossabili diffusi sono i visori e gli smartglasses, dispositivi che appartengono alla sottocategoria *Eyewear Computing*.

1.2.1 Eyewear Computing

Una interessante sottocategoria di *wearable computing* è l'*eyewear computing*. Comprende tutti i dispositivi indossabili che interagiscono con il senso della vista dell'utente. Sono dispositivi in grado di proiettare nel display informazioni o oggetti direttamente davanti agli occhi creando "un'estensione di sé stessi" [23].

I principali componenti dei dispositivi eyewear sono: [3]

- **Display**, il display può essere differenziato in display totalmente occlusivi e see-through. I primi sono display che oscurano completamente la parte dietro dello schermo, mentre i secondi sono display semi-trasparenti che consentono di vedere il mondo reale e di aggiungere contenuti virtuali.

- **Metodi di input**, essendo dispositivi indossabili, l'interazione si differenzia moltissimo dai computer o dagli smartphone. I metodi di interazione possono cambiare da modello a modello. In alcuni smartglasses sono stati integrati dei pulsanti o touchpad situati nell'asta degli occhiali che permettono di eseguire semplici comandi. Altri possono essere controllati attraverso un apposito controllore esterno o dallo smartphone, connesso via Bluetooth o WiFi.

  I metodi di input possono inoltre essere *touchless*, ovvero consentono l'interazione senza tocco. Tale categoria si suddivide in interazioni *hands-free* e *freehand*. I primi non richiedono l'interazione delle mani e possono essere realizzati con sistemi di riconoscimento vocale, tracciamento del movimento degli occhi o della testa. Mentre per i secondi, possono essere realizzati dei comandi gestuali per il controllo del dispositivo. [17]

- **Sensori**, come la maggior parte dei dispositivi, includono tantissimi sensori quali: accelerometro, giroscopio, sensore di prossimità, sensore di luminosità, gps, ecc. Alcuni dispositivi più avanzati presentano sensori per il tracciamento del movimento degli occhi e della testa.

- Altri componenti tecnici come l'unità di elaborazione, batteria e memoria.

(a) Google Glass Enterprise Edition 2 (b) Vuzix Blade

Figura 1.1: Esempi di dispositivi smartglasses

Inoltre, possono essere progettate per supportare diverse tecnologie:

- **Augmented reality**, in questo caso i contenuti digitali sono mostrati sempre al primo piano, in un livello separato e sopra al mondo fisico.
- **Mixed Reality**, gli oggetti virtuali, o meglio gli ologrammi, si integrano al mondo fisico, adattandosi allo spazio e ad altri oggetti reali, consentendo all'utente di manipolarli.
- **Virtual Reality**, rappresenta un mondo totalmente virtuale isolato dal mondo reale.

Dispositivi eyewear più noti e utilizzati sono gli smartglasses e i visori Head Mounted Display.

Smartglasses

Gli *smartglasses* sono esempi di dispositivi *eyewear*. Sono degli occhiali intelligenti per *Augmented reality*, in grado di fornire contenuti e informazioni direttamente davanti agli occhi dell'utente.

Sono dispositivi efficaci, comodi da indossare e di dimensioni leggermente più grandi rispetto agli occhiali tradizionali. Consentono di svolgere varie funzionalità: scattare foto, registrare video, ricevere e inviare messaggi, navigare su Internet, memorizzare e trasmettere dati, ecc.

Il principio di funzionamento [19] consiste nel proiettare contenuti virtuali sulle lenti riflettenti, che poi vengono rifratti fino a proiettarli davanti agli occhi dell'utente. Spesso si usano delle lenti convesse che consentono all'utente di visualizzare i contenuti ingranditi e più distanti.

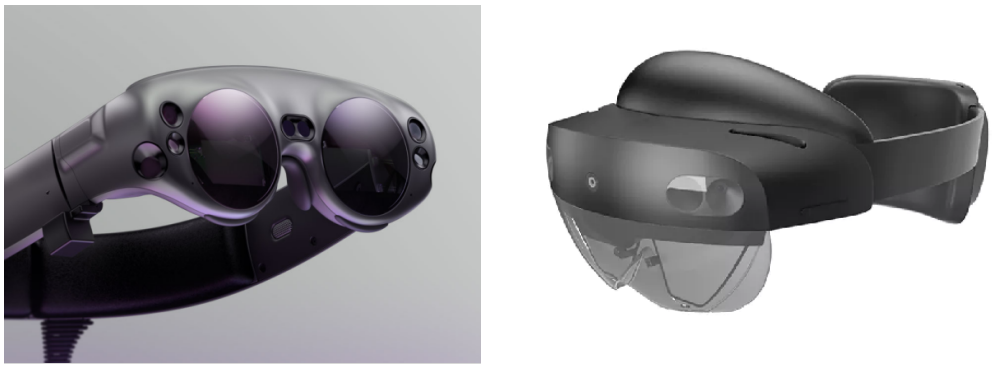
Lo schermo riflettente può essere inserito direttamente nelle lenti degli smartglasses, come nel caso dei Vuzix Blade (Figura 1.1b), oppure su un componente esterno e separato come nel caso dei Google Glass (Figura 1.1a).

Head Mounted Display

Gli Head Mounted Display sono dei visori indossabili sulla testa, sono più potenti poiché contengono un numero maggiore di sensori rispetto agli smartglasses e di conseguenza sono più costosi e più ingombranti. Si possono perciò realizzare applicazioni più complesse con tecnologie non solo *Augmented reality*, ma anche *Mixed Reality*.

Gli oggetti virtuali nella *Mixed Reality* si adattano allo spazio circostante, rispettando la prospettiva e a seconda della posizione in cui si trovano, alcune parti possono essere non visibili, nascoste dagli altri oggetti. Gli ologrammi sono quindi delle immagini tridimensionali che possono essere manipolati, spostati e ridimensionati dall'utente, con un controllore apposito oppure direttamente con comandi gestuali.

Sul mercato sono disponibili diversi dispositivi *Head Mounted Display*, esempi diffusi sono i Hololens di Microsoft e Magic Leap raffigurati nella Figura 1.2.



(a) Magic Leap (b) Microsoft Hololens2

Figura 1.2: Esempi di dispositivi Head Mounted Display

1.2.2 Utilizzo in ambito sanitario

I dispositivi wearable sono ampiamente diffusi nel settore sanitario, favorendo una maggiore precisione ed efficienza nell'esecuzione delle attività. Gli smartglasses, per esempio, danno la possibilità di essere utilizzati comodamente in qualsiasi momento, lasciando le mani completamente libere. Medici

e operatori sanitari possono quindi usare entrambe le mani per svolgere le loro mansioni e allo stesso tempo ottenere informazioni precise.

Si pensi per esempio alle attività che richiedono massima concentrazione come un medico durante l'esecuzione di un intervento chirurgico, la fase di soccorso oppure la formulazione di una diagnosi del paziente, ecc. In tutti questi contesti e molti altri, si potrebbero affiancare degli smartglasses con applicazioni utili ai medici quali: registrazione di immagini e video per la documentazione, attivazione di servizi di teleconsulto o telecooperazione con professionisti di altri ospedali, analisi di dati e parametri del paziente, oppure per poter visualizzare le istruzioni o una serie di passaggi per guidare il medico.

Dall'altro lato i dispositivi wearable possono fornire supporto anche ai pazienti attraverso servizi di telemonitoraggio e telecontrollo. Possono includere anche funzionalità di memorizzazione, analisi e condivisione dei dati, con l'obiettivo di prevenire o evitare il peggioramento delle malattie.

Capitolo 2

Il Progetto TeleStroke

L'obiettivo di questo capitolo è di introdurre il progetto di studio e di analisi dell'elaborato.

In particolare, "TeleStroke" è un progetto seguito dal gruppo di ricerca del DISI - Dipartimento di Informatica - Scienza e Ingegneria dell'Università di Bologna, campus di Cesena, e svolto in collaborazione con l'Ospedale Bufalini di Cesena, coinvolgendo l'Unità Operativa di Neurologia e AUSL Romagna.

Si tratta di un progetto che si colloca nell'ambito della telemedicina e in particolare, è un sistema di teleconsulto e di telecooperazione con lo scopo di valutare la gravità dell'ictus cerebrale. Di seguito verrà introdotto il progetto, descrivendo il contesto di applicazione, il progetto esistente e per finire gli obiettivi della tesi.

2.1 Contesto

L'AUSL Romagna adotta da molti anni un modello organizzativo *Hub and Spoke*, come spiegato nella sezione 1.1.2. L'ospedale Bufalini di Cesena è considerato il Centro Hub della Romagna per il trattamento dell'ictus, dove presenta professionisti e risorse attrezzate e fornisce supporto agli ospedali Spoke di Forlì, Rimini e Ravenna.

L'ictus è una patologia tempo-dipendente, per cui è molto importante minimizzare il tempo di presa in carico del paziente e velocizzare l'attività di diagnosi. Tuttavia, non sempre negli ospedali Spoke sono disponibili neurologi e professionisti specializzati per gestire i casi dell'ictus, soprattutto nelle ore di reperibilità notturna e nel fine settimana, a causa delle risorse non uniformi presenti nel territorio.

Per questo risulta particolarmente utile applicare i servizi della Telemedicina, infatti attraverso il teleconsulto, operatori sanitari degli ospedali Spoke potranno mettersi in contatto con specialisti del centro Hub, collaborando

insieme per fornire tempestivamente una diagnosi e una cura adatta per il paziente colpito dall'ictus. Così nasce il progetto "TeleStroke", un sistema progettato per fornire supporto agli specialisti durante la gestione del paziente affetto dall'ictus.

2.2 Introduzione al progetto

Il progetto "TeleStroke" si pone come obiettivo di fornire supporto agli specialisti e ai operatori durante la fase di diagnosi, con lo scopo di velocizzare il processo e di minimizzare gli errori.

Il sistema fornisce il servizio di teleconsulto a distanza, attivando una videoconferenza tra l'operatore sanitario e il neurologo. Allo stesso tempo il sistema mostra una serie di operazioni da far compiere al paziente, seguendo una scala predefinita.

Il neurologo registra nel sistema le informazioni del paziente e il comportamento espresso nell'eseguire i compiti indicati, e una volta raccolto tutti i dati, la scala restituisce un punteggio che indica la gravità dell'ictus.

Il sistema è stato generalizzato per poter essere applicato ad altri scenari con una modalità simile. Per cui può includere vari *template* preimpostati, che possono essere scelti dallo specialista all'attivazione di una sessione.

Nello specifico, un *template* è composto da una serie di fasi di compilazione, chiamate *checklist*. Una *checklist* può essere associata a uno o più risultati e raccoglie una lista ordinata di *step*. Ogni *step*, composto da un titolo e una descrizione, rappresenta un punto, un campo da compilare della *checklist*.

Nel progetto è già stato inserito un *template* per il trattamento dell'ictus. Esso si compone di una *checklist* per la raccolta delle informazioni del paziente e un'altra per quantificare la severità dell'ictus, basandosi sulla scala di valutazione NIH Stroke Scale[1], in breve NIHSS. La scala NIHSS è uno strumento utilizzato per valutare la gravità dell'ictus che, basandosi sulle risposte date, fornisce un risultato finale che ne indica la gravità. Infine, il neurologo analizzando il risultato ottenuto dalla valutazione, potrà suggerire un iter da intraprendere per la cura del paziente.

Il progetto è composto principalmente da tre macro-componenti: il backend, il frontend dell'operatore sanitario e il frontend dello specialista.

2.2.1 Backend

Il Backend si riferisce a tutta la logica di funzionamento nascosta dietro alle interfacce, si occupa dello scambio dei dati tra i due frontend e la gestione

[1]National Institute of Health Stroke Scale

delle informazioni, accedendo se necessario al database.

Si compone da vari *microservizi* indipendenti che collaborano tra loro per rispondere alle richieste dei frontend. I Frontend possono interagire con il Backend attraverso chiamate HTTP alle API esposte.

2.2.2 Frontend dell'operatore

L'applicazione frontend dell'operatore sanitario è stata progettata per essere eseguita sugli smartglasses.

Nello specifico, è stato selezionato il modello di smartglasses Vuzix Blade[2], in quanto risulta particolarmente efficace per questo contesto. Sono dispositivi facili e comodi da indossare, infatti permettono di inquadrare sempre il paziente registrando flussi audio e video, in modo che il neurologo da remoto possa vedere le reazioni del paziente. Inoltre permettono di visualizzare informazioni proiettate nella lente e presentano dimensioni simili a dei comuni occhiali da vista, perciò possono essere portati facilmente senza impedimento, lasciando le mani dell'operatore completamente libere per aiutare il paziente.

Il sistema già esistente, presenta due principali schermate, descritte in seguito.

Schermata di inizializzazione

Una prima schermata, come illustrato nella Figura 2.1, rappresenta la fase di inizializzazione e connessione al server, si caratterizza da una *progress bar* con uno stato indeterminato e un testo per rappresentare lo stato della configurazione. Una volta concluse le fasi preliminari di configurazione, l'applicazione si mette in attesa di ricevere chiamate.

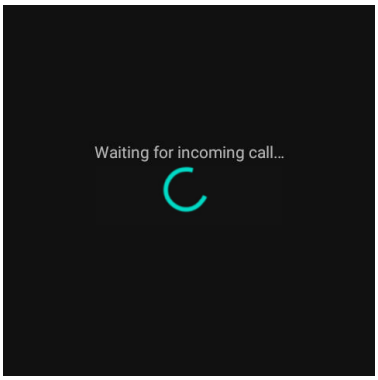


Figura 2.1: Stato di in attesa di ricevere chiamate

[2]Vuzix Blade

Schermata durante la chiamata

Una seconda schermata, come illustrato nella Figura 2.2, viene avviata una volta stabilita la connessione con lo specialista, si occupa principalmente di visualizzare gli *step* che si devono chiedere o far eseguire al paziente. Gli step sono trasmessi ogni volta che lo specialista termina la compilazione dello step precedente e sono caratterizzati da un titolo e una descrizione che riassume il compito da eseguire.

