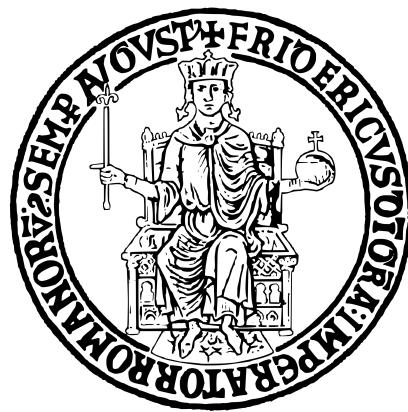


UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II



**SCUOLA POLITECNICA E DELLE SCIENZE DI
BASE**

CORSO DI LAUREA TRIENNALE IN INFORMATICA

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

TESI DI LAUREA TRIENNALE IN INFORMATICA

**HEALTHINTHECAR: SVILUPPO DI UN APPLICAZIONE ANDROID PER LA
RILEVAZIONE DEL VISO**

**HEALTHINTHECAR: DEVELOPMENT OF AN ANDROID APPLICATION FOR FACE
DETECTION**

RELATORE:

DOTT. ANDREA APICELLA

CANDIDATO:

ANTONIO D'ALTERIO

N86003547

ANNO ACCADEMICO 2022/2023

A mio nonno Antonio, che giocava sempre con me.

Prefazione

Il seguente elaborato di tesi riguarda la documentazione sullo sviluppo di un applicazione Client-Server commissionata da Teoresi S.p.A. per lo svolgimento del tirocinio.

In questo elaborato verrà descritta ed illustrata l'ideazione e l'implementazione del progetto, ponendo i riflettori sulle scelte adottate e menzionando le problematiche affrontate. In una prima parte sarà fornita un introduzione al software richiesto e sviluppato, assieme al contesto di utilizzo e tematico. In una seconda parte saranno descritti i requisiti software, ovvero i vincoli che il software deve rispettare, per poi passare ad una terza parte, che sarà invece riguardante la specifica dei requisiti, un lato più tecnico, descrivendo le scelte di design e tecnologiche adottate.

Indice

I	Introduzione	9
II	Requisiti Software	13
1	Requisiti Funzionali	16
1.1	Use Case	17
1.1.1	Calibrazione del Viso	18
1.1.2	Rilevazione del Viso	22
1.1.3	Visualizza Diagnostica	25
2	Requisiti Non Funzionali	27
I	Mockup	28
2.1	Figma	28
2.2	Palette ed altre scelte grafiche	29
2.3	Menù Scelta di Funzione	31
2.4	Calibrazione	32
2.4.1	Errore di Calibrazione	33
2.4.2	Errore di Connessione	34
2.4.3	Completamento della Calibrazione	35
2.5	Rilevazione del Viso	36
2.5.1	Errore di Connessione	37
2.6	Diagnostica	38
2.7	Impostazioni	39

III Specifica dei Requisiti e Tecnologie	40
3 Comunicazione	44
3.1 Scelta della Tecnologia di Connessione	45
3.2 La WebSocket	46
3.3 Implementazione	48
3.3.1 Lato Server	48
3.3.2 Lato Client	49
4 Server	51
4.1 Tecnologie	52
4.1.1 Python	52
4.1.2 OpenCV	54
4.1.3 NumPy	55
4.1.4 PyTorch	56
4.2 Implementazione	56
4.2.1 Il ruolo del Multiprocessing	56
4.2.2 Conversione dei Frame	58
4.2.3 Calibrazione	60
4.2.4 Rilevazione del Viso	60
5 Client	62
5.1 Tecnologie	63
5.1.1 Dart	63
5.1.2 Flutter	66
5.2 Diagrammi di Design	68
5.2.1 Class Diagram	68
5.2.2 Sequence Diagram: Calibrazione del Viso	68
5.2.3 Sequence Diagram: Rilevazione del Viso	70
5.3 Implementazione	71
5.3.1 La Navigation Bar	71
5.3.2 Camera	71

5.3.3	Calibrazione	73
5.3.4	Rilevazione	76
5.3.5	Scelta della Lingua	78
6	Valutazione delle Performance	79
6.1	Implementazione del conteggio del frame rate	80
6.2	Il Test ed i risultati	81
6.2.1	Hardware utilizzato	81
6.2.2	Calibrazione del viso	82
6.2.3	Rilevazione del viso	83
	Conclusioni e Sviluppi Futuri	85
	Bibliografia	88

Parte I

Introduzione

Il software documentato nell’elaborato di tesi qui presentato è frutto di un percorso di tirocinio svolto presso Teoresi S.p.A., il quale si è concentrato sullo sviluppo di un’applicazione client-server, che utilizzasse come parte del back-end delle funzioni di visione artificiale già testate e funzionanti di loro ideazione. Il percorso di tirocinio è iniziato con una fase di scelta del design e di architettura dell’applicativo, con particolare attenzione alla futura scalabilità e robustezza del sistema, focalizzandosi sull’offrire una comunicazione in real-time fra client e server, la quale è necessaria al corretto funzionamento delle funzioni di back-end; successivamente, è stato richiesto di passare alla progettazione grafica di un’interfaccia utente semplice, intuitiva ed efficiente, in modo da permettere un’esperienza utente ottimale.

Teoresi S.p.A.^[1] è un’azienda fondata nel 1987 che si occupa di ingegneria di alto profilo. Teoresi si preoccupa di offrire soluzioni innovative nei più disparati campi, che sia in ambito di telecomunicazioni, servizi finanziari, sistemi elettronici, veicoli industriali e agricoltura o l’industria automobilistica, la quale è l’area tematica del software in questione.

L’evoluzione del settore automobilistico va verso veicoli intelligenti, che sfruttano il progresso tecnologico per migliorare la vita dei guidatori. In questo contesto è fondamentale, fra le altre, il campo della visione artificiale¹, la quale influenza il settore per progettare veicoli con guida autonoma, migliorare la sicurezza, personalizzare l’esperienza alla guida e monitorare lo stato del conducente.



Figura 1: Logo di Teoresi S.p.A.

Teoresi ha sviluppato delle funzioni di visione artificiale contenute in un pacchetto chiamato HealthInTheCar, le quali si occupano di offrire un supporto al guidatore per

¹Sottocampo dell’IA, addestra i computer ad interpretare dati a partire da immagini, video o altre sorgenti grafiche.

monitorare il suo stato di salute ed altre informazioni utili. HealthInTheCar contiene varie funzioni, ma essendo un pacchetto in sviluppo, le funzioni principali sono due:

- La calibrazione del viso, la quale si occupa del processo di fornire al sistema tutte le informazioni riguardanti il viso del conducente che servono ad una maggiore precisione della successiva fase di rilevazione di dati a partire dal viso.
- La rilevazione del viso, la quale si occupa, previa calibrazione, di rilevare nello stato del conducente, non solo dove è poggiato il suo sguardo all'interno dell'abitacolo dell'automobile, ma anche di fornire informazioni come il suo battito cardiaco.

Come già menzionato, il progetto commissionato richiedeva lo sviluppo di un'applicazione client-server che utilizzasse, nella parte di back-end, il pacchetto di funzioni di HealthInTheCar, il quale nome è stato utilizzato anche per l'applicazione finale. Il client doveva essere un'applicazione sviluppata preferibilmente su Android che potesse connettersi al server ed usufruire dei suoi servizi.

Il software sviluppato, pertanto, è in realtà una demo² per il pacchetto di funzioni *HealthInTheCar*, in modo da visualizzarne il risultato e l'applicazione pratica a dei possibili clienti.

Il lato server è stato sviluppato in *Python*[2] [3], così come le funzioni di HealthInTheCar, in modo da facilitarne l'utilizzo ed il collegamento. Il server è costruito al fine di ricevere uno streaming video dal client, ed utilizzare lo stesso come sorgente video per le funzioni preesistenti, in quanto esse richiedono un dispositivo di I/O che gli permettano di ricevere una fonte video sulla quale effettuare le operazioni di visione artificiale, al fine di poter elaborare i dati e fornire al client i risultati di tale processamento.

Parte fondamentale dello sviluppo del server è stata la logica di conversione video, era necessario che la fonte video proveniente dal client fosse compatibile con il formato di frame video che invece le funzioni di HealthInTheCar potessero elaborare, pertanto la libreria di principale utilizzo è stata *OpenCv*[4][5], specializzata nell'elaborazione,

²Versione dimostrativa di un programma o di un insieme di funzioni a scopo dimostrativo o per una campagna pubblicitaria

manipolazione ed analisi di immagini e video; *OpenCv*, inoltre, è sviluppata principalmente in *C++*, così da usufruire della sua velocità e non essere svantaggiata dalle prestazioni di *Python*.

Lato client, invece, l'applicativo è sviluppato in *Flutter*[6][7], framework open-source sviluppato da *Google* per il linguaggio di programmazione *Dart*, specializzato nella creazione di interfacce utente tramite un'ampia gamma di widget personalizzabili e un'efficace gestione dello stato del software. Il client è sviluppato principalmente come un applicazione *Android* ma, grazie all'utilizzo di *Flutter*, è possibile utilizzarla con alcuni accorgimenti anche come applicazione web, desktop o *iOS*. Il client non solo ha il compito di inviare uno streaming video al server, ma anche di fornire all'utente un'interfaccia facile ed intuitiva per l'utilizzo; una volta che i dati dell'elaborazione sono ricevuti dal server, il client ha in aggiunta il compito di visualizzarli a schermo, cosicché l'utente ne possa vedere il risultato.