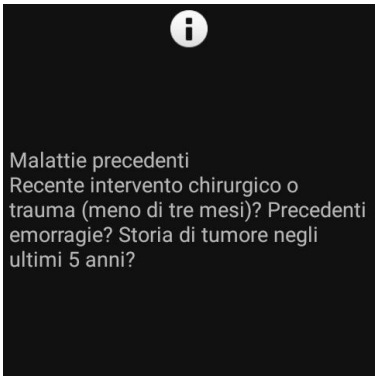


Figura 2.2: Esempio di un step

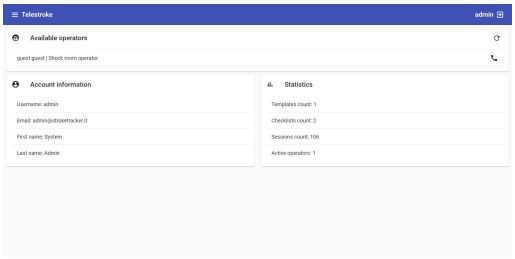
2.2.3 Frontend dello specialista

Il frontend dello specialista è sviluppato su una web app, ed è pensato proprio per poter essere eseguito su vari dispositivi. Infatti è possibile che lo specialista si trovi in una postazione fissa e sia in grado di utilizzare un *computer*, oppure se si è in movimento allora avrà la possibilità di utilizzare per esempio un *tablet*. La web app consente: di iniziare una nuova sessione con l'operatore, di comunicare e visualizzare i flussi video in tempo reale e di compilare gli *step* per la valutazione della gravità dell'ictus del paziente.

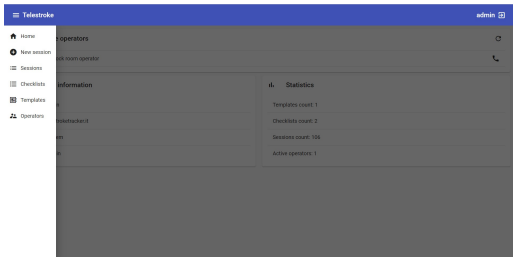
Dopo aver inserito le credenziali per l'autenticazione al sistema, si può accedere alla schermata *Home*, illustrato nella Figura 2.3a, con la quale lo specialista può avviare facilmente una chiamata con gli operatori disponibili. Tale schermata permette inoltre di visualizzare le informazioni dell'utente e una statistica generale sulle sessioni svolte.

Nelle interfacce è presente una *sidenav*, dove è possibile accedere alle pagine riguardanti le sessioni, le checklist, i template e gli operatori disponibili. Una volta attivata la chiamata con l'operatore, ci si ritrova nella pagina della sessione, indicata nella Figura 2.3c, dove lo specialista potrà visualizzare il flusso di video dell'operatore a destra, e compilare le *checklist* del *template* selezionato nella parte a sinistra.

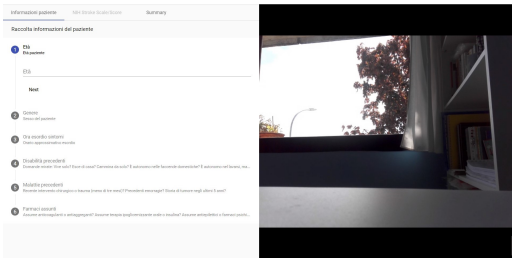
La pagina della sessione è stata realizzata utilizzando una navigazione a *tab*, dove per ciascun *tab* è associato a una fase di compilazione (*checklist*), nell'ultimo tab, invece, si può visualizzare il riepilogo della sessione, con informazioni sul tempo e sui progressi delle *checklist* come indicato nella Figura 2.3d.



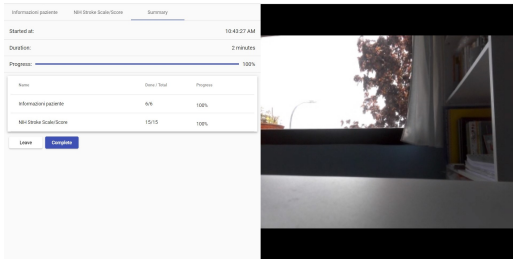
(a) Schermata home



(b) Sidenav



(c) Schermata sessione



(d) Schermata riepilogo

Figura 2.3: Le schermate principali del Frontend dello specialista

2.3 Obiettivo della tesi

Il progetto "TeleStroke" viene affiancato al lavoro degli operatori sanitari e dei medici per la gestione dei pazienti affetti da ictus, al fine di minimizzare i tempi della diagnosi, nonché la presa in carico ospedaliera.

Perciò, considerando il particolare contesto di collocamento del sistema, è molto importante che l'applicazione supporti il lavoro dei professionisti fornendo informazioni utili e necessari. Allo stesso tempo, l'applicazione deve facilitare l'utilizzo e l'apprendimento, senza creare ulteriori difficoltà e perdite di tempo durante l'uso.

Per cui l'obiettivo generale della tesi si concentra sul refactoring del progetto, aggiungendo funzionalità che facilitando l'utilizzo da parte dei professionisti e allo stesso tempo cercando di aumentare l'usabilità.

Dopo aver introdotto il sistema già esistente, possiamo notare che il frontend dello specialista, ovvero l'applicazione web, presenta schermate e funzio-

nalità piuttosto complete, con dei piccoli aggiustamenti che si possono fare per migliorare l'usabilità.

Mentre per quanto riguarda l'applicazione dell'operatore sanitario, risulta ancora un prototipo piuttosto minimale: le schermate possono essere migliorate e si potrebbero includere nuove informazioni e funzionalità che supportano il lavoro del personale.

Pertanto, la tesi si concentra soprattutto sul refactoring del frontend dell'operatore sanitario, ovvero l'applicazione eseguita sugli *smartglasses* Vuzix Blade, con alcune modifiche apportate nel frontend dello specialista per migliorare l'usabilità.

2.4 Requisiti ed Analisi

Di seguito verranno illustrate i vari requisiti che sono state inizialmente definite per portare un miglioramento del sistema. Verranno prima descritti i requisiti funzionali, e in seguito quelli non funzionali.

2.4.1 Requisiti funzionali

I requisiti funzionali rappresentano l'effettivo comportamento e le funzionalità che il sistema deve possedere.

Frontend dell'operatore Il frontend dell'operatore risulta piuttosto minimale, per cui sono state definite una serie di funzionalità che potrebbero essere utili durante la fase di gestione del paziente.

- Consentire all'operatore di poter avviare la chiamata manualmente, invece di attivare la chiamata autonomamente con l'avvio dell'applicazione.
- L'applicazione deve poter visualizzare una serie di stati del dispositivo, come il livello della batteria degli *smartglasses*, la presenza di una connessione ad Internet, dispositivo di uscita dell'audio, ecc. permettendo all'operatore di osservarli in qualsiasi momento. Il sistema deve inoltre avvisare se qualche parametro non si trovi nello stato di normalità.
- Poter visualizzare in alcune fasi della sessione, il flusso video dello specialista, invece di sentire solo l'audio, al fine di ottenere un'interazione più naturale.
- Una volta iniziata la chiamata, sarebbe utile poter visualizzare un cronometro, permettendo all'operatore sanitario di avere un'idea del tempo trascorso durante la sessione.

- Durante la visualizzazione degli step, sarebbe utile visualizzare una barra di avanzamento degli step compilati, in questo modo l'operatore sanitario avrà una visione generale dell'andamento della compilazione.

Frontend dello specialista Per quanto riguarda invece, l'interfaccia dello specialista, anche se è abbastanza curato e completo, sono stati definiti alcuni componenti che possono essere aggiunti:

- Durante l'inizio della sessione, sarebbe utile poter visualizzare le informazioni della sessione e avere un pulsante che consente di avviare la compilazione dei dati.
- Permettere allo specialista di visualizzare il risultato finale per ogni fase di compilazione nella pagina di riepilogo e di poter inserire delle annotazioni finali per la diagnosi del paziente.

2.4.2 Requisiti non funzionali

I requisiti non funzionali individuano i vincoli e le proprietà non comportamentali del sistema che devono essere soddisfatti.

In particolare, il sistema deve soddisfare l'**usabilità**, infatti dato il delicato ambito di utilizzo, l'applicazione deve essere facile e veloce da utilizzare, senza creare ulteriori difficoltà e supportando i medici nelle operazioni di gestione del paziente. Per cui, aspetti della grafica come colori, forme, tipografica, devono essere facilmente visibili anche in ambienti e situazioni diversi, soprattutto per il frontend dell'operatore essendo eseguita sugli *smartglasses* con uno schermo trasparente. Eventuali pulsanti devono essere accessibili e facili da selezionare.

Un altro aspetto molto importante è che l'applicazione, e in particolare l'interfaccia della *web-app*, deve poter adattarsi su vari dispositivi con schermi di differenti dimensioni. Pertanto le interfacce devono essere **responsive**, ovvero devono adattarsi ai vari dimensioni e risoluzioni dello schermo garantendo la visibilità di tutti gli elementi e se è necessario, rendendo l'interfaccia scorrevole.

Capitolo 3

Guidelines per la progettazione di applicazioni wearable

Considerando il delicato ambito di applicazione del progetto e il particolare dispositivo *wearable* utilizzato per eseguire l'applicazione dell'operatore, si emerge la necessità di approfondire sul "come" progettare applicazioni per i dispositivi wearable e soprattutto applicazioni per *smartglasses*.

Queste applicazioni devono essere progettate tenendo in considerazione le diverse configurazioni che ci sono tra i dispositivi *wearable* e i *computer* o gli *smartphone*. Infatti si differenziano per esempio dal diverso tipo e dimensione del display, dai diversi metodi di interazione, dalla limitata capacità computazionale, ecc. Sono tutti aspetti che possono influenzare il design delle applicazioni.

Inoltre, un requisito importante da soddisfare per lo sviluppo di applicazioni è l'usabilità. L'usabilità pone al centro l'utente, sviluppando così, applicazioni che rispecchiano le sue necessità ed esigenze.

In questo capitolo verranno discussi i principi di progettazione per applicazioni per i dispositivi *wearable*, con una particolare attenzione sugli smartglasses Vuzix Blade, il dispositivo scelto per eseguire l'applicazione dell'operatore sanitario.

Il capitolo inizierà introducendo il concetto di usabilità con una descrizione sui parametri e criteri che devono essere seguiti durante la progettazione. Successivamente verranno discussi i principali aspetti da prestare attenzione per la progettazione di applicazioni usabili per smartglasses basandosi soprattutto sulle linee guida fornite da Vuzix e facendo anche un confronto con le indicazioni fornite da altre case produttrici di questi occhiali.

3.1 Usabilità

L'ISO (International Organization for Standardization) definisce l'usabilità come:

> "il grado in cui un prodotto può essere usato da particolari utenti per raggiungere certi obiettivi con efficacia, efficienza, soddisfazione in uno specifico contesto d'uso" [21]

L'usabilità studia il modo con cui le operazioni possono essere completati facilmente, velocemente e con soddisfazione dall'utente. Le tecniche dell'usabilità si basano sul *user-centered design*, ponendo l'utente sempre al centro dell'attenzione.

Ha come parametri di misura l'efficacia, l'efficienza e la soddisfazione. In particolare: [24]

- **L'efficacia** studia come gli utenti possono raggiungere l'obiettivo in modo completo e preciso, misurando il numero di passi richiesti per completare le operazioni.
- **L'efficienza** misura la quantità di risorse impegnate nell'eseguire le operazioni richieste.
- **La soddisfazione** è un aspetto soggettivo che misura l'accettabilità delle funzionalità da parte dell'utente.

Il concetto nasce nell'ambito dell'ergonomia, ma successivamente con la diffusione delle tecnologie, viene applicato anche nei prodotti informatici. Pone una maggiore attenzione nello sviluppo delle interfacce grafiche, siccome è il principale componente con il quale l'utente interagisce. In effetti, all'utente non interessa l'implementazione che sta sotto all'applicazione, ma ciò che viene mostrato nell'interfaccia e i risultati ottenuti dopo un'azione.

Di seguito vengono illustrati i criteri dell'usabilità da rispettare durante la progettazione delle applicazioni. [24, 25, 10]

- **Apprendibilità**. L'applicazione deve essere veloce da apprendere anche per gli utenti che lo utilizzano per la prima volta. Si deve quindi creare interfacce semplici, includendo, se necessario, istruzioni per guidare l'utente a svolgere determinate azioni.
- **Velocità**. Gli utenti devono poter raggiungere i loro obiettivi rapidamente, seguendo il percorso più semplice.
- **Soddisfazione**. Gli utenti devono trovare l'applicazione piacevole da utilizzare.

- **Facilità di navigazione**. L'applicazione deve essere facile da utilizzare e navigare.
- **Memorabilità**. L'utente deve poter riuscire a utilizzare l'applicazione anche dopo un periodo di inattività.
- **Prevenzione degli errori**. L'applicazione deve ridurre la percentuale di errori incorreggibili. Se si verificano degli errori durante lo svolgimento di determinate azioni, occorre segnalare all'utente la causa e i passaggi da seguire per correggerli.

3.2 Linee guida per il design di applicazioni per smartglasses

I dispositivi *eyewear* possono essere indossabili durante tutto l'arco della giornata, si rendono disponibili quando l'utente ne ha bisogno e possono funzionare allo stesso modo, anche quando l'utente non ne ha più bisogno. Per cui le applicazioni devono essere progettate per non distrarre l'utente dalle attività che sta svolgendo, a differenza degli *smartphone* che sono pensate proprio per catturare l'attenzione dell'utente.

Pertanto, le interfacce grafiche devono essere chiare e semplici contenendo informazioni accurate e utili all'utente, evitando di inserire elementi decorativi o non appropriati per il contesto dell'applicazione.

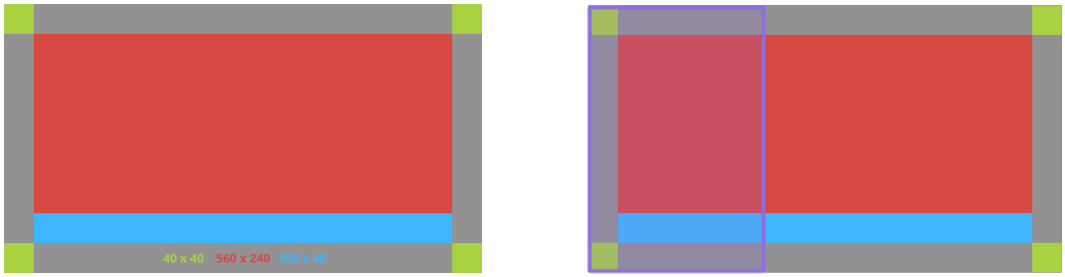
La maggior parte degli *smartglasses* si basa su un sistema operativo *Android*, per cui è possibile far eseguire applicazioni *mobile*. Per lo sviluppo di applicazioni per *smartglasses*, molti aspetti si riferiscono ancora sui concetti e principi dei dispositivi *mobile*.

Tuttavia, ad oggi molte case produttrici come Google o Vuzix, mettono a disposizione delle linee guida per il design di applicazioni dedicato esclusivamente per questi tipi di dispositivi.

Successivamente vengono riportati i principi di design più significative, con particolare riferimento alle linee guida messe a disposizione per Vuzix Blade [16] e confrontandosi con le indicazioni fornite da Google per gli smartglasses Glass [12, 13].

3.2.1 Layout

Una disposizione degli elementi ben studiata permette di ottenere un'interfaccia chiara e ordinata, rendendo la navigazione piacevole e aiuta l'utente a orientarsi meglio all'interno dell'applicazione. Considerando che lo spazio a disposizione per le applicazioni è limitato, solitamente si preferiscono layout



(a) Main template (b) Standard template

Figura 3.1: Esempi di layout forniti da Google

semplici e minimali: troppe informazioni in una schermata creano solo confusione all'utente. Infatti, l'attenzione dell'utente è limitata, perciò è meglio dedicarla ai contenuti più importanti, cercando di eliminare elementi di decorazione o non necessarie. Le varie case produttrici mettono a disposizione dei propri template per il posizionamento degli elementi.

Google Glass colloca le schermate in ordine temporale, dove a ogni schermata viene associata una *card* ed è possibile scorrere verso destra e verso sinistra per vedere le *card* passate e future. Esempi di impaginazione delle *card* si possono trovare nella Figura 3.1. In particolare, la prima figura (Figura 3.1a) mostra una disposizione consigliata per la schermata principale, in cui il contenuto più importate viene inserito nella zona centrale colorata di rosso. Mentre la seconda figura (Figura 3.1b) predispone uno spazio laterale per posizionare altri contenuti come varie informazioni o immagini.

Anche l'azienda Vuzix fornisce una serie di linee guida per la disposizione degli elementi. Propone, come Google, di utilizzare un template *center-content*, in cui le informazioni vengono mostrate al centro della schermata delineato con un margine minimo per sfruttare al massimo lo spazio a disposizione (Figura 3.2a). Si possono inoltre, inserire dei menu per mostrare le operazioni disponibili all'utente. Mentre per i contenuti più importanti viene consigliato, sempre di posizionarli al centro della schermata, ma lasciando opportuni spazi vuoti attorno, come visibile nella Figura 3.2b.

3.2.2 Menu

Il menu è un componente che permette di raggruppare una serie di elementi o azioni disponibili per l'utente.

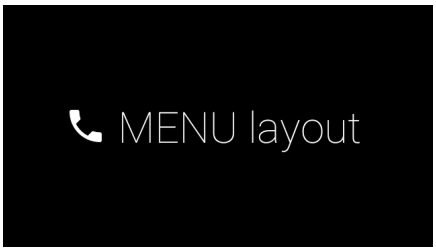
Il menu principale degli smartglasses, in generale è rappresentato da una lista di applicazioni con cui è possibile scorrere e attivare un'applicazione interagendo con il touchpad oppure attraverso comandi vocali.

Nella Figura 3.3 viene illustrato degli esempi di menu delle applicazioni.

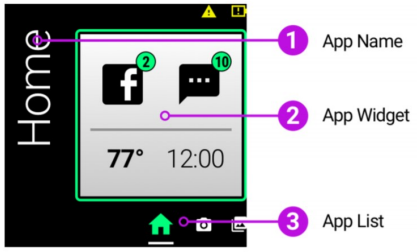
(a) Center content layout con un margine piccolo e un menu

(b) Center content layout per evidenziare contenuti importanti

Figura 3.2: Esempi di layout forniti da Vuzix



(a) Menu di Google Glass



(b) Apps Menu di Vuzix Blade

Figura 3.3: Esempi di Menu delle applicazioni

Action Bar

L'*Action Bar*, messo a disposizione da Vuzix, è un menu di navigazione che fornisce informazioni o azioni chiare e concise per l'utente. È composto da una lista di pulsanti progettate per essere compresi facilmente, e che possono essere situate nella parte centrale o inferiore della schermata, esempi sono illustrati nella Figura 3.4.

Le *Action Bar* forniscono all'utente una serie di azioni disponibili che possono essere selezionate con i vari metodi di input. Si può subito notare che sono caratterizzate da colori facilmente visibili, l'elemento selezionato è il più grande, per cui quando l'utente scorre la lista, vengono spostati tutti gli elementi lasciando sempre l'elemento selezionato al centro, offrendo una migliore esperienza utente.

Si suggerisce di inserire il pulsante per tornare indietro come prima opzione, in modo che l'utente possa tornare indietro di una schermata, a meno che un pulsante non includa già l'azione indietro, come nel caso della cancellazione di

un elemento. Vuzix fornisce delle librerie che permettono facilmente di creare le *Action Bar*, lasciando ampio spazio alla personalizzazione.

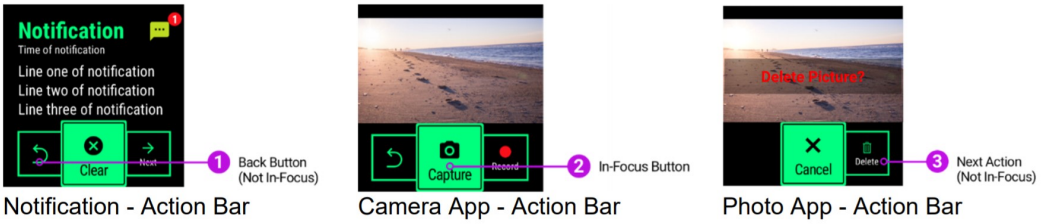


Figura 3.4: Indicazioni per il design dell'Action Bar

3.2.3 Colori e temi

La scelta giusta dei colori è un aspetto molto importante per creare applicazioni usabili.

Il colore di sfondo di default per gli *smartglasses* è nero, un colore privo di luce che non viene renderizzato dai proiettori, permettendo quindi di vedere il mondo reale dietro all'interfaccia e sviluppare applicazioni per *Augmented reality*. Pertanto, la visualizzazione di alcuni contenuti o testi potrebbe non essere facilmente visibile in certi ambienti, soprattutto per zone all'aperto molto luminose.

In generale si raccomanda di selezionare dei colori contrastanti e saturi, per esempio, Vuzix consiglia la palette di colori illustrato nella Figura 3.5: i colori primari sono il verde e il bianco, adatti per la visualizzazione di testo o elementi grafici, mentre i colori secondari, giallo e rosso sono riservati soprattutto per allarmi e avvisi. Per le comunicazioni importanti si consiglia di non rappresentarle solo tramite dei colori, ma di abbinarle con icone o etichette in modo da rendere accessibili anche per le persone con una diversa percezione del colore.

Se l'applicazione è pensata principalmente per un uso esterno, allora si consiglia di includere anche un tema chiaro, caratterizzato da uno sfondo di colore bianco e testi di colore scuro.

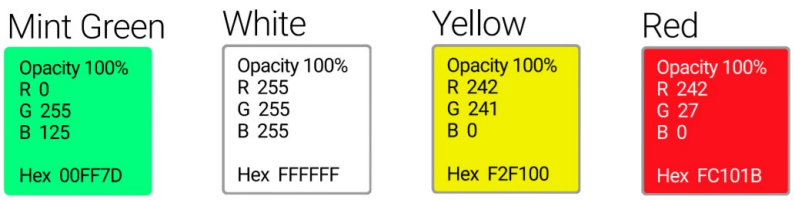


Figura 3.5: Palette di colori consigliato da Vuzix

3.2.4 Tipografia

Un altro aspetto da prendere in considerazione per migliorare l'usabilità è lo stile e la dimensione del testo.

Testi troppo lunghi non sono adatti per essere letti sugli smartglasses, si consiglia sempre di inserire testi brevi, concisi e facili da comprendere. Considerando che l'utente potrebbe essere in movimento durante l'uso dell'applicazione, le dimensioni del testo devono essere appropriate, da poter visualizzare senza sforzare troppo l'occhio dell'utente.

Per quanto riguarda il font, tipicamente si consiglia di scegliere una famiglia di font abbastanza leggibile e accessibile, un esempio potrebbe essere la famiglia di font *Roboto*[1] di Google, consigliato anche da *Material Design*.

Come si può vedere nella Figura 3.6, la famiglia di font può assumere vari stili e forme, adattandosi in tutti i contesti.

In una schermata si devono evitare di inserire troppe tipologie di font e stili, questi rendono la pagina più disordinata e disorientano l'utente.

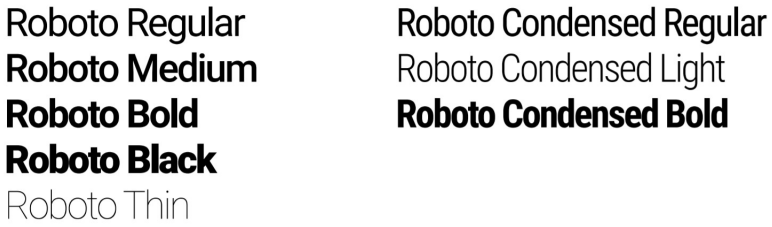


Figura 3.6: Famiglia di font Roboto

[1]Roboto - Google Fonts

Capitolo 4

Progettazione e Sviluppo

A seguito delle prime fasi di studio e approfondimento dei concetti che stanno dietro al progetto, è possibile procedere alla parte di progettazione e sviluppo. In particolare, in questo capitolo verranno descritte le varie funzionalità e componenti che sono state aggiunte per potenziare e migliorare il sistema.

Il capitolo inizierà con la fase di progettazione, seguirà con una panoramica sulle tecnologie utilizzate, per poi procedere alla fase di sviluppo e implementazione.

4.1 Progettazione

In questa sezione verranno approfondite le scelte progettuali e di design dei vari componenti che hanno portato a un miglioramento dei Frontend, sia dal lato dell'operatore e sia dal lato dello specialista.

4.1.1 Frontend dell'operatore

Il Frontend dell'operatore sanitario, essendo eseguita sugli *smartglasses*, deve essere progettata in modo da rendere l'applicazione intuitiva e facile da utilizzare seguendo le guidelines approfondite nel capitolo 3. Inoltre deve rispettare i vari requisiti e funzionalità definiti nella fase di analisi.

Come primo passo della fase di progettazione sono stati realizzati vari *mockup*, delle rappresentazioni fittizie che permettono di visualizzare l'effetto finale dell'applicazione. Nello specifico, prima dell'implementazione è stata realizzata una versione iniziale dei mockup che poi sono state integrate con alcune idee e miglioramenti durante la fase di sviluppo.

Figura 4.1: Mockup della schermata iniziale

Schermata iniziale

Il mockup della schermata iniziale, rappresentato nella Figura 4.1, comprende oltre al titolo del progetto, anche un menu con le varie opzioni disponibili per l'utente.

Gli elementi del menu sono stati ingranditi per facilitare la visualizzazione sugli schermi piccoli degli *smartglasses* e per facilitare la selezione da parte dell'utente. Il menu contiene le seguenti opzioni:

- **Back**. Opzione per tornare indietro, e quindi consente l'uscita dall'applicazione.
- **New Call**. Opzione di default che consente di avviare una nuova chiamata, in questo caso viene notificato al backend la disponibilità dell'operatore. Il Backend si occuperà di inserirlo nella lista degli operatori sottoscritti, permettendo così allo specialista di poter selezionarlo e iniziare la chiamata.
- **Settings**. Considerando che al momento tutte le configurazioni vengono già impostati durante l'installazione dell'applicazione, è stata predisposta l'opzione "Impostazioni" per poter visualizzare le configurazioni attuali.
- **Change Settings**. Opzione per modificare le impostazioni attraverso la scansione di un *qr-code*.

Rappresentazione degli stati

Per quanto riguarda gli stati dell'applicazione, si è deciso di rappresentarli con delle icone, in modo che l'utente possa osservare velocemente la situazione del dispositivo.

Come mostrato nella Figura 4.2, sono stati definiti i seguenti stati:

- **Connessione alla rete**, in quanto per stabilire una comunicazione audio/video con lo specialista, è necessario una connessione a Internet.

- **Stato e livello della batteria**, in quanto la trasmissione di flussi video comporta un consumo di batteria più elevato e quindi con un livello troppo basso, la chiamata potrebbe interrompersi velocemente.

- **Presenza di altoparlante o cuffie**, senza la quale l'operatore non potrà sentire quello che gli viene riferito dallo specialista, questo stato è stato inserito perché nel modello di *smartglasses* scelto, non è presente un altoparlante interno.

- **Stato della connessione con lo specialista**, stato che viene mostrato solo quando è stata stabilita correttamente la connessione con lo specialista.

Figura 4.2: Stati dell'applicazione

Visualizzazione step

Durante la sessione di chiamata con lo specialista, l'applicazione mobile mostrerà gli *step* che serviranno a guidare l'operatore sanitario nella gestione del paziente.

Inizialmente si era pensato, oltre a visualizzare una breve descrizione dello step, di selezionare delle icone che rappresentino le operazioni, in modo che l'operatore possa visualizzare immediatamente l'azione da svolgere, ma poi non è stato preso in considerazione per i seguenti motivi:

- Potrebbe distrarre l'operatore sanitario.