La comunicazione tra il server ed il client è stata affidata ad una *websocket*[8], la quale è un protocollo di comunicazione basato su una connessione *TCP (Transmission Control Protocol)*[9] persistente, che permette una comunicazione in real time ed in ambo i versi, adatto alla necessità di trasferire uno streaming video continuo e con poca latenza, senza la necessità di dover richiedere aggiornamenti grazie alla sua capacità di comunicazione istantanea ed asincrona.

Il software viene documentato attraverso una prima parte, relativa ai requisiti dello stesso, i quali descrivono le funzioni che deve svolgere nel dettaglio attraverso casi d'uso e diagrammi associati, oltre ad uno studio dell'interfaccia grafica, per poi passare ad un'altra parte più tecnica, nella quale si vanno a descrivere le tecnologie utilizzate, lato server, client e dal punto di vista della comunicazione, andando a specificare l'implementazione delle funzioni più rilevanti, oltre ad uno studio delle performance dell'elaborazione video.

Parte II

Requisiti Software

I requisiti software descrivono gli aspetti ed i vincoli del sistema che possono o non possono essere collegati direttamente alle sue funzionalità.

In particolare, per la descrizione dei requisiti e delle specifiche del sistema, è stato di supporto il sistema FURPS+[10], il quale descrive i seguenti aspetti:

1. **Functionality (Funzionalità)** : riguarda la descrizione di cosa deve fare il sistema; esso è trattato nel capitolo dei *Requisiti Funzionali*.
2. **Usability (Usabilità)** : descrive i requisiti che indicano il grado di comprensione di un software e la sua facilità d'uso[11]; l'aspetto è descritto nel capitolo dei *Requisiti Non Funzionali*.
3. **Reliability (Affidabilità)** : definisce la facoltà del prodotto di essere affidabile sotto alcune condizioni, lasso di tempo e numero di operazioni; l'affidabilità del prodotto è lasciata alla parte relativa alle tecnologie utilizzate, *Specifica dei Requisiti*.
4. **Performance (Prestazioni)** : specifica il grado in cui il software riesce a elaborare e produrre richieste entro i limiti di tempo designati; esso è menzionato nella sezione del *Conteggio del Frame Rate*.
5. **Supportability (Sostenibilità)** : definisce la capacità del prodotto di essere funzionante per tutto il suo ciclo vitale; per quanto concerne la sostenibilità del prodotto, essa è lasciata alla parte relativa alle tecnologie utilizzate, *Specifica dei Requisiti*.
6. + : sono le restanti categorie di requisiti software, le principali sono:
 - (a) **Design e Implementazione:** descritto nella parte relativa alla *Specifica dei Requisiti*.
 - (b) **Interfaccia:** descritto nel capitolo relativo ai *Mockup*.
 - (c) **Elementi Fisici:** menzionati nella parte relativa alla *Specifica dei Requisiti*.

Tali requisiti sono divisi in tre categorie principali: i Requisiti Funzionali, che descrivono la 'F' in FURPS, ovvero cosa deve fare il sistema nel pratico, a cosa serve;

i Requisiti Non Funzionali, che descrivono gli aspetti del sistema non direttamente collegati alle sue funzionalità; infine, la Specifica dei Requisiti, ovvero tutti i restanti, che riguardano il lato tecnico dello sviluppo relativo al prodotto.

Capitolo 1

Requisiti Funzionali

Sommario

1.1 Use Case	17
1.1.1 Calibrazione del Viso	18
1.1.2 Rilevazione del Viso	22
1.1.3 Visualizza Diagnostica	25

Per quanto concerne i Requisiti Funzionali, essi descrivono le interazioni tra il sistema e il suo ambiente, definendone il comportamento, infatti corrispondono alla forma di "Il sistema deve fare *<requisito>*". Tali requisiti, inoltre, generano una serie di casi d'uso, o *use case*[12], essi sono evidenti a seguito di un analisi delle richieste di uno *stakeholder*¹; gli *use case* individuati guidano l'architettura dell'applicativo, in quanto definiscono tutte le funzioni che devono essere sviluppate.

1.1 Use Case

In questa sezione verranno esplorati e descritti i casi d'uso dell'applicativo, tramite l'utilizzo di *Use Case Diagram* e descrizioni testuali strutturate che seguono il template di Cockburn.

Gli *Use Case Diagram* sono diagrammi utilizzati per descrivere le funzionalità di un applicativo, esse sfruttano il modello *UML*², descrivendo gli attori³ e le funzionalità stesse del sistema; non modella "come" funziona il sistema ma descrive le funzionalità del sistema.

I casi d'uso possono fare utilizzo anche di una descrizione testuale, questa può essere libera (*Casual Representation*) o far uso di template da riempire (*Fully Dressed Used Case*), come quello di Cockburn. Il template di Cockburn include una descrizione di ciò che il sistema e gli utenti si aspettino prima che inizi uno scenario, la descrizione dell'uso della funzionalità, dei suoi possibili errori e la descrizione dello stato finale del sistema al termine del caso d'uso.

¹Si intendono tutte le parti interessate nella produzione di un prodotto software, che siano i clienti o gli utenti finali.

²*Unified Modeling Language*[13], linguaggio utilizzato per la descrizione visuale della struttura di software.

³Chi interagisce col sistema, possono essere utenti o figure tecniche precise, come un database.

1.1.1 Calibrazione del Viso

Il software deve poter effettuare una calibrazione del viso, in modo che i dati e le metriche rilevate possano essere utilizzate successivamente per una rilevazione del viso quanto più precisa possibile. La calibrazione del viso consiste nel monitoraggio del viso e dello sguardo in sette punti diversi di un abitacolo di automobile: il sistema mostra a schermo una zona all'interno dell'auto, l'utente dovrà direzionare lo sguardo verso quella zona e, dopo qualche secondo, se la zona è stata correttamente registrata, viene riprodotto un sonoro di successo e si passa ad una successiva zona da visualizzare e da mappare, fino alla corretta mappatura di sette zone diverse, altrimenti, viene riprodotto un suono di errore e viene ripetuta la calibrazione per quella zona.

Di seguito lo *use case diagram* per la funzionalità:

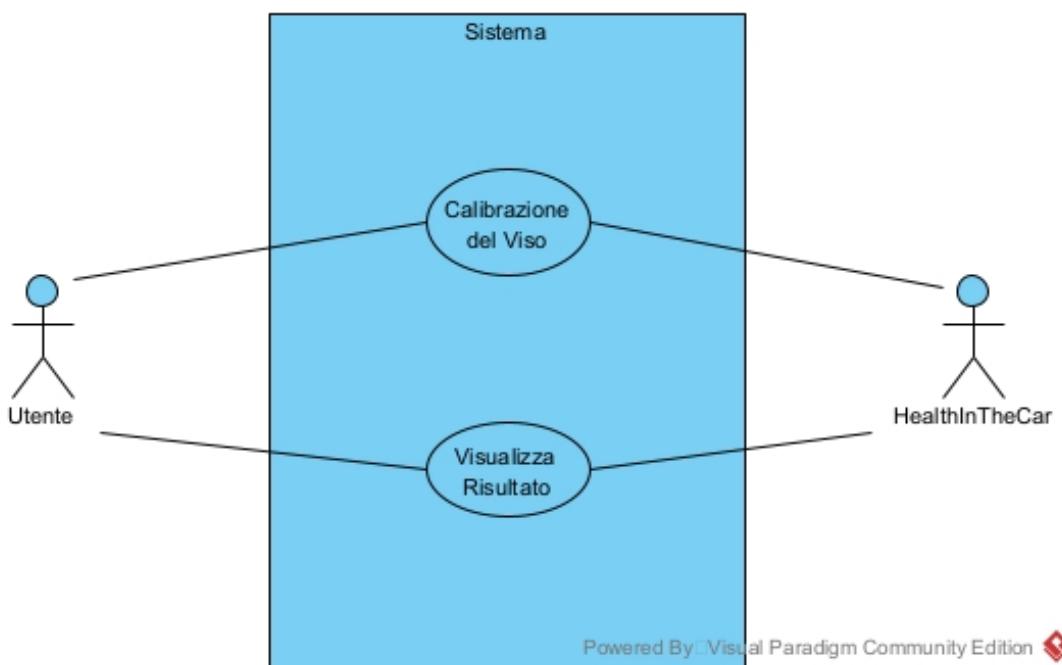


Figura 1.1: Il diagramma descrive la calibrazione del viso da parte di un utente, i dati verranno poi elaborati dal pacchetto di funzioni di HealthInTheCar, che permetterà la visualizzazione del risultato della calibrazione all'utente stesso.

Di seguito, la descrizione testuale strutturata della funzione di Calibrazione del viso tramite template di Cockburn:

USE CASE #1	Calibrazione del Viso			
Goal in Context	Calibrare il viso dell'utente per la rilevazione del viso			
Preconditions	Ci si deve trovare sulla schermata di Scelta Funzione. Bisogna essere connessi al server.			
Success End Condition	Calibrazione Effettuata con Successo			
Failed End Condition	Calibrazione non effettuata con successo, per perdita di connessione con il server.			
Primary Actor	Utente			
Trigger	L'attore clicca sul bottone di "Calibrazione" nella schermata di Scelta Funzione.			
DESCRIPTION	Step n°	Utente	Sistema	Health InTheCar
	1	Clicca sul bottone di "Calibrazione"		
	2		Mostra la schermata di Calibrazione del Viso	
	3		Inizia la calibrazione del viso mostrando la zona da visualizzare a schermo	

DESCRIPTION	Step n°	Utente	Sistema	Health InTheCar
	4	Guarda nella zona indicata		
	5		Invia lo streaming al pacchetto HealthInThe-Car	
	6			Elabora lo streaming
	7			Invia al sistema il risultato
	8		Riproduce il suono di successo	
	9		Mostra il messaggio di successo	
	10		Torna alla schermata di scelta funzione	

	Step n°	Utente	Sistema	Health InTheCar
EXTENSIONS #1 "Error 01: Errore di Calibrazione "	8.a		Riproduce il suono di insuccesso	
	9.a		Mostra a schermo "Errore nella calibrazione, ripeti la calibrazione!"	
	10.a		Ritorna allo step 3	
	Step n°	Utente	Sistema	Health InTheCar
SUBVARIATIONS #1 "Annulla"	4.b	Torna indietro con il bottone di sistema		
	5.b		Ritorna alla scelta di funzione	

Tabella 1.1: Descrizione testuale strutturata della funzione di Calibrazione del viso tramite template di Cockburn.