- Risulta difficile trovare una icona adatta per ogni step che sia in grado di rappresentarlo efficacemente.

- Per rendere l'applicazione estendibile anche per altri *template*, si dovrebbe associare a ogni step la sua icona e memorizzarla nel database. Per cui ogni volta che si completa uno step, si deve trasmettere, oltre all'informazione dello step, anche la sua icona. Ciò comporta un consumo di risorse più elevato.

Figura 4.3: Mockup della schermata di visualizzazione degli step

Pertanto, si è deciso di far visualizzare solo il titolo dello step che comunque contiene una descrizione abbastanza concisa.

Come raffigurato nella Figura 4.3, l'informazione dello step si trova al centro della schermata, seguendo il layout *center-content*. Nella parte superiore, oltre alla lista degli stati precedentemente definito, è stato aggiunto anche un cronometro che rappresenta il tempo trascorso dall'inizio della sessione.

Mentre nella parte inferiore, si può trovare una *progress bar* che rappresenta lo stato di avanzamento della compilazione degli step. Questa può risultare utile, per esempio nelle situazioni in cui il paziente non si trova in condizioni ottimali per svolgere le operazioni richieste, e a questo punto l'operatore può valutare a seconda di quanti step sono rimasti, se accelerare il processo oppure interrompere la sessione.

Visualizzazione video

Durante tutta la sessione di chiamata, l'applicazione permette di riprodurre flussi audio dello specialista; e solo in alcuni parti della chiamata si è pensato di far visualizzare anche il flusso video dello specialista, al fine di ottenere una interazione più naturale.

Nella Figura 4.4 è raffigurato il mockup per la schermata della sessione con il flusso video dello specialista, assieme a un'informazione sul suo nome e cognome. Anche in questa schermata sarà incluso un'indicazione del tempo e gli stati dell'applicazione nella barra superiore.

Il flusso video non viene mostrato in tutta la sessione perché comporta un elevato consumo di risorse e di energia, oltre a occupare molto spazio nel display, rendendo difficile la collocazione delle informazioni degli step.

In primo luogo, il flusso video viene visualizzato una volta iniziata la comunicazione con lo specialista e prima dell'inizio della compilazione degli step.

Figura 4.4: Mockup della schermata di visualizzazione del video

La presenza del video permette all'operatore sanitario e allo specialista di presentarsi e di poter introdurre le condizioni del paziente naturalmente e senza dover ascoltare solamente l'audio.

Una volta che lo specialista inizia con la compilazione degli step, nell'applicazione viene cambiata la schermata e ci si occupa di far visualizzare le informazioni degli step ricevuti.

Quando lo specialista ha terminato di compilare tutti degli step del template, l'applicazione riattiva il flusso video per permettere all'operatore sanitario e allo specialista di poter discutere insieme interagendo fra loro, per la formulazione della diagnosi finale e l'iter da intraprendere per il paziente.

Riassumendo, nella Figura 4.5 è possibile vedere la sequenza delle schermate che vengono visualizzate durante una sessione di chiamata con lo specialista.

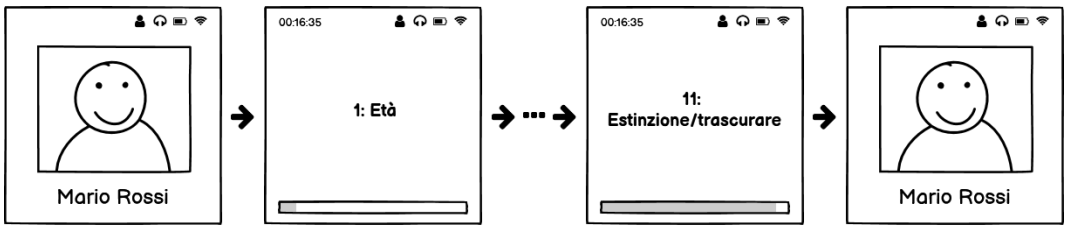


Figura 4.5: Sequenze di schermate durante una sessione

4.1.2 Frontend dello specialista

Per quanto riguarda il frontend dello specialista, sono state apportate alcune modifiche per migliorarne l'usabilità.

Nelle Figure 4.6 e 4.7 sono evidenziate con un rettangolo rosso, le varie parti aggiunte alle schermate che riguardano una sessione.

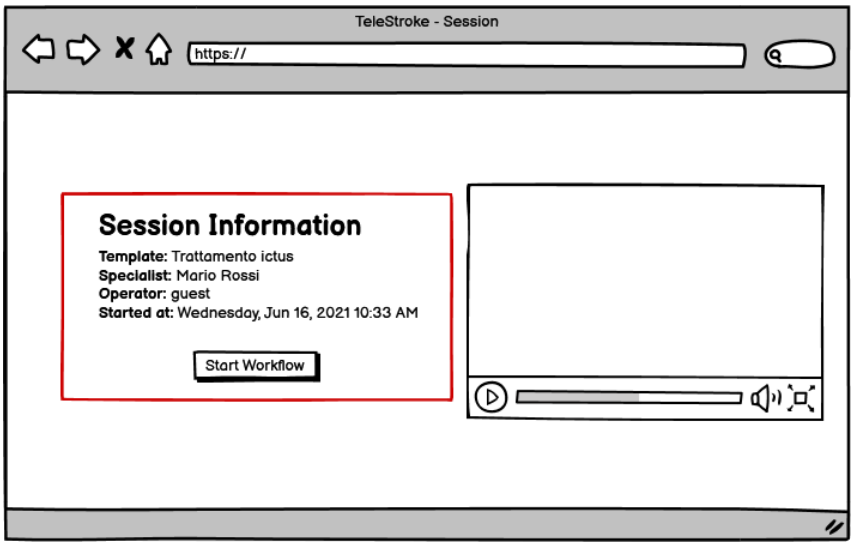


Figura 4.6: Mockup della schermata di inizio sessione

Nello specifico, il sistema permette di far visualizzare all'inizio di una sessione, le informazioni relative alla sessione corrente e include un pulsante per iniziare il flusso di lavoro, accanto del quale è possibile visualizzare il flusso video dell'operatore sanitario (Figura 4.6). In questo modo lo specialista potrà confrontarsi con l'operatore e conoscere la situazione del paziente, prima di iniziare la compilazione dei dati.

Cliccando sul pulsante, è possibile iniziare la compilazione dei dati del template selezionato. La pagina è organizzata con una navigazione a tab, in cui le prime schede rappresentano le fasi di compilazione contenente gli step associati, mentre l'ultima il riepilogo della sessione. In questo momento, l'applicazione invierà al frontend dell'operatore dei messaggi contenenti le informazioni relative agli step.

Dopo aver compilato gli step, si può arrivare alla schermata di riepilogo cliccando sull'ultimo tab. In questa scheda è possibile visualizzare i dettagli della sessione, con i suoi progressi.

È stato aggiunto nella tabella dei progressi, una colonna che mostra i risultati ottenuti per ogni fase di compilazione. Inoltre, è stato aggiungere anche una *text area* per poter scrivere le considerazioni finali e annotare la diagnosi del paziente, basandosi sulla situazione del paziente e sui risultati finali ottenuti. (Figura 4.7)

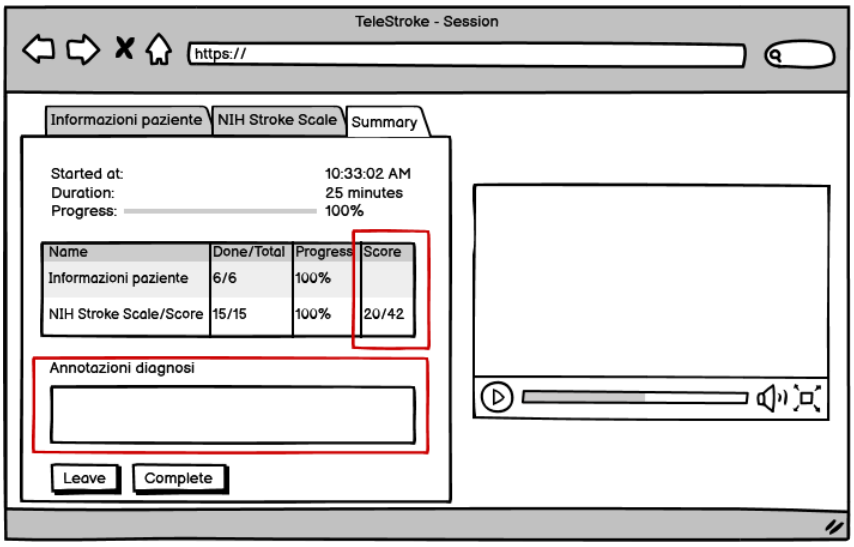


Figura 4.7: Mockup della schermata di riepilogo della sessione

4.2 Tecnologie e Linguaggi

Prima di entrare nella parte di sviluppo, è necessario fornire una panoramica generale sulle principali tecnologie e linguaggi che si andranno ad utilizzare.

4.2.1 Vuzix Blade

L'hardware utilizzato per eseguire l'applicazione frontend dell'operatore è lo smartglasses Vuzix Blade.

Vuzix Blade è un modello di *smartglasses* per *Augmented reality* prodotto dall'azienda omonima Vuzix. Presenta un display a colori solo sulla lente di destra con dimensioni 480 x 853 pixel, ma la finestra ha proporzioni 1:1 di dimensione 480 x 480 pixel, permettendo così di regolare la posizione del campo visivo in alto o in basso.

Il sistema operativo installato è una versione modificata di Android 5.1.1 Lollilop, lavora su un preprocessore Quad Core ARM con una RAM di 1 GB e incorpora una batteria interna da 470 mAh. Lo spazio di archiviazione è di 8GB con la possibilità di espanderlo fino a 32 GB inserendo una MicroSD.

È dotato di una fotocamera da 8 Megapixel e consente di registrare video con una risoluzione di 720p con 30 fps oppure 1080p con 24 fps. Include i sensori: giroscopio, accelerometro, magnetometro, sensore di luce ambientale e sensore di pressione.

Figura 4.8: Smartglasses Vuzix Blade

Infine, l'utente può interagire con il dispositivo attraverso comandi vocali oppure attraverso il *touchpad* situato nell'asta degli *smartglasses*. [26]

4.2.2 Kotlin

Il frontend dell'operatore è un'applicazione *Android* scritta tramite il linguaggio *Kotlin*[1] .

Kotlin è un linguaggio di programmazione *open source* e *general-purpose* sviluppato dall'azienda JetBrains, noto anche per aver creato l'IDE IntelliJ IDEA su cui si basa Android Studio. Inoltre, il linguaggio Kotlin è stato integrato nell'ambiente di sviluppo Android Studio ed è il linguaggio raccomandato da Google per lo sviluppo di applicazioni Android.

Di seguito vengono descritte le principali caratteristiche del linguaggio Kotlin. [15]

- È un linguaggio **conciso**, attraverso cui è possibile impiegare meno tempo per la scrittura del codice, per esempio è possibile definire velocemente classi che contengono i metodi `getters`, `setters`, `equals()`, `hashCode()`, `toString()` e `copy()` con una sola linea di codice. Inoltre, le dichiarazioni di variabili e funzioni sono state semplificate, grazie alla presenta della *type inference*, con cui non occorre più preoccuparsi di dichiarare il tipo dei dati. Un'altra caratteristica è che permette di gestire facilmente le operazioni asincrone attraverso le *coroutines*.

- È un linguaggio **interoperabile** con Java, ciò significa che si possono utilizzare le librerie Java preesistenti e allo stesso tempo le applicazioni scritte in Kotlin possono essere utilizzati con Java.

[1] Kotlin

- È un linguaggio **sicuro**, infatti molti errori vengono segnalati autonomamente dal compilatore. Un esempio è la *null safety*: le variabili in Kotlin per default non possono assumere valori nulli, per poter assegnare valori nulli bisogna dichiararli esplicitamente. Inoltre consente di chiamare metodi di tali variabili in modo sicuro, in questo modo è possibile evitare eccezioni comuni durante l'esecuzione, come la `NullPointedException`.

4.2.3 Angular

Il frontend dello specialista è un'applicazione web sviluppato con l'utilizzo del framework *Angular*[2].

Angular è un framework open source e mantenuto da Google che permette la creazione di applicazioni single page (*Single Page Application* SPA), dove le risorse sono caricate dinamicamente e aggiornate in tempo reale direttamente nel browser, senza che deve ricaricare la pagina, con lo scopo di offrire una migliore esperienza utente.

Inoltre, con Angular non solo è possibile separare la logica dell'applicazione dagli elementi grafici, ma anche organizzare la struttura in vari moduli e creare componenti che possono essere riutilizzati in più pagine, rendendo il codice modulare e riusabile. [2]

Pertanto, i componenti sono gli elementi essenziali per le applicazioni Angular, in particolare ogni componente è caratterizzato da:

- Un **template HTML** che rappresenta la view del componente, ovvero definisce tutti gli elementi necessari per la rappresentazione grafica del componente.

- Una **classe Typescript** che definisce il componente, descrive il suo comportamento e le sue funzionalità.

- Un **selettore CSS**, ovvero un nome univoco per identificare il componente. Può essere utilizzato come tag nel codice HTML, per includere il componente nella pagina. Angular si occupa di istanziare il componente ogni volta che trova il tag associato.

- Si può aggiungere, ma non è obbligatorio, anche un **file CSS** per gestire lo stile del componente.

[2] Angular

4.2.4 WebRTC e PeerJS

Si è utilizzato la libreria WebRTC[3] per instaurare una connessione streaming video real time tra i due frontend.

WebRTC (*Web Real-Time Communication*) è una tecnologia *open source* che permette di stabilire connessioni dirette *peer-to-peer* per la comunicazione di voce, video e anche dati testuali in tempo reale. [27]

È una tecnologia inclusa nello standard W3C[4] ed è supportato da quasi tutti i browser e anche dalle applicazioni mobile. Inoltre è in grado di gestire il rendering dei contenuti multimediali sia quelli remoti sia quelli locali.