1.1.2 Rilevazione del Viso

Il software deve poter effettuare una rilevazione del viso, elaborando lo streaming video ed indicando a schermo qual è la zona visualizzata dall'utente con lo sguardo, oltre ad indicare il battito cardiaco dello stesso.

Di seguito lo use case diagram per la funzionalità:

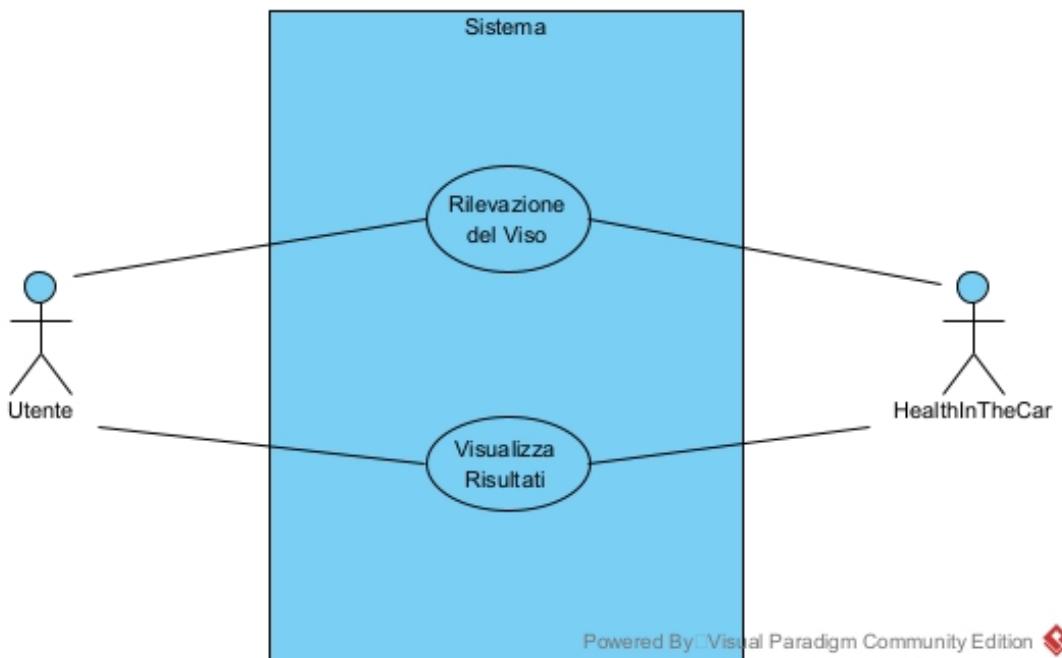


Figura 1.2: Il diagramma descrive la rilevazione del viso da parte di un utente, i dati verranno poi elaborati dal pacchetto di funzioni di HealthInTheCar, che permetterà la visualizzazione del risultato della rilevazione all'utente stesso.

Di seguito, la descrizione testuale strutturata della funzione di Rilevazione del viso tramite template di Cockburn:

USE CASE #1	Rilevazione del Viso
Goal in Context	Rilevare il viso dell'utente per mostrare in che zona si posa il suo sguardo e mostrare il suo battito cardiaco
Preconditions	Ci si deve trovare sulla schermata di Scelta Funzione. Bisogna essere connessi al server.
Success End Condition	Rilevazione Effettuata con Successo

Failed End Condition	Rilevazione non effettuata con successo, per perdita di connessione con il server.			
Primary Actor	Utente			
Trigger	L'attore clicca sul bottone di "Rileva Viso" nella schermata di Scelta Funzione.			
DESCRIPTION	Step n°	Utente	Sistema	Health InTheCar
	1	Clicca sul bottone di "Rileva Viso"		
	2		Mostra la schermata di Rilevazione del Viso	
	3		Inizia la rilevazione del viso	
	4	Posa il viso nella zona ammessa per la rilevazione dalla camera del dispositivo		
	5		Invia lo streaming al pacchetto HealthInThe-Car	
	6			Elabora lo streaming

	Step n°	Utente	Sistema	Health InTheCar
DESCRIPTION	7			Invia al sistema la zona visualizzata dall'attore e il battito cardiaco
	8		Mostra a schermo la zona visualizzata dall'utente ed il suo battito cardiaco	
EXTENSIONS #1 "Error 02: Errore di Rilevazione"	8.a		Mostra a schermo "Errore nella rilevazione!"	
	9.a		Ritorna alla scelta di funzione	
SUBVARIATIONS #1 "Annulla"	4.b	Torna indietro con il bottone di sistema		
	5.b		Ritorna alla scelta di funzione	

Tabella 1.2: Descrizione testuale strutturata della funzione di Rilevazione del viso tramite template di Cockburn.

1.1.3 Visualizza Diagnostica

Il software deve poter permettere la visualizzazione della diagnostica dell’utente, a seguito di una rilevazione del viso.

Di seguito lo use case diagram per la funzionalità:

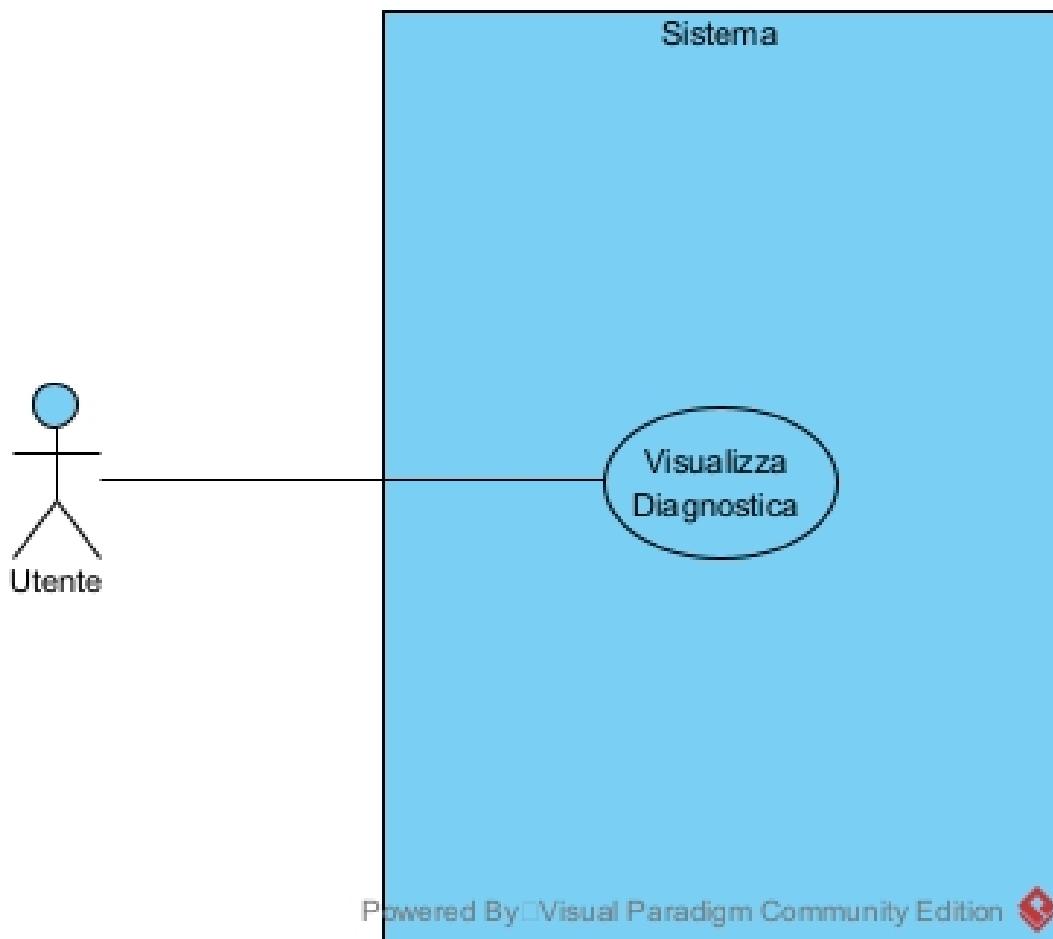


Figura 1.3: Il diagramma descrive la visualizzazione della diagnostica dell’utente, a seguito di una rilevazione del viso.

Di seguito, la descrizione testuale strutturata della funzione di visualizzazione della diagnostica tramite template di Cockburn:

USE CASE #1	Visualizza Diagnostica		
Goal in Context	Visualizzare la diagnostica delle rilevazioni dell'utente		
Preconditions	Ci si deve trovare sulla schermata di Diagnostica. Bisogna essere connessi al server.		
Success End Condition	Diagnostica caricata con successo.		
Failed End Condition	Diagnostica non caricata con successo, per perdita di connessione con il server.		
Primary Actor	Utente		
Trigger	L'attore clicca sul bottone di "Diagnostica" sulla navigation bar.		
DESCRIPTION	Step n°	Utente	Sistema
	1	Clicca sul bottone di "Diagnostica" sulla navigation bar	
	2		Mostra la schermata di diagnostica
	3		Carica i dati di diagnostica
EXTENSIONS #1 "Error 03: Errore nel caricamento della Diagnostica"	Step n°	Utente	Sistema
	3		Mostra a schermo "Errore nel caricamento della diagnostica!"

Tabella 1.3: Descrizione testuale strutturata della funzione di visualizzazione della diagnostica tramite template di Cockburn.

Capitolo 2

Requisiti Non Funzionali

Sommario

I Mockup	28
2.1 Figma	28
2.2 Palette ed altre scelte grafiche	29
2.3 Menù Scelta di Funzione	31
2.4 Calibrazione	32
2.4.1 Errore di Calibrazione	33
2.4.2 Errore di Connessione	34
2.4.3 Completamento della Calibrazione	35
2.5 Rilevazione del Viso	36
2.5.1 Errore di Connessione	37
2.6 Diagnostica	38
2.7 Impostazioni	39

I requisiti non funzionali descrivono gli aspetti del sistema che non sono collegati direttamente alle sue funzionalità, corrispondono quindi alla forma di "Il sistema deve essere *<requisito>*". A differenza dei requisiti funzionali, essi non sono obbligatori, ma fortemente consigliati per migliorare l'esperienza utente, infatti descrivono come il sistema dovrebbe comportarsi, anche a seconda dell'uso degli utenti, definendo le performance, la sicurezza ed altre proprietà.

I Mockup

In questo capitolo saranno mostrati i *mockup* dell'applicazione, ovvero una rappresentazione a scopo illustrativo delle interfacce grafiche del progetto.

I *mockup* presenti nelle seguenti pagine sono stati realizzati con lo strumento di prototipazione Figma[14].

In questa sezione saranno spiegate le interfacce grafiche dell'applicazione, insieme al perché di alcune scelte dal punto di vista del design.

2.1 Figma

Figma[14] è un tool largamente utilizzato da designer, product manager e sviluppatori, ideato ed impiegato per la creazione di prototipi di interfacce grafiche per software.

Figma permette di creare interfacce grafiche dinamiche, che riescono a riprodurre delle funzionalità del prodotto finito, in modo da fornire un possibile primo test di usabilità pre-implementazione di funzionalità di un software. Fra le varie qualità di Figma, esso permette un utilizzo in contemporanea con più utenti, in modo da condividere spazi di lavoro all'interno di un team.

Tale strumento è stato utilizzato in modo da mostrare allo *stakeholder* le interfacce grafiche ideate prima della loro implementazione, in modo da avere un riscontro an-

ticipato sulle scelte di design ed effettuare modifiche prima dello sviluppo di alcune funzionalità, risparmiando tempo nella possibile modifica di codice.

2.2 Palette ed altre scelte grafiche

Per la realizzazione del software sono state fatte alcune scelte oculate per il compito finale dell'applicativo.

I colori principali per l'applicativo sono i seguenti:

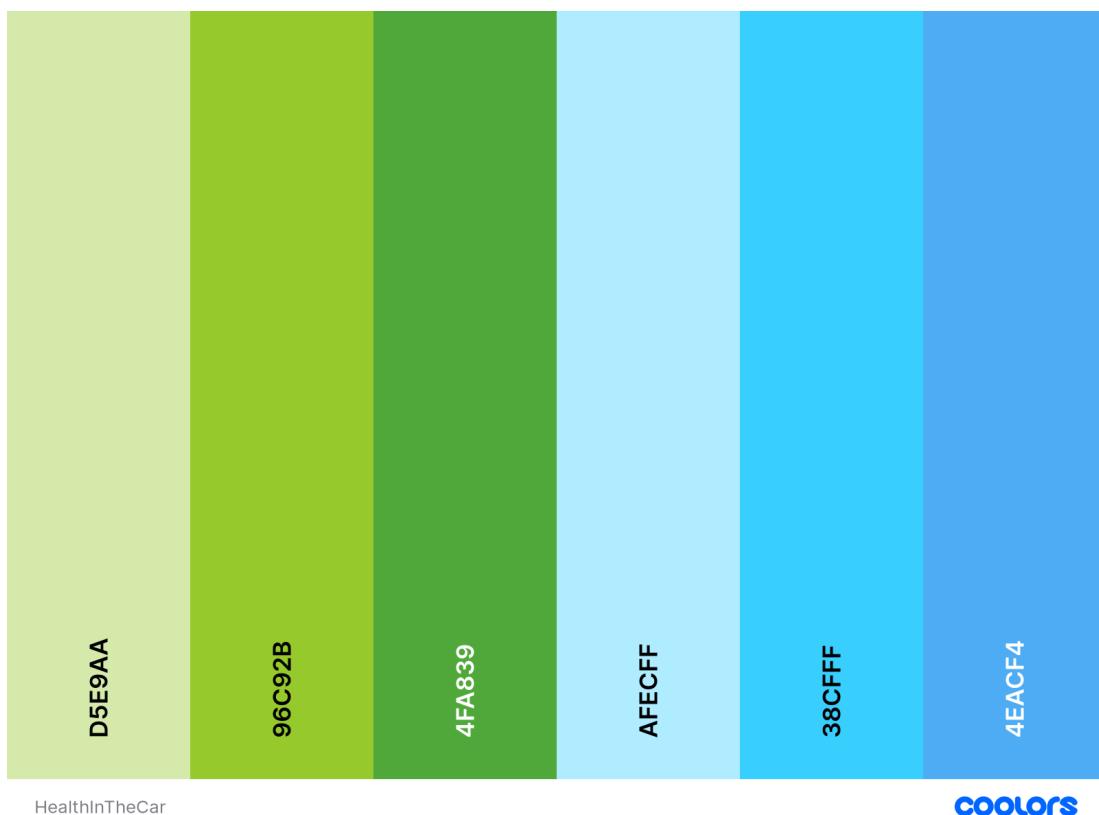


Figura 2.1: Palette principale utilizzata per l'applicazione HealthInTheCar.

I colori principali sono due, il verde ed il blu, per i quali sono stati scelti tre alternative di gradiente diverso per ognuno, in modo da utilizzarli in combinazione per rendere l'interfaccia più fresca e non stancante; la scelta di questa palette è stata fatta pensando a dei colori che si legassero bene al logo di Teoresi, oltre allo studio della color therapy[15][16], il blu tende ad essere associato a calma e a tranquillità, mentre

il verde è associato ad equilibrio ed armonia, oltre a migliorare l'umore, rilassare ed alleviare lo stress.

Per il font è stato utilizzato *Inter*[17], della famiglia di Google Fonts, il quale rende i testi dell'applicativo freschi alla vista e semplici da leggere, ciò è aiutato anche dalla distinzione del colore e delle grandezze del font, oculata per ruolo ed importanza specifica per il testo in questione.

I button sono aiutati nella distinzione per l'occhio da un'elevazione rispetto allo sfondo, oltre che da un'estetica accomunata con i bordi smussati, per renderne più facile il riconoscimento.

2.3 Menù Scelta di Funzione

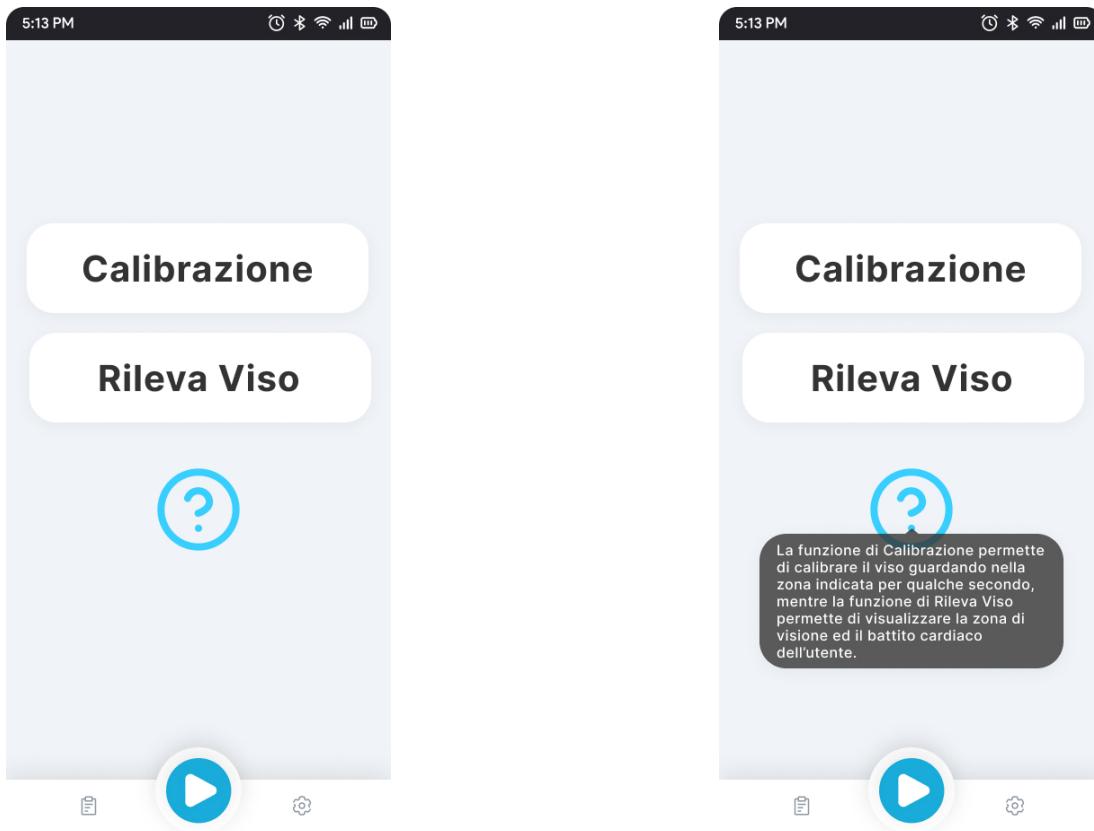


Figura 2.2: Schermate della scelta di funzione dell'applicativo, a destra è mostrato il tooltip di tutorial delle funzioni.