Per poter utilizzare WebRTC, è richiesto un server di coordinazione che implementa il meccanismo di segnalazione (*signalling*) per poter trasmettere le informazioni necessarie per la connessione dei client, denominati *peer*. Però questa parte non è incluso nel protocollo WebRTC e perciò viene utilizzato anche la tecnologia *PeerJS*[5], una libreria che permette di semplificare l'utilizzo della tecnologia WebRTC, fornendo il protocollo di signalling e il server per la gestione dei peer.

4.2.5 Docker

Per il *deployment* del backend e del frontend dello specialista è stato utilizzato lo strumento *Docker*[6] . Docker è diventato uno strumento tra i più utilizzati per dispiegare microservizi, infatti permette di preparare agevolmente e velocemente l'ambiente di esecuzione di un'applicazione. Le applicazioni vengono eseguite in container separati creando uno spazio utente isolato, rendendo l'applicazione indipendente dall'esterno e dalla macchina fisica.

I container rappresentano delle istanze di immagini, e quest'ultime, possono essere create attraverso un *Dockerfile*, ovvero un file che contiene tutti i comandi e le istruzioni che servono alla creazione dell'immagine.

Inoltre, è possibile definire un file contenente le configurazioni di tutti i servizi necessari per il sistema, e attraverso il tool *docker-compose* è possibile automatizzare la creazione di tutti i servizi attraverso un singolo comando.

[3]WebRTC - Web Real Time Communication
[4]W3C - World Wide Web Consortium
[5]PeerJS
[6]Docker

4.3 Sviluppo e Implementazione del Frontend Android

A seguito delle osservazioni effettuate con la fase di analisi e di progettazione, è possibile procedere con la fase di sviluppo e implementazione, in questa sezione verranno trattate le scelte implementative più rilevanti per il refactoring del frontend Android. Come descritto in precedenza, il frontend Android, è l'applicazione eseguita sugli smartglasses Vuzix Blade e utilizzata dall'operatore sanitario,

L'applicazione è stata sviluppata con il linguaggio di programmazione Kotlin ed è pensato per funzionare su più versioni Android: sia su una versione meno recente per il dispositivo Vuzix Blade, con una versione di Android 5.1.1 e API level 22, sia per dispositivi con una versione più recente.

In generale il progetto è stato organizzato in due macro attività:

- **MainActivity**. Rappresenta l'Activity iniziale dell'applicazione, in questa schermata viene mostrato il titolo, gli stati e un menu contenente una serie di azioni selezionabili.

- **SessionActivity**. Rappresenta l'Activity che gestisce la sessione di chiamata con lo specialista. Essa si occupa di visualizzare il flusso video e le informazioni degli step che sono state ricevute.

Di seguito verranno mostrate i vari componenti aggiunti per migliorare l'applicazione dell'operatore sanitario.

4.3.1 Creazione dell'Action Menu

Per la creazione del menu è stato scelto di utilizzare l'*Action Menu* fornito da Vuzix invece di usare il menu tradizionale di Android, il quale non è adatto per la visualizzazione sui dispositivi *wearable* con schermi piccoli. L'Action Menu è stato progettato specificatamente per gli *smartglasses* infatti è caratterizzato da forme e colori altamente visibili come descritto nella sezione 3.2.2.

Per inserirlo nell'interfaccia grafica, prima di tutto si devono creare e definire gli elementi che verranno mostrati nel menu, creando una risorsa di menu in un file XML separato, situato nella directory `res/menu/`. In questo modo si avrà la possibilità di separare il codice per la definizione degli elementi dall'effettivo comportamento dell'applicazione, rendendo il codice sviluppato modulare e riusabile.

Come si può vedere nel Listato 4.1, il menu può essere definito con il tag `<menu>`, e può contenere vari elementi definiti mediante il tag `<item>`.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity">

    <item
        android:id="@+id/action_menu_call"
        android:title="@string/new_call"
        android:icon="@drawable/ic_baseline_add_ic_call_24"/>

    <item
        android:id="@+id/action_menu_settings"
        android:title="@string/settings"
        android:icon="@drawable/ic_baseline_settings_24"/>

    <item
        android:id="@+id/action_menu_scan_qrcode"
        android:title="@string/change_settings"
        android:icon="@drawable/ic_baseline_qr_code_scanner_24"/>

</menu>
```

Listato 4.1: Creazione e definizione degli elementi del menu

Ogni elemento del menu può possedere diversi attributi, in particolare:

- `android:id`. Rappresenta l'ID univoco dell'elemento, con cui è possibile riferirsi all'oggetto.
- `android:icon`. Rappresenta il riferimento a un drawable che può essere associato all'elemento come icona.
- `android:title`. Rappresenta il titolo associato all'elemento.
- `android:showAsAction`. Specifica come questo elemento deve essere visualizzato.

Per caricare questi elementi nell'Action Menu, la `MainActivity` che è l'Activity associata alla schermata, deve estendere la classe `ActionMenuActivity` fornita dalle librerie di Vuzix come raffigurato nel Listato 4.2.

La classe fornisce metodi che possono essere sovrascritti per creare un menu d'azione personalizzato, in particolare nel Listato 4.3 vengono illustrati i vari metodi che sono stati sovrascritti.

```
package it.unibo.telestroke

import android.os.Bundle
import com.vuzix.hud.actionmenu.ActionMenuActivity

//This class extends the ActionMenuActivity
class MainActivity : ActionMenuActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    //..
}
```

Listato 4.2: Creazione della classe MainActivity

Si può chiamare il metodo `onCreateActionMenu()` per ottenere il momento in cui è stato creato il menu d'azione di base, contenente di default il pulsante per tornare indietro. Si deve notare che tale metodo è diverso dal `onCreateOptionMenu()` che usiamo solitamente per la creazione del menu tradizionale di Android. Sovrascriviamo questo metodo per dire di riempire il menu d'azione con gli elementi precedentemente definiti, chiamando il metodo `menuInflater.inflate`.

Inoltre è possibile personalizzare altre caratteristiche, infatti è stato ridefinito il metodo `alwaysShowActionMenu()` con il quale è possibile specificare se il menu deve essere sempre visibile oppure nascosto con la possibilità di richiamarlo con una interazione nel *touchpad*. Nel progetto, si è deciso di rendere il menu sempre visibile affinché l'utente possa visualizzare e scegliere facilmente l'opzione desiderata, perciò il valore di ritorno del metodo è settato a `true`.

È stato sovrascritto anche il metodo `getDefaultAction()` per specificare l'elemento "New Call", elemento di posizione 1, come elemento di default situato sempre al centro del menu ogni volta che riparte l'applicazione.

```
//Create the Vuzix ACTION Menu and not the Android Option Menu.
override fun onCreateActionMenu(menu: Menu?): Boolean {
    super.onCreateActionMenu(menu)
    menuInflater.inflate(R.menu.main_menu, menu)
    return true
}
```

```
//True if always show the Action Menu, false oltherwise.
override fun alwaysShowActionMenu(): Boolean {
    return true
}

//The default action item to start at on activity restarts.
override fun getDefaultAction(): Int {
    return 1
}
```

Listato 4.3: MainActivity.kt, creazione di un Action Menu

4.3.2 Gestione degli stati dell'applicazione

Come definito nelle sezioni precedenti, l'applicazione consente di tenere sotto controllo una serie di stati del dispositivo, rappresentati con differenti icone e colori, e situate nella barra degli strumenti.

Gli stati, in particolare sono: connessione alla rete, livello e stato della batteria del dispositivo, tipo di uscita audio e connessione con lo specialista. Di seguito verranno descritte in dettaglio gli aspetti principali per l'implementazione di ciascun stato.

Connessione alla rete

Per controllare lo stato della connessione di rete, è stato registrato un `NetworkCallback`[7], ovvero delle funzioni che vengono chiamate in risposta a degli eventi della rete.

Come si può vedere nel Listato 4.4, sono state registrate due funzioni per rilevare il cambio dello stato di rete:

- `onAvailable`. Questo metodo viene chiamato quando il sistema rileva che il dispositivo è connesso alla rete.
- `onLost`. Questo metodo viene chiamato quando il sistema rileva che il dispositivo è disconnesso dalla rete.

```
private val networkCallBack = object :
    ConnectivityManager.NetworkCallback() {

        // Called when a network is ready for use
        override fun onAvailable(network: Network) {
```

[7] ConnectivityManager.NetworkCallback

```
            GlobalScope.launch(Dispatchers.Main) {
                setStatusItem(NetworkStatus.On)
            }
        }

        // Called when a network disconnects
        override fun onLost(network: Network) {
            GlobalScope.launch(Dispatchers.Main) {
                setStatusItem(NetworkStatus.Off)
            }
        }
    }
```

Listato 4.4: NetworkCallback

Livello della batteria

Per implementare lo stato del livello della batteria, è stato utilizzato un `BroadcastReceiver`[8], un componente di Android che permette la registrazione agli eventi generati dal sistema. Il sistema, al verificarsi dell'evento, invia automaticamente un messaggio broadcast a tutte le applicazioni registrati all'evento, seguendo il modello *publish-subscribe*.

In particolare è stato registrato l'evento `Intent.ACTION_BATTERY_CHANGED`, con il quale viene notificato quando il sistema ha rilevato un cambiamento dello stato di carica o del livello della batteria. Nel Listato 4.5 è possibile vedere la creazione dell'oggetto BroadcastReceiver e la registrazione all'evento.

```
this.batteryLevelReceiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        if (intent.action == Intent.ACTION_BATTERY_CHANGED) {
            val level = intent.getIntExtra("level", -1)
            val status =
                intent.getIntExtra(BatteryManager.EXTRA_STATUS, -1)

            if (level != -1 && status != -1) {
                Log.d(TAG, "Battery level changed: $level")
                setStatusItem(BatteryStatus(level, status))
            }
        }
    }
}
```

[8] BroadcastReceiver

```
//Register a BroadcastReceiver for the battery level information
application.registerReceiver(batteryLevelReceiver,
    IntentFilter(Intent.ACTION_BATTERY_CHANGED))
```

Listato 4.5: BroadcastReceiver per il livello della batteria

Inoltre lo stato della batteria, viene associato a una icona e un colore differente a seconda del livello corrente, come mostrato nella Figura 4.9.

Figura 4.9: Le icone e i colori associati allo stato della batteria

Questo meccanismo è possibile implementarlo con un `LevelListDrawable`[9]. Ovvero è una risorsa che può gestire diversi Drawable, ciascuno assegnato a un intervallo di valori che rappresenta il livello. Nel codice, è possibile specificare il livello del LevelListDrawable settando l'attributo `level`, e durante l'esecuzione verrà caricato l'immagine associato al valore corrente del livello.

Il LevelListDrawable può essere creato in un file XML situato nella cartella `res/drawable/` e può essere definito con il tag `<level-list>`. Può contenere vari elementi rappresentati come un `item`. Ogni elemento può essere associato ad un intervallo di livelli oppure può essere specificato solamente il massimo livello, inoltre ogni elemento contiene un riferimento ad un altro drawable, come mostrato nel Listato 4.6.

```
<?xml version="1.0" encoding="utf-8"?>
<level-list
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:minLevel="0" android:maxLevel="5"
        android:drawable="@drawable/ic_baseline_battery_alert_24" />
    <item android:minLevel="6" android:maxLevel="15"
        android:drawable="@drawable/ic_battery_level10_24" />
    <item android:minLevel="16" android:maxLevel="25"
        android:drawable="@drawable/ic_battery_level20_24" />
    <item android:minLevel="26" android:maxLevel="35"
        android:drawable="@drawable/ic_battery_level30_24" />

    ...
```

[9] LevelListDrawable

```
</level-list>
```

Listato 4.6: LevelListDrawable per il livello della batteria

Dispositivo di uscita audio

Per rilevare il dispositivo di uscita audio, sono state identificate diverse situazioni, infatti è possibile che il dispositivo abbia un altoparlante interno, oppure sia collegato ad un auricolare normale o connesso ad un auricolare Bluetooth.

È possibile verificare se il dispositivo presenta un altoparlante interno attraverso la chiamata al metodo `hasSystemFeature` di packageManager, specificando come argomento `PackageManager.FEATURE_AUDIO_OUTPUT`.

Per gli altri casi, è necessario registrare un BroadcastReceiver sui relativi eventi, in particolare:

- `AudioManager.ACTION_HEADSET_PLUG`. Viene inviato un messaggio quando vengono inserite o scollegati gli auricolari.
- `BluetoothDevice.ACTION_ACL_CONNECTED`. Viene inviato un messaggio se è stato connesso un dispositivo Bluetooth, perciò una volta ricevuto tale messaggio viene effettuato un controllo per vedere se la tipologia del dispositivo è un auricolare.
- `BluetoothDevice.ACTION_ACL_DISCONNECTED`. Viene inviato un messaggio se è stato disconnesso un dispositivo Bluetooth, anche in questo caso viene effettuato il controllo della tipologia del dispositivo.
- `BluetoothHeadset.ACTION_VENDOR_SPECIFIC_HEADSET_EVENT`. Viene inviato un messaggio contenente gli aggiornamenti delle informazioni inerenti allo stato dell'auricolare Bluetooth, è stato utilizzato per ricavare il livello della batteria.

Infine, nella Figura 4.10 viene mostrato gli stati che possono essere associate al dispositivo di uscita audio con le relative icone e colori.

Figura 4.10: Possibili stati associate al dispositivo di uscita audio

Connessione con lo specialista

Lo stato che rappresenta la connessione con lo specialista viene mostrato solo durante la sessione, per evitare di confondere ulteriormente l'operatore sanitario.

In particolare, la `SessionActivity` implementa anche l'interfaccia `WebRtcEventListener` e si mette in ascolto degli eventi generati dalla connessione WebRTC. Lo stato della connessione con lo specialista, viene attivata all'interno del metodo `onCallRequest()`, ovvero è il momento in cui viene ricevuto la richiesta di chiamata dallo specialista. Mentre viene disattivata con la chiamata al metodo `onSignallingConnectionClosed()`, momento in cui viene ricevuto il segnale di chiusura della connessione, oppure all'interno del metodo `onSignallingConnectionError()`, il la quale viene chiamato a causa di un errore nella connessione.

4.3.3 Interfaccia grafica MainActivity

Nella Figura 4.11 viene illustrato la schermata principale con gli elementi precedentemente descritti: il titolo, la barra degli stati e il menu con le opzioni selezionabili.

Figura 4.11: Schermata MainActivity

Cliccando sull'opzione "New Call", è possibile avviare una nuova chiamata, registrando la disponibilità dell'operatore sanitario.

Inoltre, considerando che per avviare una chiamata, è essenziale avere una connessione a Internet e un livello della batteria non troppo basso; è stato impedito di avviare la chiamata in queste situazioni e per indicarlo, è stato aggiunto una animazione di tipo blink (Listato 4.7).

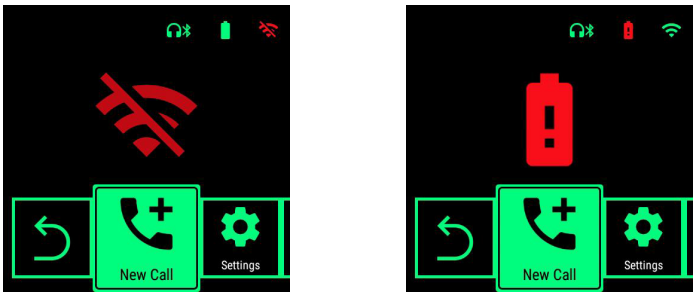
In particolare, l'animazione ha un periodo di 200 ms, inizia nascondendo il titolo, in modo da mostrare lo stato che ha dato problemi al centro della