La schermata principale, subito dopo la *splash screen*¹ iniziale di caricamento, è quella di scelta di funzioni, in essa sono presenti i due bottoni "Calibrazione" e "Rileva Viso" che riportano alla schermata corrispondente, oltre ad un bottone di tutorial, che una volta premuto rileva un *tooltip*² che spiega come utilizzare le due funzioni.

In questa schermata, così come nelle altre è presente una *navigation bar*, che permette di cambiare fra le varie schermate dell'applicazione, per essere rinvolti alla schermata di scelta delle funzioni bisogna premere sul bottone centrale, mentre per la diagnostica il pulsante a sinistra ed infine a destra per le impostazioni.

¹Schermata iniziale di un applicazione, utilizzata tipicamente per mascherare il caricamento.

²Messaggio breve di popup che spiega il funzionamento di uno strumento.

2.4 Calibrazione

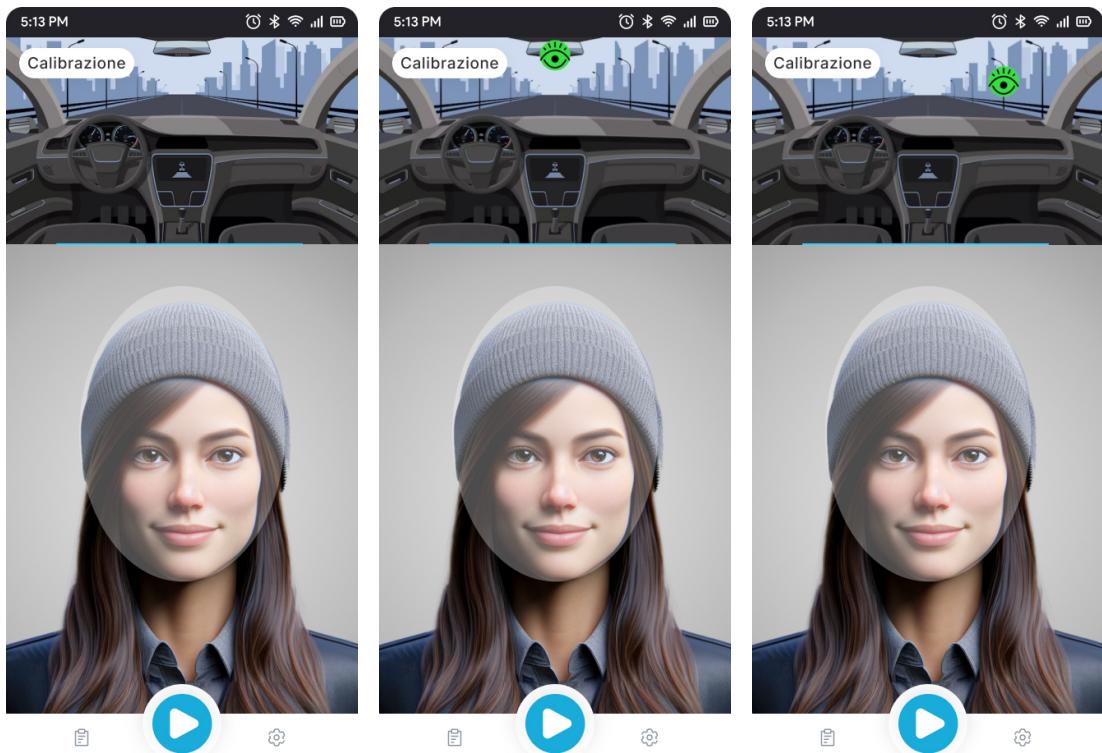


Figura 2.3: I tre *mockup* rappresentano la schermata di Calibrazione, a sinistra la schermata di default, le altre due mostrano due delle zone che l'applicativo chiede di visualizzare.

La schermata di Calibrazione si compone di una immagine superiore, la quale mostra la zona verso la quale l'utente deve volgere lo sguardo per continuare la calibrazione; nella parte sottostante è presente invece la visualizzazione della camera frontale del dispositivo dell'utente, nella quale è presente una figura in trasparenza che indica la zona nella quale l'utente deve posare il suo sguardo per permettere una corretta calibrazione del viso. La calibrazione del viso procede mostrando ogni volta una zona diversa, con un suono che indica l'inizio della calibrazione per zona ed un suono che ne indica invece il corretto completamento. Le zone mostrate sono in totale sette.

2.4.1 Errore di Calibrazione



Figura 2.4: Il *mockup* mostra la schermata di errore nella calibrazione.

Nel caso in cui ci sia un errore nella calibrazione del viso per dinamiche diverse dalla disconnessione dal server, come possono essere un non corretto posizionamento nella zona di rilevazione della camera, la chiusura degli occhi o una posizione non frontale del viso, viene riprodotto un suono di errore, oltre a venir mostrato un testo di errore, successivamente viene ripetuta la calibrazione per la zona corrispondente.

2.4.2 Errore di Connessione



Figura 2.5: La schermata mostra il messaggio di errore nel caso in cui sia interrotta la connessione con il server.

Nel caso in cui, durante la calibrazione, avvenga la disconnessione con il server, si ritorna alla schermata di scelta di funzione con la visualizzazione a schermo di un pop up che mostra il messaggio di errore.

2.4.3 Completamento della Calibrazione

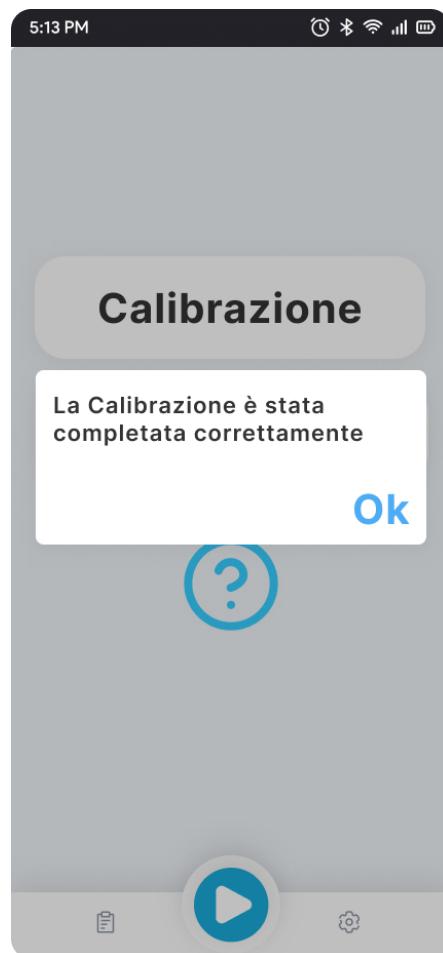


Figura 2.6: Il *mockup* mostra il messaggio di corretta fine calibrazione.

Nel caso in cui la calibrazione ha successo in tutte e sette le zone mostrate, si ritorna alla schermata di scelta di funzione, con la visualizzazione di un pop up a schermo che mostra il messaggio di corretta calibrazione.

2.5 Rilevazione del Viso

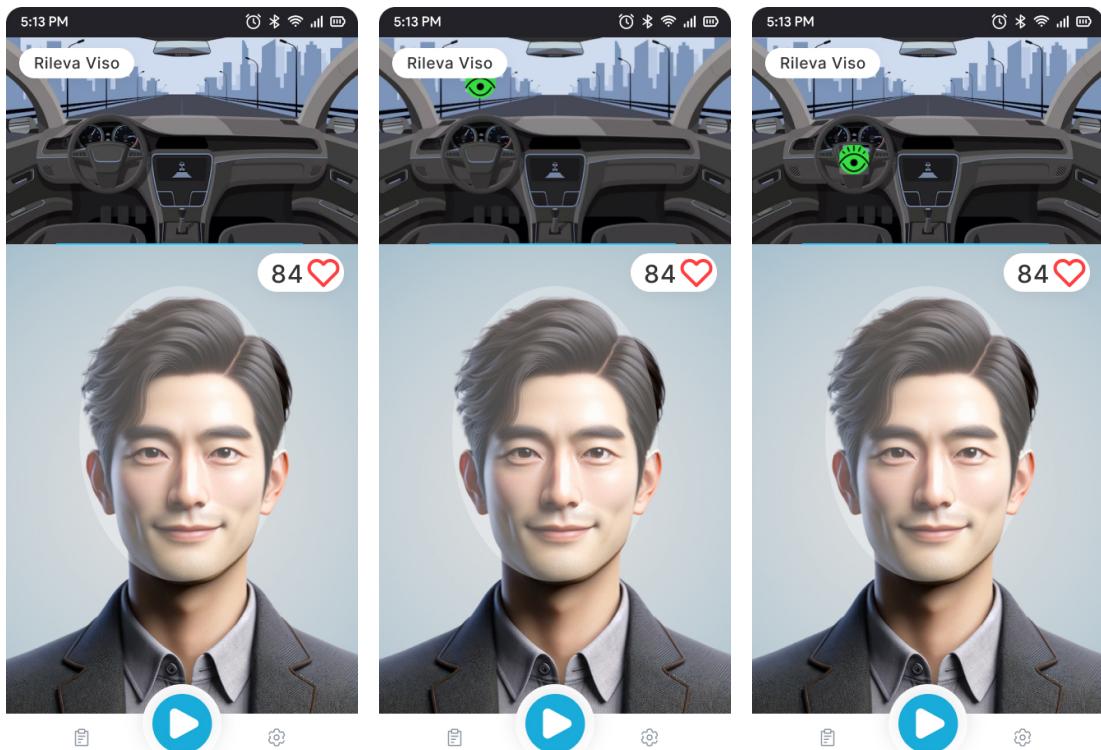


Figura 2.7: I tre *mockup* mostrano la schermata di rilevazione del viso, a sinistra la schermata di default, le altre due mostrano possibili zone di visualizzazione.

La schermata di rilevazione del viso è composta da un'immagine soprastante che indica la zona visualizzata dall'utente, essa viene modificata in tempo reale a seconda dello sguardo dell'utente. Al di sotto della zona è presente, come per la calibrazione, la visualizzazione della camera frontale del dispositivo, con una zona evidenziata che indica dove l'utente deve poggiare il suo viso per una corretta rilevazione del viso. Inoltre, a differenza della schermata di calibrazione, qui è presente un indicatore in alto a destra della camera, che indica il battito cardiaco dell'utente, questo non viene visualizzato fin da subito, ma dopo qualche secondo, in quanto il server deve prima elaborare i dati e solo dopo inviarli indietro, questo viene aggiornato con frequenza di 3 secondi.

2.5.1 Errore di Connessione

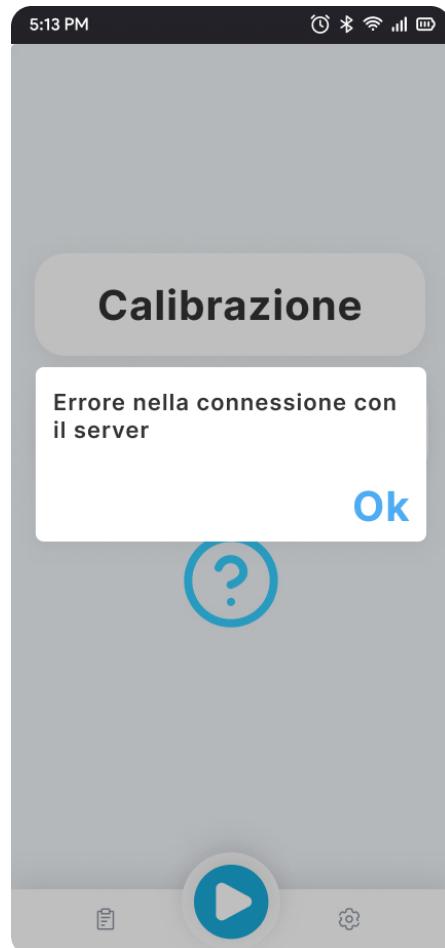


Figura 2.8: La schermata mostra il messaggio di errore nel caso in cui sia interrotta la connessione con il server.

Nel caso in cui, durante la rilevazione del viso, avvenga la disconnessione con il server, si ritorna alla schermata di scelta di funzione con la visualizzazione a schermo di un popup che mostra il messaggio di errore, così come per la calibrazione.

2.6 Diagnostica



Figura 2.9: Il *mockup* mostra la schermata di visualizzazione della diagnostica.

La schermata di diagnostica mostra alcuni dati relativi all’utente a seguito di una rilevazione del viso, questa schermata è al momento inutilizzata, in quanto il pacchetto HealthInTheCar è ancora in fase di sviluppo.

2.7 Impostazioni



Figura 2.10: La schermata di prototipo mostra le impostazioni dell'applicativo.

La schermata di impostazioni dell'applicativo mostra, al momento, un'unica opzione, quella di modifica di lingua per l'applicativo, esse sono due, l'italiano e l'inglese, la modifica avviene tramite uno switch. Inoltre, in questa schermata è presente il logo dell'azienda proprietaria ed il copyright.

Parte III

Specifiche dei Requisiti e Tecnologie

L'applicazione HealthInTheCar è un software client-server TCP-IP che utilizza funzioni di visione artificiale, sviluppato con l'utilizzo di varie tecnologie che saranno descritte nel corso di questo capitolo.

Cos'è un software Client-Server

Un'applicazione client-server^[18] è un modello architetturale che si compone di due programmi, il client ed il server, i quali sono collegati attraverso una rete, come ad esempio Internet, tramite protocolli di comunicazione come HTTP e TCP/IP.

Il server è il software che fornisce un servizio, dei dati o risorse al software client; le sue principali responsabilità includono: la gestione delle risorse, le quali possono essere file o servizi specifici, oltre che fornire un servizio di memorizzazione dei dati e l'elaborazione delle richieste del client.

Il client è l'applicazione che accede ai servizi forniti dal server; le sue principali responsabilità riguardano: l'interfaccia utente, l'inoltro di richieste al server e la gestione delle risposte del server.

Sulla visione artificiale

Nonostante sia solo negli ultimi anni che l'intelligenza artificiale abbia acquisito popolarità, grazie anche al suo incredibile progresso, mostrato ad esempio da *chatbot*³ basati su AI quali ad esempio *ChatGPT* di *OpenAI* o *Microsoft Copilot* di *Microsoft* o tecniche come il *Deep Learning*⁴, tale campo è in realtà sempre stato presente nel mondo dell'informatica, e trova le sue radici nel 1956 ad una conferenza sull'intelligenza artificiale nel Dartmouth College, stesso anno nel quale Allen Newell ed Herbert Simon presentarono al mondo il *Logic Theorist*, il primo programma in grado di effettuare pensieri logici^[19]. L'intelligenza artificiale o *IA* è lo studio di come sistemi informatici possano simulare il pensiero e l'intelligenza umana, attraverso meccanismi

³Software progettato per simulare conversazioni umane, interagendo con testo o voce.

⁴Metodo di Machine Learning, simula il funzionamento delle reti neurali cerebrali umane per l'apprendimento.

capaci di apprendere dai dati forniti e migliorarsi nel tempo. Differentemente da ciò che si pensa comunemente, l'intelligenza artificiale ha spesso un ruolo poco invasivo, ed è utilizzata ampiamente in molti settori per migliorare e facilitare la vita ed alcuni compiti dell'essere umano. Uno dei sotto campi principali dell'intelligenza artificiale è la visione artificiale, o *Computer Vision*. Da esseri umani riusciamo a distinguere il mondo esterno e la sua tridimensionalità con apparente facilità, neurofisiologi come David Hubel e Torsten Wiesel, vincitori del premio Nobel per la Medicina nel 1981 grazie alle loro scoperte sull'elaborazione delle informazioni visive nella corteccia visiva primaria[20][21], o Margaret Livingstone che collaborò allo studio di come il cervello riesca ad elaborare informazioni visive complesse come il riconoscimento facciale[22], hanno dimostrato che in realtà c'è un numero più che consistente di processi a livello biologico che avvengono nel nostro corpo per permettere al nostro cervello di distinguere ciò che percepiamo con la vista, questi stessi processi vogliono essere replicati sui calcolatori, ed ecco ciò di cui si occupa la computer vision. Con la computer vision[23][24] si vuole descrivere il mondo che ci circonda, in modo che le macchine possano guardare il mondo esterno così come lo fanno gli esseri umani, in modo da ricostruire le sue proprietà, quali ad esempio la forma, l'illuminazione o le distribuzioni di colore.

La computer vision è utilizzata nel mondo moderno per una vastità di compiti:

- **Riconoscimento Ottico dei Caratteri (OCR):** processo che converte un documento con scrittura umana in un linguaggio comprensibile da una macchina.
- **Ispezione di Macchine:** ispezione rapida di componenti di macchine tramite visione stereoscopica, in modo da controllare malfunzionamenti in automobili o velivoli.
- **Checkout automatizzato:** il riconoscimento di oggetti è impiegato nel checkout per voli o store automatizzati.
- **Settore medico:** la computer vision può offrire un supporto per diagnosi o per lo studio della morfologia del cervello umano.

- **Guida Autonoma:** può aiutare nella comprensione del mondo esterno da parte di veicoli con guida automatica.
- **Gaming:** la computer vision è stata utilizzata negli anni nel settore del gaming per implementare metodi alternativi di controllo, come il *Kinect* di *XBOX*.
- **Video Tracking:** il video tracking ha vari campi di utilizzo, come la sorveglianza del traffico o il motion capture utilizzato nel mondo del cinema.

Questi erano solo alcuni esempi dei campi per i quali la computer vision è impiegata, in generale essa è utilizzata anche per tutti quei compiti che richiedono elaborazioni di immagini, rilevamento e tracciamento di oggetti, segmentazione, analisi del movimento e della posa, oltre che ricostruzione 3D.

Capitolo 3

Comunicazione

Sommario

3.1 Scelta della Tecnologia di Connessione	45
3.2 La WebSocket	46
3.3 Implementazione	48
3.3.1 Lato Server	48
3.3.2 Lato Client	49

Per la comunicazione tra client e server si è scelto di utilizzare una *websocket*[8] tra le alternative. Una *websocket* può essere implementata sia per applicazioni Client-Server che per browser, essa è l'ideale per un applicazione che necessitano una comunicazione in real-time ed always-on, ovvero stabilisce una connessione persistente tra il client ed il server, dove entrambi i lati possono inviare dati contemporaneamente senza alcun tipo di limitazione.