```
private fun hideTextAndBlinkImageAnimation(text: View, img: View) {
    val blink: Animation = AlphaAnimation(1F, 0F)
    blink.interpolator = AccelerateInterpolator()
    blink.duration = 200
    blink.repeatCount = 5
    blink.repeatMode = Animation.REVERSE
    blink.setAnimationListener(object : Animation.AnimationListener {
        override fun onAnimationEnd(animation: Animation?) {
            img.visibility = View.GONE
            text.visibility = View.VISIBLE
        }

        override fun onAnimationRepeat(animation: Animation?) {}
        override fun onAnimationStart(animation: Animation?) {
            text.visibility = View.GONE
        }
    })
    img.startAnimation(blink)
}
```

Listato 4.7: Animazione per avvisare all'utente gli stati anomali

schermata, lampeggiando per 5 volte e una volta finito viene visualizzato di nuovo il titolo. Nella Figura 4.12 è possibile vedere degli screenshot al momento dell'animazione.

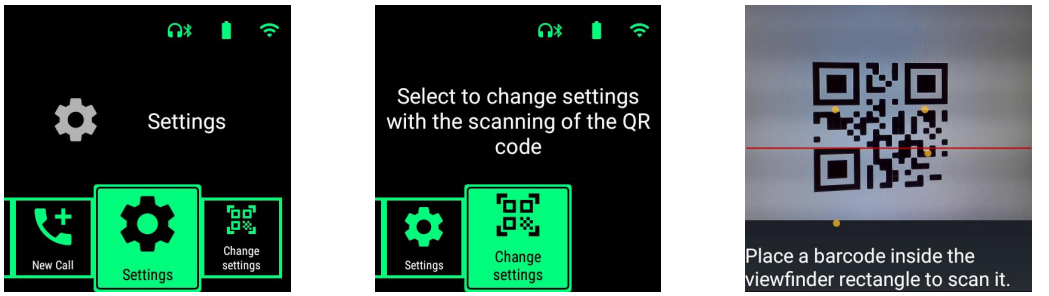


(a) Assenza di connessione alla rete

(b) Livello di batteria troppo basso

Figura 4.12: Indicazioni per gli stati d'allarme

Inoltre, la schermata viene cambiata dinamicamente con lo scorrere del menu, mostrando le informazioni associate all'opzione, come raffigurato nella Figura 4.13.



(a) Settings (b) Change Settings (c) Scansione del QRcode

Figura 4.13: Schermata principale con le varie opzioni selezionabili

Cliccando sull'opzione "Change Settings" è possibile cambiare le impostazioni mediante la scansione di un codice QR (Figura 4.13b). La scansione, visibile nella Figura 4.13c, viene affidata allo scanner del sistema, richiamando l'Intent specifico `ScannerIntent.ACTION`, fornita dalle librerie di Vuzix.

4.3.4 Visualizzazione del flusso video

Come accennato precedentemente, la comunicazione di informazioni tra i due Frontend durante una sessione di chiamata, viene gestita con le tecnologie WebRTC e PeerJS.

Per quanto riguarda la visualizzazione del flusso video, è stato definito prima di tutto in un file XML, un contenitore `org.webrtc.SurfaceViewRenderer`, reso disponibile dalle librerie WebRTC, come illustrato nel Listato 4.8.

```
<org.webrtc.SurfaceViewRenderer
    android:id="@+id/remote_view_renderer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

Listato 4.8: SurfaceViewRenderer

Il Render viene prima inizializzato con le impostazioni definite che riguardano la risoluzione del video e il numero di frame al secondo. Successivamente, ogni volta che si ricevono dei flussi di video, questi, vengono impostati nel Renderer con il metodo `setRenderer` della classe `MediaConnection`, il quale si occupa di gestire tutte le connessioni multimediali, come mostrato nel Listato 4.9.

```
private fun manageMediaConnection(connection: MediaConnection) =
    GlobalScope.launch(Dispatchers.Default) {
    var notSet = true

    for (track in connection.awaitVideoTracks()) {
        Log.i(TAG, "Received new remote video track: \$track")
        if (notSet) {
            launch(Dispatchers.Main) {
                connection.setRenderer(track, remote_view_renderer)
            }
            notSet = false
        } else {
            Log.i(TAG, "Skipped video track because the rendered is
                already set: \$track")
        }
    }
}
```

Listato 4.9: Metodo che gestisce i flussi di video ricevuti

4.3.5 Cronometro

Il cronometro viene mostrato durante una sessione per fornire un'indicazione del tempo trascorso all'operatore sanitario.

Per implementarlo è stato usato l'oggetto `Chronometer`[10] messo a disposizione da Android. L'elemento Chronometer può essere semplicemente definito nel file XML con il tag apposito, come mostrato nel Listato 4.10.

```
<Chronometer
    android:id="@+id/chronometer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

Listato 4.10: Chronometer

Come default, il valore del timer viene mostrato nel formato "MM:SS" oppure "H:MM:SS" a seconda del valore corrente.

Nel Listato 4.11 mostra i metodi che sono stati chiamati al lato codice per iniziare e per stoppare il timer. In particolare, è possibile far iniziare il cronometro chiamando il metodo `start()`, resettando come tempo base `SystemClock.elapsedRealtime()`, mentre è possibile fermarlo con il metodo `stop()`.

[10] Chronometer

```
//Start the timer.
private fun startTimer() {
    chronometer.base = SystemClock.elapsedRealtime()
    chronometer.start()
    chronometer.visibility = View.VISIBLE
}

//Stop the timer.
private fun stopTimer() {
    chronometer.stop()
    chronometer.visibility = View.GONE
}
```

Listato 4.11: Metodi per iniziare e fermare il Chronometer

4.3.6 Stato di avanzamento degli step

Per mostrare lo stato di avanzamento degli step, è stata creata una `ProgressBar`[11].

Durante la sessione, vengono anche trasmessi le indicazioni sullo stato di avanzamento, oltre alle informazioni degli step, perché in questo modo, lo stato rimane aggiornato anche dopo una ripresa di una sessione precedentemente non terminata.

Nel Listato 4.12, viene mostrato la definizione dell'elemento ProgressBar nella View, nello specifico oltre a indicare l'id e le dimensioni, è stato impostato lo stile come `Widget.AppCompat.ProgressBar.Horizontal` per ottenere una ProgressBar orizzontale invece, di utilizzare una ProgressBar circolare impostata di default.

Il valore della ProgressBar è compreso in un intervallo tra 0 e 100, ed è possibile aggiornarlo dal codice chiamando semplicemente il metodo `setProgress(int)`.

```
<ProgressBar
    android:id="@+id/stepProgress"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>
```

Listato 4.12: ProgressBar

[11] ProgressBar

4.3.7 Scambio delle informazioni durante una sessione

Come accennato in precedenza, il canale di comunicazione WebRTC consente di trasmettere anche dei dati ai peer.

In particolare sono state individuate diverse tipologie di messaggi che possono essere inviate, per cui è stata creata una enumerazione, mostrata nel Listato 4.13, contenente le tipologie:

- `Started`. Questo messaggio viene inviato all'inizio della comunicazione, contiene le informazioni riguardanti il nome e il cognome dello specialista, da mostrare sotto al flusso video.
- `NextStep`. Questo messaggio contiene le informazioni degli step con il titolo e l'indicazione sullo stato di avanzamento. Grazie a tale messaggio otteniamo le informazioni inerenti agli step, da mostrare nell'interfaccia e possiamo aggiornare la *progress bar*.
- `LastStep`. Questo messaggio viene inviato quando lo specialista ha terminato di compilare l'ultimo step. Con tale messaggio, l'applicazione viene cambiata schermata, mostrando il flusso video.
- `Finished`. Questo messaggio viene inviato quando la sessione è terminata correttamente e tutti i campi sono stati compilati.
- `Aborted`. Questo messaggio viene inviato quando si vuole interrompere la sessione senza aver finito di compilare tutti i campi.

```
enum class MessageType {
    @SerializedName("0")
    Started,
    @SerializedName("1")
    NextStep,
    @SerializedName("2")
    LastStep,
    @SerializedName("3")
    Finished,
    @SerializedName("4")
    Aborted
}
```

Listato 4.13: Le possibili tipologie di messaggi che possono essere trasmesse

Il messaggio trasmesso viene rappresentato con una classe `MessageBody` che contiene un riferimento alla tipologia e se presente, un messaggio `data`

rappresentato come una Stringa. La classe viene creata mediante l'utilizzo della `data class`, fornito dal linguaggio Kotlin, come raffigurato nel seguente Listato.

```
data class MessageBody(val type: MessageType, val data: String)
```

Listato 4.14: MessageBody

Il messaggio viene ricevuto dal canale di comunicazione in formato JSON, e per ognuno di questi, viene scomposto e riempito nei campi della classe `MessageBody` attraverso il comando `gson.fromJson()` fornita dalla libreria Gson[12] di Google.

Questo metodo prende una stringa in formato JSON e la classe con cui si vuole convertire la stringa, e restituisce direttamente l'oggetto istanziato con i campi riempiti dai dati presenti nel JSON.

Il campo `data` del messaggio ricevuto, può essere ulteriormente scomposto e in base alle informazioni, può essere rappresentato come un oggetto `SpecialistInfo`, contenente le informazioni dello specialista, oppure come un oggetto `StepInfo`, contenente le informazioni degli step.

Nel Listato 4.15 viene raffigurato la gestione dei dati ricevuti dal canale di comunicazione. Si può vedere che a seconda della tipologia del messaggio ricevuto, vengono eseguite diverse operazioni.

```
//..

for (channel in connection.awaitExchanges()) {
    Log.i(TAG, "Received a new data channel (${channel.id}")
    val gson = Gson()
    try {
        launch(Dispatchers.Main) {
            for (data in channel.receive()) {
                try {
                    val messageBody = gson.fromJson(data,
                        MessageBody::class.java)

                    when (messageBody.type) {
                        MessageType.Started -> {
                            val specialistInfo =
                                gson.fromJson(messageBody.data,
                                SpecialistInfo::class.java)
                            specialist_info.text = String.format("%s
                                %s", specialistInfo.firstName,
```

[12] Gson

```
                            specialistInfo.lastName)
                        startTimer()
                        showSnackBar("Session started")
                    }
                    MessageType.NextStep -> {
                        val stepInfo =
                            gson.fromJson(messageBody.data,
                            StepInfo::class.java)
                        val message = stepInfo.name
                        if (currentLayout == video_layout) {
                            setLayoutVisible(step_layout)
                        }
                        changeStepDescription(message)
                        setStepProgress(stepInfo.progress)
                    }
                    MessageType.LastStep -> {
                        setLayoutVisible(video_layout)
                    }
                    MessageType.Finished -> {
                        stopTimer()
                        showSnackBar("Session completed")
                    }
                    MessageType.Aborted -> {
                        stopTimer()
                        showSnackBar("Session aborted")
                    }
                }
            } catch (e: Throwable) {
                Log.e(TAG, "Unable to parse incoming message", e)
            }
        }
    }
    } catch (e: Throwable) {
        Log.e(TAG, "Channel closed: ${channel.id}")
    }
}

//..
```

Listato 4.15: Gestione dei dati ricevuti

4.3.8 Interfaccia grafica SessionActivity

All'attivazione della sessione, la schermata SessionActivity mostra il flusso video dello specialista con un'informazione del suo nome e cognome, come si può vedere nella Figura 4.14. Nella barra superiore, possiamo vedere il cronometro e gli stati del dispositivo.

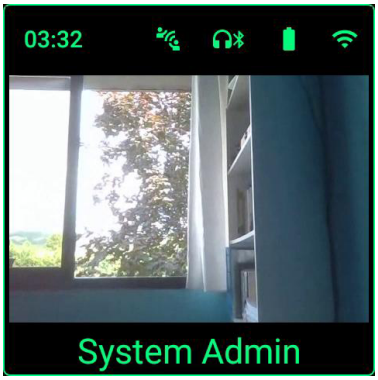


Figura 4.14: Schermata che mostra il flusso video dello specialista

Come spiegato precedentemente, una volta iniziata la compilazione degli step, il video viene nascosto, e viene visualizzata le informazioni degli step con la barra di avanzamento nella parte inferiore.

Nella Figura 4.15 sono illustrate degli esempi di step mostrati durante una sessione, e in particolare la Figura 4.15a viene raffigurato uno step appartenente alla *checklist* "Informazioni paziente", mentre la Figura 4.15b rappresenta uno step appartenente alla *checklist* "NIH Stroke Scale".

I colori sono stati scelti seguendo le palette e le indicazioni descritte nella Sezione 3.2.3, nello specifico il bianco e il verde sono i due colori primari utilizzati per le icone e i testi, mentre altri colori quali giallo e rosso sono stati utilizzati per rappresentare gli stati con allarme. Le dimensioni del testo sono state ingrandite opportunamente per garantire la leggibilità.

Inoltre è stato aggiunto un bordo sottile verde per delineare i confini della schermata, al fine di evidenziare meglio il display dell'applicazione.

(a) Informazioni del paziente (b) Step NIHSS

Figura 4.15: Esempi di step

4.4 Sviluppo e Implementazione del Frontend Angular

Il Frontend Angular, è un'applicazione web pensata per essere utilizzata dallo specialista.

Come già accennato, il frontend Angular già sviluppato in precedenza dal gruppo di ricerca, risulta piuttosto completo, tuttavia, sono state definite alcune funzionalità nella fase di analisi con l'obiettivo di portare un miglioramento nell'interfacce e aumentare l'usabilità.

Innanzitutto tutte le schermate sono state rese **responsive**, perciò è possibile utilizzare agevolmente l'applicazione indipendente dalla dimensione dello schermo del dispositivo.

Di seguito verranno descritte le scelte implementative delle parti aggiunte nelle schermate, facendo riferimento ai requisiti definiti nella fase di analisi e ai mockup realizzati nella fase di progettazione.

4.4.1 Informazioni della sessione

Una volta iniziata la sessione, è possibile visualizzare le informazioni della sessione mostrando: i dati relativi al *template* selezionato, il nome dei professionisti e il tempo di inizio.

È stato creato un componente `session-info` separato per rappresentare le informazioni, in modo da poter riutilizzare lo stesso componente in più schermate.

Nel Listato 4.16, mostra la classe Typescript che definisce il componente `session-info`. Un componente può essere definito attraverso il decoratore

`@Component` specificando un selettore CSS, il file HTML da utilizzare come template e un foglio di stile CSS.

La classe `SessionInfoComponent` è una semplice classe dove prende in input le informazioni della sessione corrente da far visualizzare nel template.

```
import { Component, Input} from "@angular/core";
import { CompleteSessionInfo } from
    "src/app/models/info/complete-session-info";

@Component({
    selector: 'app-session-info',
    templateUrl: './session-info.component.html',
    styleUrls: ['./session-info.component.css']
})

export class SessionInfoComponent{

    @Input("sessionInfo")
    sessionInfo: CompleteSessionInfo;
}
```

Listato 4.16: Classe del componente

Infatti nel Listato 4.17, è raffigurato il template HTML del componente, come si può notare, è possibile accedere alle variabili della classe grazie al `data binding` di Angular.

In particolare, Angular interpreta tutte le espressioni che sono racchiuse nelle doppie parentesi graffe: {{ }} e perciò nel codice, le informazioni della sessione sono ricavate accedendo alle proprietà della variabile `sessionInfo`.

```
<div>
    <h2>Session Information</h2>
    <mat-list>
        <mat-list-item>
            <span class="bold-text">Template: </span>
            {{ sessionInfo.template.name }}
        </mat-list-item>
        <mat-list-item>
            <span class="bold-text">Specialist: </span>
            {{ sessionInfo.specialist.firstName }}
            {{ sessionInfo.specialist.lastName }}
        </mat-list-item>
        <mat-list-item>
            <span class="bold-text">Operator: </span>
```

```
            {{ sessionInfo.operator.firstName }}
            {{ sessionInfo.operator.lastName }}
        </mat-list-item>
        <mat-list-item>
            <span class="bold-text">Started at: </span>
            {{ sessionInfo.startDate | amDateFormat: 'LLLL' }}
        </mat-list-item>
        <mat-list-item>
            <span class="bold-text">Finished at: </span>
            {{ (sessionInfo.endDate | amDateFormat: 'LLLL') || 'In
                progress' }}
        </mat-list-item>
    </mat-list>
</div>
```

Listato 4.17: Template HTML del componente

Definito ora il componente, è possibile richiamarlo attraverso il selettore `<app-session-info>` e specificando come parametro di input la sessione di cui si vuole visualizzare le informazioni.

Il Listato 4.18 rappresenta la pagina principale della sessione, e in particolare la parte iniziale con le informazioni della sessione corrente e un pulsante per iniziare la compilazione.

```
<div class="session-container">
    <div class="template-container"
        [ngClass]="{ 'template-container-extended': disableVideo }">

        <div class="start-container" *ngIf="!isCompiling">
            <app-session-info [sessionInfo]="sessionInfo">
                </app-session-info>
            <button mat-raised-button
                (click)="startCompiling()">Start workflow</button>
        </div>

        ...

      </div>
</div>
```

Listato 4.18: Template HTML della sessione, nel listato viene raffigurato il riferimento al componente

L'interfaccia grafica di questa pagina, è possibile visualizzarla nella Figura 4.16. In particolare, cliccando sul pulsante per iniziare il flusso di lavoro, viene abi-

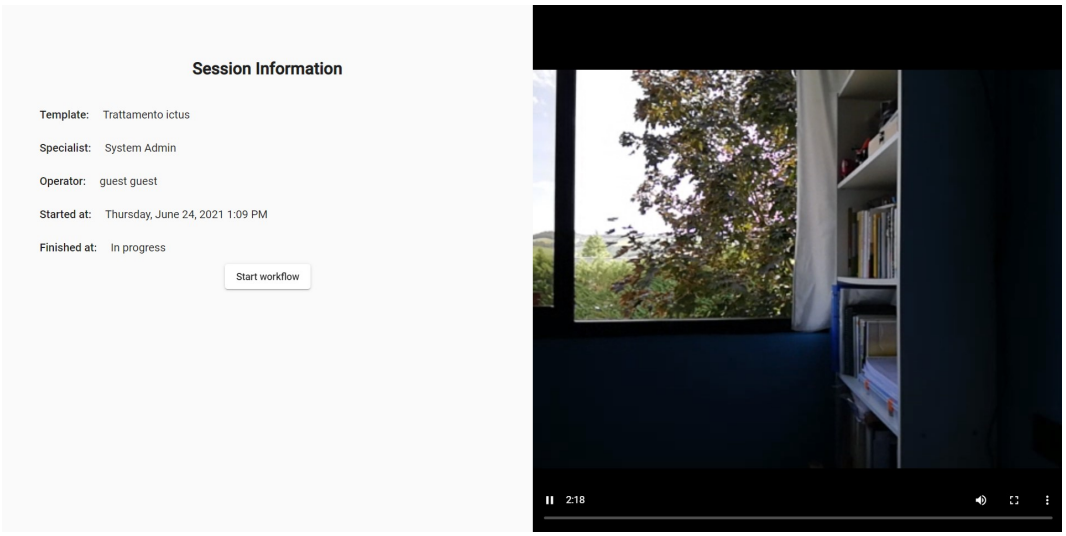


Figura 4.16: Interfaccia grafica della schermata di inizio sessione

litato la pagina che consente di compilare i dati. A questo punto, le informazioni degli step verranno trasmesse al frontend dell'operatore, che li potrà visualizzare negli smartglasses.

4.4.2 Compilazione di dati

La schermata per la compilazione dei dati è possibile vedere nella Figura 4.17. La pagina, come accennato in precedenza, è organizzato con una navigazione a tab. Le prime schede rappresentano le *checklist*, ovvero le fasi di compilazione, mentre l'ultima scheda contiene il riepilogo e i progressi dei dati compilati.

Si può notare che l'interfaccia è *scrollable*, e i campi di testo sono stati resi tutti visibili, in modo che si adattino a tutte le dimensioni dello schermo.

4.4.3 Invio dei messaggi

Durante la sessione, vengono inviati oltre all'audio e al video, anche dei dati necessari per il funzionamento dell'applicazione mobile.

I dati inviati sono coerenti con i dati che sono ricevuti dal frontend dello specialista, descritti nella sezione 4.3.7. I messaggi vengono inviati al peer remoto connesso attraverso il metodo `sendMessage` illustrato nel Listato 4.19. Come parametro prende in ingresso: la tipologia del messaggio e un oggetto che rappresenta i dati da inviare. Viene creato di seguito un oggetto di tipo `MessageBody` contenente la tipologia e, se presente, l'oggetto `data` serializzato

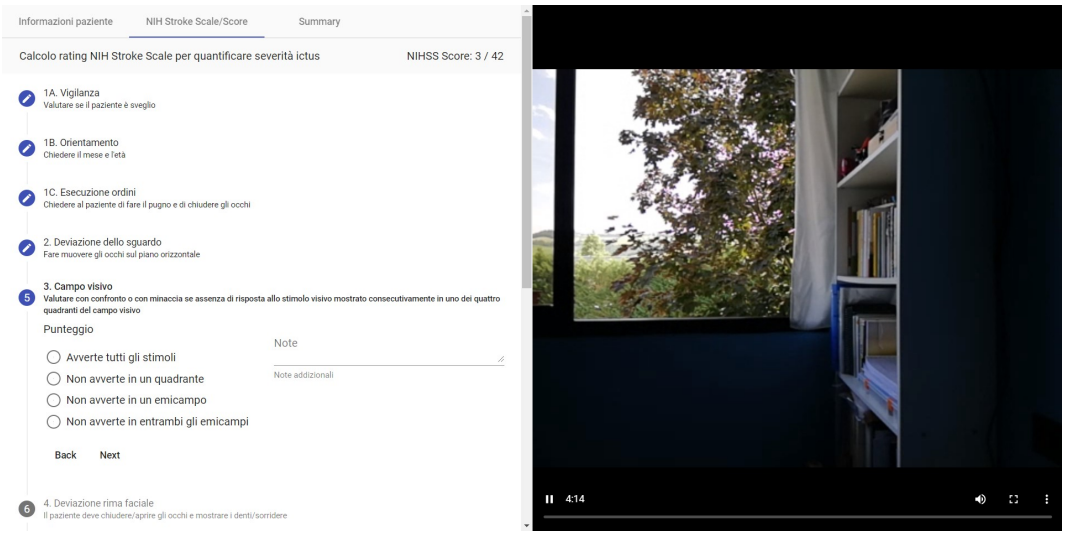


Figura 4.17: Interfaccia grafica della schermata per la compilazione dei dati

come una stringa JSON. Il messaggio verrà poi inviato al *peer* chiamando il metodo `peerService.sendData(message)`.