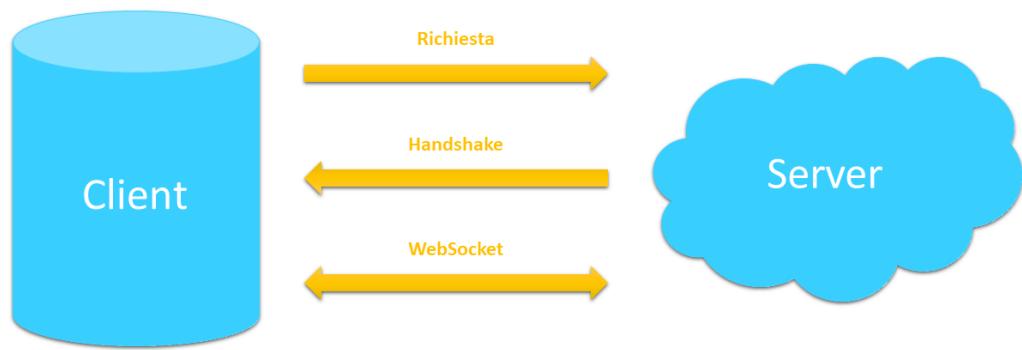


Figura 3.1: Il diagramma descrive il design di comunicazione del software.

3.1 Scelta della Tecnologia di Connessione

Durante le fasi iniziali della progettazione del software, si posero davanti tre modi diversi per la comunicazione fra client e server: una *API REST* implementata in *Flask*¹ lato server, una *socket TCP* ed una *websocket*.

Le prime due scelte presentano alcuni svantaggi per il caso specifico:

- **Socket TCP:** sono connessioni point-to-point che richiedono una creazione dedicata tra client e server, il ché può portare a problemi di scalabilità nel caso

¹Micro-framework web per l'implementazione facilitata di API REST in Python.

si vogliono rendere disponibili connessioni a partire da più client, oltre ad avere problemi di latenza e congestione di rete.

- **API REST:** permettono al client ed al server di comunicare tramite richieste HTTP, il ché le rende meno efficienti rispetto ad una *websocket*, in quanto ad ogni frame dello streaming video dovrebbe corrispondere potenzialmente una richiesta HTTP, introducendo un *overhead*² aggiuntivo di comunicazione dovuto all'intestazione della richiesta. Inoltre, le *API REST* non offrono una comunicazione bidirezionale in tempo reale, limitando le dinamiche tra client e server.

Inoltre, entrambe le tecnologie, sono poco adatte a gestire problemi di connessione ed interruzioni di rete rispetto alla *websocket*: le *socket TCP* potrebbero richiedere una riconnessione manuale o la gestione di errori personalizzati; mentre un *API REST* potrebbe richiedere il ripristino dello stato dell'applicazione e la gestione degli errori attraverso il protocollo HTTP.

Una *websocket* permette di superare i punti critici precedentemente menzionati offrendo una connessione bidirezionale e full-duplex con una latenza ridotta, rendendo la comunicazione in real-time il suo punto forte.

3.2 La WebSocket

Le *websocket* sono un protocollo di comunicazione basato su *TCP (Transmission Control Protocol)*[9], protocollo affidabile che garantisce la consegna dei dati in ordine e senza errori; le *websocket* sono particolarmente adatte per applicazioni web che richiedono una comunicazione continua e reattiva, come streaming dati, aggiornamenti e chat in tempo reale e giochi online.

Le *websocket* più recenti utilizzano lo standard del modello *RFC6455 (hybi-17)*[25], le sue caratteristiche principali sono:

- **Connessione persistente:** una volta stabilita la connessione tra client e server tramite *websocket*, entrambi possono inviare e ricevere dati senza dover stabilire

²Sovraccarico, serie di costi aggiuntivi in termini di risorse o richieste.

nuove connessioni, a differenza delle richieste HTTP tradizionali che invece sono transazioni stateless³.

- **Comunicazione bidirezionale:** le *websocket* consentono una comunicazione bidirezionale e full-duplex, in modo che sia il client ed il server possono inviare dati simultaneamente lungo la stessa connessione, consentendo una comunicazione più reattiva ed evitando di eseguire polling⁴ lato client per ottenere aggiornamenti dal server.
- **Protocollo Basato su Frame:** i dati inviati tramite *websocket* sono contenuti in frame.
- **Handshake di connessione:** la connessione *websocket* è stabilita a seguito di un handshake fra client e server, la quale avviene tramite una richiesta HTTP a partire dal client, che viene poi confermata dal server, stabilendo la connessione.
- **Protocollo Basato su Eventi:** la connessione è basata su eventi, in modo da gestire messaggi ed aggiornamenti di stato senza dover fare polling continuo.
- **Scalabilità e Prestazioni:** rispetto alle richieste HTTP tradizionali, le *websocket* consentono una comunicazione più reattiva e fluida, le connessioni persistenti diminuiscono l'overhead di rete ed il tempo di latenza; inoltre, le *websocket* consentono una comunicazione in tempo reale efficiente e con un gran numero di client connessi.

Il modello di sicurezza della *websocket* è *origin-based*, ovvero il server limita l'accesso solo alle richieste provenienti da determinate origini, prevenendo gli attacchi da origini non autorizzate; inoltre, tale connessione utilizza un mascheramento per nascondere la trasmissione dei dati, impedendo a terze parti di intercettare ed interpretare facilmente i dati inviati tra client e server.

³Senza stato, ogni richiesta è indipendente e non vengono salvate informazioni relative alla stessa.

⁴Tecnica per ricevere informazioni da un software, il client interroga continuamente il server per controllare un possibile cambiamento di stato.

3.3 Implementazione

L'implementazione della *websocket* è stata effettuata tramite l'utilizzo della libreria *websocket* lato server in Python e del pacchetto *web_socket_channel* lato client in Flutter.

3.3.1 Lato Server

Per la realizzazione della *websocket* nel server è stata utilizzata la libreria *websocket*[26] di Python, che offre un'implementazione delle *websocket RFC6455*, per la comunicazione bidirezionale tramite una connessione *TCP*. Il codice della *websocket* lato server è stato sviluppato utilizzando come supporto le librerie *asyncio*, per la gestione della comunicazione asincrona, e di *multiprocessing*, in modo da eseguire processi paralleli per la calibrazione e la rilevazione del viso a seguito della ricezione dello streaming video.

La *websocket* lato server viene aperta dal seguente codice:

```
async def main():

    start_server = await websockets.serve(
        lambda websocket, path: receive_video(websocket),
        host="***.***.*.*",
        port=port,
        ping_timeout=None
    )

    await start_server.wait_closed()
    cv2.destroyAllWindows()
```

Il comando *await websockets.serve(...)* crea un server *websocket*, passando come argomento la funzione di *receive_video*, l'indirizzo IP dell'host e la porta su quale il server dovrebbe essere in ascolto, oltre a impostare a *None* il timeout per i messaggi di ping inviati dal server al client, indicando che non ci sarà alcun timeout. Tramite,

invece, il comando `await start_server.wait_closed()`, si aspetta la chiusura asincrona della `websocket`.

La `websocket` lato server invia dati con il comando `send()`; di seguito un esempio per l'invio del messaggio contenente il battito cardiaco dell'utente rilevato:

```
await websocket.send(heart_prediction)
```

Inoltre, il server lato `websocket` è stato progettato per essere robusto, gestendo eventuali errori durante la ricezione dello streaming video e le interruzioni di connessione. In caso di errori durante la calibrazione o la rilevazione del viso, il server invia un messaggio al client per segnalare il problema.

3.3.2 Lato Client

Per la realizzazione della `websocket` nel client è stato utilizzato il pacchetto `web_socket_channel`[27] di *Dart* che fornisce un'astrazione per la comunicazione tramite `websocket`. Il client si connette tramite *URL*⁵ e la porta del server tramite la quale stabilisce la connessione alla `websocket`; inoltre, il client gestisce vari eventi durante la comunicazione con il server, come la ricezione di messaggi di risposta o notifiche di errore.

All'interno del client la connessione tramite `websocket` è implementata come un *singleton*, design pattern che garantisce l'esistenza di una sola istanza di classe e che fornisce un punto di accesso globale a quella stessa istanza. La classe `websocket` contiene una variabile statica privata `_singleton`, che è un'istanza della stessa classe `websocket`, la quale ha un costruttore privato `_internal()` che impedisce l'istanziazione diretta della classe da parte di altro codice esterno.

```
static final WebSocket _singleton = WebSocket._internal();
```

Il metodo statico `websocket` è pertanto l'unico modo per ottenere un'istanza della connessione, grazie alla sua implementazione tramite un *factory method*.

⁵Uniform Resource Locator, indirizzo web che indica in modo univoco una risorsa su internet.

```

factory WebSocket() {
    return _singleton;
}

```

La connessione con la *websocket* avviene tramite il metodo *connect()*, all'interno del quale viene istanziato un oggetto *WebSocketChannel* tramite il metodo statico *connect()*, passando come argomento un oggetto *Uri*, ovvero l'URL del server al quale connettersi.

```
_channel = WebSocketChannel.connect(Uri.parse(_url));
```

Per quanto riguarda l'invio di dati al server, il client invia dati tramite il comando *send()*; di seguito un esempio per l'invio di un singolo frame dello streaming video:

```
_channel!.sink.add(imageData);
```

Tramite il metodo *listen()*, invece, il client verrà posto in ascolto sulla connessione, chiamando una funzione di *callback*⁶ che verrà eseguita ad ogni messaggio ricevuto. Un esempio di funzione di *callback* può essere *_handlePong()*, implementata per rendere la connessione più robusta: tramite un meccanismo di ping, dove il client manda un messaggio al server periodicamente tramite il metodo *startPing()*, farà ricevere in risposta, a meno che non ci siano problemi di connessione, un messaggio di pong dal server, che sarà gestito tramite la funzione di *callback* indicata. Di seguito un esempio di *listen* con funzione di *callback*:

```

_channel!.stream.listen((message) {
    if (message == "pong") {
        _handlePong(message);
    } else if (calibration == true) {
        _handleCalibrationImageData(message);
    } else if (detect == true) {
        _handleDetectData(message);
    }
})

```

⁶Funzione che viene passata come parametro ad un'altra funzione.