```
// Sends a message to the remote peer.
private sendMessage(type: MessageType, data?: any) {
    let message = new MessageBody(type);
    if (data) {
        message.data = JSON.stringify(data);
    }
    this.peerService.sendData(message);
}
```

Listato 4.19: Metodo che si occupa di inviare i messaggi

4.4.4 Schermata di Riepilogo sessione

Una volta compilati i dati, è possibile accedere alla pagina di riepilogo della sessione che viene raffigurato nella Figura 4.18. In particolare è stato aggiunto nella tabella che rappresenta il riepilogo delle fasi di compilazione, ovvero le *checklist* del template, una nuova colonna "Result". In questa colonna viene mostrato, se presente, i risultati associati della fase.

In questo modo lo specialista avrà una visione globale dei risultati ottenuti per ogni fase di compilazione e in base a questi dati, potrà formulare una diagnosi per il paziente.

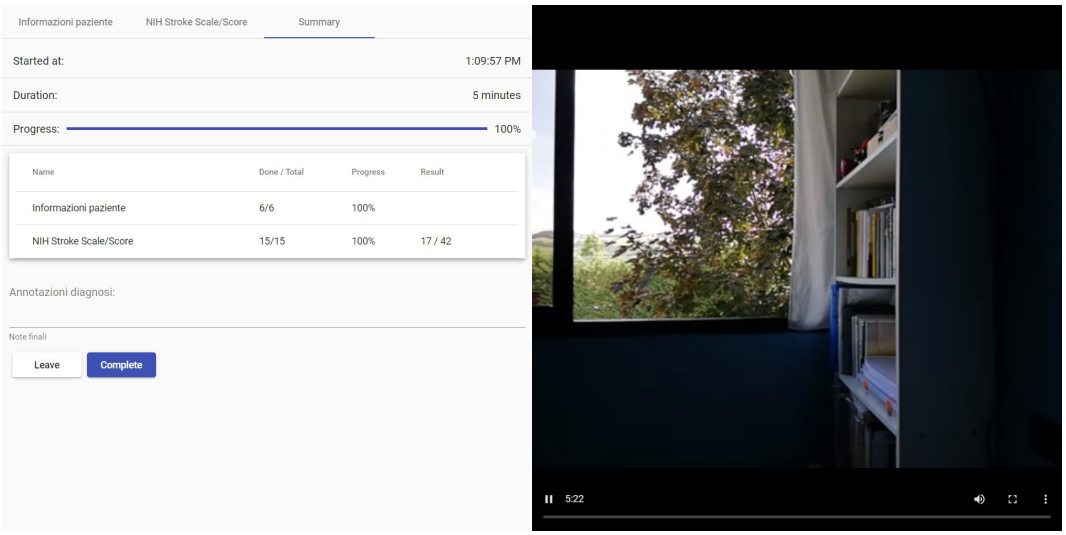


Figura 4.18: Interfaccia grafica della schermata di riepilogo della sessione

A tale scopo, è stato aggiunto anche una text-area con la direttiva di Angular Material *matInput*[13] per consentire allo specialista di inserire le annotazioni della diagnosi del paziente e di mantenerli salvati nel sistema.

[13]Angular Material: Input

Capitolo 5

Validazione

La fase di validazione viene fatta durante tutta la fase di sviluppo, per verificare il corretto funzionamento dei componenti. In questo capitolo verranno descritte le varie prove che sono state effettuate per i frontend e per concludere le osservazioni finali e alcune proposte per sviluppi futuri.

5.1 Validazione dei Frontend

I frontend sono stati testati durante tutta la fase di sviluppo, per verificare il corretto funzionamento e l'interazione tra i componenti. Per il testing è stato simulato l'ambiente di esecuzione creando un server sulla macchina locale. Nello specifico si è utilizzato lo strumento *Docker* per dispiegare i microservizi del backend e anche il frontend Angular. Di seguito verranno descritte le validazioni e considerazioni suddivise nei due frontend.

5.1.1 Frontend Android

L'applicazione dell'operatore è stata provata sia su uno smartphone Android, sia sugli smartglasses Vuzix Blade, per garantire il corretto funzionamento sui diversi tipi e sulle diverse configurazioni dei dispositivi. In particolare, è stato verificato se: il sistema gestisse correttamente i vari stati, lo scambio dei messaggi e le funzionalità aggiunte nelle schermate durante la sessione.

Per quanto riguarda il comportamento dei vari stati, sono state effettuate diverse prove per il corretto rilevamento della connessione/disconnessione della rete Internet, lo stato e il livello della batteria, la presenza di un altoparlante interno e la presenza di auricolari con fili o Bluetooth. Nello specifico, per controllare lo stato della batteria, si è utilizzato anche lo strumento Android Virtual Device (AVD) fornito dall'IDE Android Studio, il quale consente di creare un dispositivo virtuale e di poter simulare le proprietà della batteria.

Un problema che non si è potuto risolvere riguarda la rilevazione dello stato degli auricolari con fili sul dispositivo Vuzix Blade. Nello specifico, considerando i veloci aggiornamenti di Android che introducono nuovi metodi ma contrassegnano anche vari metodi come deprecati, si è cercato di trovare una soluzione che funzioni anche per la versione di Android installata sugli smartglasses, ovvero Android 5.1.1. Nonostante questo, lo smartglasses non riceve messaggi dell'auricolare dal sistema, mentre provando la stessa applicazione con altri due smartphone, in particolare una con versione Android 7.0 e l'altra Android 10, funzionano correttamente. È possibile che, essendo una versione modificata di Android, la casa produttrice abbia scelto di non implementare alcune API del framework. Questo non significa che una soluzione a questo problema non possa essere trovata con un eventuale sviluppo futuro.

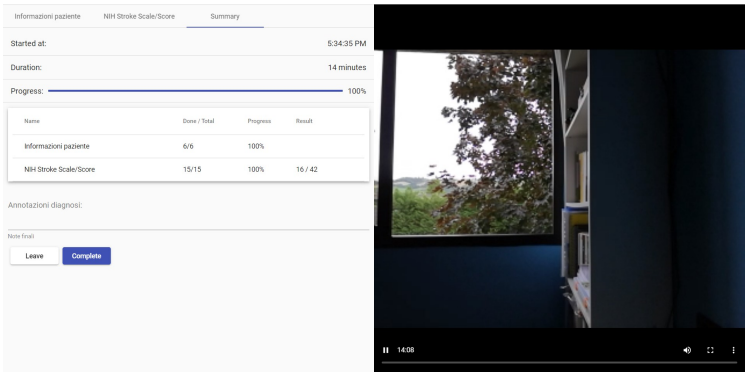
Si è anche testato il funzionamento degli elementi mostrati nella schermata durante una sessione, testando lo scambio dei messaggi e il loro comportamento. In particolare, oltre a simulare lo svolgimento di sessioni normali, si è testato varie situazioni, elencate di seguito:

- **Interruzione della sessione e una successiva ripresa di tale sessione**, per verificare se la barra di avanzamento riprendesse dal punto in cui si è interrotto.
- **Disconnessione dalla rete**, in questo caso la sessione viene interrotta e l'applicazione ritorna nella schermata principale.
- **Modifica della risposta di uno step**, dopo la modifica di una risposta dall'applicazione web, vengono trasmessi le informazioni riguardanti lo step successivo e aggiornata la progress bar.

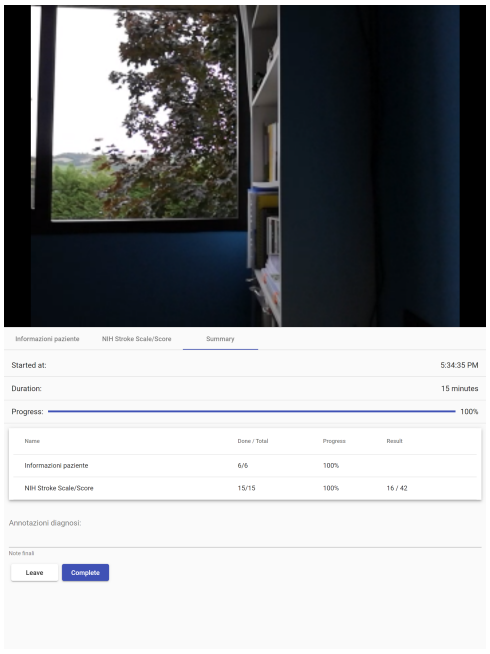
5.1.2 Frontend Angular

Per la validazione del frontend Angular, è stato creato un ambiente di testing locale tramite il comando `ng serve`. In questo modo, è possibile visualizzare l'applicazione Angular accedendo su `http://localhost:4200/`, e ogni volta che viene modificato un file sorgente, viene eseguito automaticamente la compilazione e l'aggiornamento delle modifiche, ricaricando la pagina web nel browser.

Inoltre, per testare se l'applicazione web sia resposive, sono stati sfruttati gli strumenti per lo sviluppatore messi a disposizione dal browser per simulare la visualizzazione su dispositivi con differenti dimensioni dello schermo. Nella Figura 5.1 viene mostrato la schermata di riepilogo della sessione eseguita su un PC Desktop con una risoluzione di 1920×1080 e simulata su un Tablet con risoluzione di 1024×1366.



(a) PC Desktop con una risoluzione di 1920×1080



(b) Tablet con una risoluzione di 1024×1366

Figura 5.1: Schermata di riepilogo della sessione su vari dispositivi

5.2 Considerazioni finali e possibili miglioramenti

Tutti gli errori e i problemi manifestati durante la fase di testing sono stati corretti. Conclusa la fase di sviluppo e validazione, si può affermare che i componenti sviluppati rispettano le specifiche definite nella fase di analisi.

Per quanto riguarda il frontend Android, la parte con cui si è concentrata maggiormente la tesi, allo stato attuale presenta ancora diverse sfide da affrontare legate all'hardware degli *smartglasses* Vuzix Blade come per esempio il surriscaldamento del dispositivo dopo un certo periodo di utilizzo e una limitata capacità della batteria, richiedendo una carica dopo ogni sessione.

Mentre per il software, molti aspetti si riferiscono ancora ai principi della progettazione *mobile*, anche se molte case produttrici stanno a mano a mano fornendo le proprie linee guida e librerie a supporto dello sviluppo di applicazioni per questi dispositivi. Per il refactoring dell'applicazione Android, sono state seguite le linee guida e utilizzate le librerie fornite per gli smartglasses Vuzix. Infine, sono stati rilevati alcuni possibili sviluppi futuri che potrebbero portare un ulteriore miglioramento al sistema, descritti in seguito:

- **Validazione con gli esperti del dominio**. Si potrebbe effettuare una validazione vera e propria con gli esperti del dominio, ovvero con medici e operatori sanitari, per ottenere feedback sul lavoro svolto e per verificare se è stato migliorato effettivamente l'usabilità.

- **Ripristino della sessione**. Al momento durante una sessione, se una parte si disconnette momentaneamente, a causa per esempio a problemi della rete, allora non avrà la possibilità di rientrare nella precedente sessione, ma bisogna attivare una nuova chiamata. Per cui, potrebbe essere utile aggiungere un meccanismo che consenta di riprendere la sessione.

- **Gestione delle impostazioni**. Si potrebbe aggiungere una pagina ad hoc nell'applicazione web, che consente di settare le impostazioni e generare un QRcode per farli acquisire dagli smartglasses.

- **Supporto ad assistenti virtuali**. Si potrebbe integrare funzionalità per il riconoscimento di comandi vocali, consentendo al personale sanitario di concentrarsi maggiormente alla gestione del paziente.

- **Integrazione con sistemi esistenti**. Si potrebbe integrare con sistemi esistenti della struttura sanitaria per la condivisione di dati e consentire di visualizzare informazioni storiche relative al paziente, rendendo il sistema più efficace e permettendo allo specialista di fornire una cura adatta per il paziente.

Conclusioni

Questo elaborato ha come obiettivo di proseguire il lavoro del progetto "TeleStroke", un sistema a supporto del personale medico per la valutazione della gravità dell'ictus celebrale.

Essendo l'ictus, una patologia tempo dipendente, è molto importate velocizzare i tempi per la gestione del paziente e minimizzare gli errori. Per questo si avvalgono tecnologie di telemedicina e wearable computing per supportare il lavoro dei professionisti, consentendo di connettere ospedali di zone periferiche con gli ospedali centrali specializzati. Inoltre, strumenti standardizzati come l'NIHSS possono essere in aiuto per la quantificazione della severità dell'ictus, fornendo una serie di passi per guidare i professionisti durante tutta la fase di gestione del paziente.

Per il miglioramento del progetto sono state seguite le linee guida e le indicazioni messe a disposizione per lo sviluppo di applicazioni per lo smartglasses Vuzix Blade, cercando di rendere il sistema facile da usare e usabile.

Conclusa la fase di sviluppo e di validazione, si può affermare che i requisiti definiti nella fase di analisi sono stati soddisfatti, portando ad una versione più vicina a quella finale per essere messa in funzione sul campo.

Ringraziamenti

Vorrei ringraziare innanzitutto i professori Alessandro Ricci e Angelo Croatti per avermi offerto l'opportunità di contribuire a questo progetto, molto interessante sia per l'ambito di applicazione sia per le tecnologie utilizzate. Vorrei ringraziarli anche per l'enorme disponibilità dimostrata durante tutto il percorso di questa esperienza, per i preziosi consigli forniti e per le chiare e precise risposte ai miei dubbi.

Ringrazio anche le colleghe dell'università, che mi hanno accompagnato durante questi tre anni, con le quali ho realizzato dei fantastici progetti e di aver condiviso i momenti difficili, le mie preoccupazioni ma anche momenti belli e soddisfacenti.

Un profondo ringraziamento va alla mia famiglia, che mi ha sempre sostenuta, appoggiando le mie scelte e avendomi dato l'opportunità di proseguire gli studi, senza di loro non avrei raggiunto questo traguardo. Un grazie infinite ai miei amici e parenti e a tutte quelle persone che da sempre sono stati vicini e che si sono interessati dei miei percorsi e dei miei obiettivi.

Infine, vorrei anche ringraziare me stessa, di aver trovato finalmente la strada giusta, per tutto l'impegno che ho messo in questi anni, per aver superato ogni difficoltà e soprattutto per non essermi arresa durante questo percorso.

Bibliografia

[1] Android. Documentation | android developers. `https://developer.android.com/docs`. Accessed June 2021.

[2] Angular. Angular. `https://angular.io/`. Accessed June 2021.

[3] Woow Barfield. *Fundamentals of Wearable Computers and Augmented Reality, Second Edition*. CRC Press LLC, 2015.

[4] Marie Chan, Daniel Estève, Jean-Yves Fourniols, Christophe Escriba, and Eric Campo. Smart wearable systems: Current status and future challenges. *Artificial Intelligence in Medicine*, 56(3):137–156, 2012.

[5] European Commission. Market study on telemedicine, 2018.

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

[7] Ministero della salute. Indicazioni nazionali per l'erogazione di prestazioni in telemedicina, 2020.

[8] Docker. Empowering app development for developers | docker. `https://www.docker.com/`. Accessed June 2021.

[9] WHO Global Observatory for eHealth. Telemedicine: opportunities and developments in member states: report on the second global survey on ehealth, 2010.

[10] Interaction Design Foundation. What is usability? `https://www.interaction-design.org/literature/topics/usability`. Accessed May 2021.

[11] Davide Giacomini. Studio e sviluppo di un sistema di telerefertazione. Tesi di laurea (laurea magistrale), Università di Bologna, 2020.

[12] Google. Design guidelines. `https://developers.google.com/glass-enterprise/guides/design-guidelines`. Accessed May 2021.

[13] Google. Principles. `https://developers.google.com/glass/design/principles`. Accessed May 2021.

[14] Smita Jhajharia, S Pal, and Seema Verma. Wearable computing and its application. *(IJCSIT) International Journal of Computer Science and Information Technologies*, 5:5700–5704, 07 2014.

[15] Kotlin. Kotlin programming language. `https://kotlinlang.org/`. Accessed June 2021.

[16] Marc Krolczyk and Mike Telek. *Vuzix Blade™ Dev Kit User Experience (UX) Design Guidelines*. Vuzix Corporation, 2018.

[17] Lik-Hang Lee and Pan Hui. Interaction methods for smart glasses: A survey. *IEEE Access*, 6:28712–28732, 2018.

[18] Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE Trans. Information Systems*, vol. E77-D, no. 12:1321–1329, 12 1994.

[19] N. Mohamudally. *State of the Art Virtual Reality and Augmented Reality Knowhow*, chapter WaveguideType HeadMounted Display System for AR Application. IntechOpen, 2018.

[20] National Institute of Neurological Disorders National Institute of Health and Stroke. Stroke scale. `https://www.ninds.nih.gov/sites/default/files/NIH_Stroke_Scale_Booklet.pdf`.

[21] ISO/TC 159/SC 4 Ergonomics of human-system interaction. *Ergonomic requirements for office work with visual display terminals (VDTs) – Part 1: General introduction*. 13.180 Ergonomics 35.180 IT terminal and other peripheral equipment. ISO, 1997.

[22] WHO Group Consultation on Health Telematics (1997: Geneva Switzerland). 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, 11-16 december, geneva, 1997, 1998.

[23] Thad Starner. Project glass: An extension of the self. *IEEE Pervasive Computing*, 12(2):14–16, 2013.

[24] Usabilità. `https://docs.italia.it/italia/designers-italia/design-linee-guida-docs/it/stabile/doc/user-research/usabilita.html#usabilita`. Accessed May 2021.

[25] Usability.gov. Usability evaluation basics. `https://www.usability.gov/what-and-why/usability-evaluation.html`. Accessed May 2021.

[26] Vuzix. Vuzix blade. `https://www.vuzix.com/products/blade-smart-glasses-upgraded`. Accessed June 2021.

[27] WebRTC. Webrtc. `https://webrtc.org/`. Accessed June 2021.