Capitolo 4

Server

Sommario

4.1	Tecnologie	52
4.1.1	Python	52
4.1.2	OpenCV	54
4.1.3	NumPy	55
4.1.4	PyTorch	56
4.2	Implementazione	56
4.2.1	Il ruolo del Multiprocessing	56
4.2.2	Conversione dei Frame	58
4.2.3	Calibrazione	60
4.2.4	Rilevazione del Viso	60

Il server di HealthInTheCar è sviluppato in *Python*, in linea con il pacchetto di funzioni rilasciate dall’azienda per la rilevazione del viso, esso riceve uno streaming video dal client ed elabora la risposta, inviando il risultato al client stesso, per la sua visualizzazione.

4.1 Tecnologie

Python è il linguaggio utilizzato per l’implementazione del pacchetto di funzioni di HealthInTheCar, pertanto è stato reso più semplice scegliere lo stesso linguaggio anche per la programmazione del server. Tra le librerie di *Python* utilizzate, le più importanti sono **PyTorch**, **OpenCV** e **Numpy**.

4.1.1 Python

Python[2][3] è un linguaggio orientato agli oggetti, debolmente tipato e adatto allo sviluppo di applicazioni di scripting o computazione numerica, oltre ad essere un linguaggio interpretato e ad alto livello, creato da Guido van Rossum e rilasciato nel 1991.

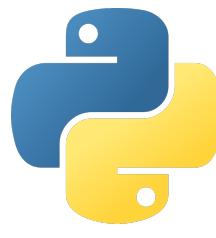


Figura 4.1: Logo di *Python*.

Le caratteristiche principali di *Python* sono:

- **Sintassi semplice ed esplicita:** *Python* è stato progettato per essere chiaro ed esplicito, in modo da facilitare la comprensione di codice e per essere un linguaggio potenzialmente *entry-level*¹, data la sua facilità d’uso.

¹Adatto ai principianti.

- **Interpretato e Interattivo:** il linguaggio è interpretato, il codice viene eseguito direttamente da un interprete senza la necessità di compilazione, evitando di aspettare per la compilazione. *Python*, inoltre, possiede un’interfaccia interattiva chiamata *REPL* (*Read-Eval-Print Loop*), che consente agli sviluppatori di eseguire istruzioni una alla volta e vederne immediatamente i risultati.
- **Tipizzazione Dinamica e Forte:** *Python* rende opzionale la dichiarazione del tipo di una variabile, esso viene determinato automaticamente in base al valore che contiene. Nonostante la sua tipizzazione dinamica, *Python* possiede anche la caratteristica di essere un linguaggio a tipizzazione forte, ovvero non vengono eseguite conversioni automatiche di tipo incoerenti e vengono sollevate eccezioni se si effettuano operazioni che non sono valide per il tipo.
- **Ampia Libreria Standard:** il linguaggio di programmazione è dotato di un ampia libreria standard, che gli offre una vasta gamma di operazioni e compiti comuni, tra cui quelli di input ed output, rendendolo meno dipendente da librerie di terze parti.
- **Multi-Paradigma:** *Python* supporta diversi paradigmi di programmazione, quali programmazione imperativa, funzionale ed orientata agli oggetti; questa flessibilità permette agli sviluppatori di adottare stili diversi a seconda della loro necessità, combinando anche diversi approcci quando necessario.

Python è utilizzato in un vasta gamma di settori ed applicazioni diverse, tra cui:

- Sviluppo web, con l’utilizzo di framework come *Django* e *Flask*;
- Data science ed analisi dei dati, con l’utilizzo di librerie come *Pandas*, *NumPy* e *SciPy*;
- Intelligenza artificiale e machine learning, con l’utilizzo di framework come *TensorFlow* e *PyTorch*;
- Automazione di sistemi e scripting;
- Applicazioni desktop e mobile, con l’utilizzo di strumenti quali *PyQt* e *Kivy*;

- Sviluppo di giochi, con librerie come *Pygame*.

Python, come già menzionato, è un linguaggio interpretato, questo lo rende particolarmente lento rispetto ad altri linguaggi di programmazione, ma allo stesso tempo, è possibile utilizzare il concetto di estensione, scrivendo parti del codice in linguaggi come *C* o *C++*, che sono invece linguaggi compilati e più veloci rispetto a *Python*, queste estensioni sono integrate tramite meccanismi come *C extension modules*, che permette di chiamare librerie scritte in *C* direttamente da *Python*, e *Cython extensions*, che aggiunge i tipi statici, compilando il codice *Python* in *C*, sfruttando la sua velocità.

4.1.2 OpenCV

OpenCV è stata una libreria fondamentale per l'applicazione HealthInTheCar, utilizzata principalmente lato server per la manipolazione dello streaming video proveniente dal Client, in modo da convertire le immagini da un formato ad un altro.

OpenCV[4][5] (Open Source Computer Vision Library) è una libreria open-source di algoritmi di *computer vision* ed elaborazione delle immagini, creata nel 1999 in Intel da Gary Bradsky. Dalla versione di *OpenCV* 2.x API, essa è scritta principalmente in *C++*, rendendola particolarmente veloce ed efficace.

Il pacchetto *OpenCV* contiene varie funzionalità, divise per sezioni: *image processing*, *video analysis*, supporto per l'elaborazione di immagini 2D, ma anche di immagini 3D per la realtà aumentata, per il rilevamento di oggetti, la calibrazione della fotocamera 3D o la ricostruzione di scene, utile in applicazioni di robotica ed altre ancora.

OpenCV supporta vari linguaggi di programmazione e sistemi operativi, essa si interfaccia con operazioni *GPU*² basate su *CUDA* e *OpenCL*, il che lo rende particolarmente efficiente ed usabile in contesti che richiedono risposte in tempo breve, come



Figura 4.2: Logo di *OpenCV*.

²Scheda grafica.

sistemi di visione per veicoli autonomi, sistemi di monitoraggio del traffico, sistemi di sorveglianza ed altro ancora. *OpenCV-Python* è la libreria di *Python* che fa da wrapper ad *OpenCV*; tutte le strutture di cui fa uso *OpenCV* in *Python*, sono convertite in array *Numpy*. *OpenCv* fa inoltre uso di altre librerie per il suo corretto funzionamento, quale ad esempio *Matplotlib* per la visualizzazione delle immagini, oltre a fornire un supporto per l'integrazione con framework di apprendimento automatico e visione artificiale come *PyTorch* e *TensorFlow*, consentendo agli sviluppatori di utilizzare modelli pre-addestrati per il riconoscimento di immagini, classificazione ed altro ancora.

4.1.3 NumPy

NumPy[28][29] è stata fondamentale come supporto ad *OpenCV*, oltre a fornire numerose funzioni di calcolo numerico utilizzate per la conversione dello stream video proveniente dal client.

NumPy è una delle librerie open-source di *Python* più importanti per la computazione scientifica e numerica.

Il nucleo di *NumPy* è rappresentato dalla struttura dati *array*, i quali sono simili alle liste di *Python*, ma ottimizzati in modo da contenere elementi di tipo omogeneo, rendendoli ideali per rappresentare dati numerici come vettori, matrici e tensori. *NumPy* permette di vettorializzare le operazioni su array senza dover esprimere cicli esplicativi, oltre ad avere la funzionalità di *broadcasting*, che gli permette di eseguire operazioni tra array di dimensioni diverse in maniera automatica. *NumPy* è inoltre scritta in *C* e *C++*, che rende il suo utilizzo efficiente in termini di velocità di esecuzione ed ottimizzazione della memoria, in quanto i suoi algoritmi utilizzano le caratteristiche hardware del sistema, quali ad esempio la parallelizzazione e la *cache*³ della CPU⁴.



Figura 4.3: Logo di *NumPy*.

³Memoria molto veloce e costosa.

⁴*Central Processing Unit*, il processore di un calcolatore elettronico, rappresenta il componente principale di un computer.

4.1.4 PyTorch

PyTorch[30] è stata la libreria fondamentale sulla quale sono state eseguite la maggior parte delle operazioni di *computer vision* del pacchetto di funzioni di HealthInTheCar.

PyTorch è una libreria open-source per l'apprendimento automatico ed il *deep learning* ideata e sviluppata da *Facebook's AI Research lab (FAIR)*.

Il nucleo di *PyTorch* è il modulo *torch*, che fornisce un implementazione efficiente di tensori, i quali sono simili agli array di *Numpy*, ma con la particolarità di essere ottimizzati per l'uso con la *GPU* per accelerare il calcolo, infatti *PyTorch* offre un supporto nativo per l'elaborazione su *GPU*. A differenza di librerie come *TensorFlow*, *PyTorch* utilizza un modello di calcolo basato su grafico computazionale dinamico, ovvero gli permette di costruire il grafo computazionale durante l'esecuzione del codice. Inoltre, la libreria offre strumenti per convertire i modelli addestrati in formati che sono compatibili per dispositivi mobile ed *embedded*⁵, consentendo inferenze sui dispositivi locali piuttosto che utilizzare la connessione ad Internet.



Figura 4.4: Logo di PyTorch.

4.2 Implementazione

In questa sezione sarà dettagliata l'implementazione del server, escludendo il ruolo della *websocket* e della comunicazione, la quale è già stata discussa nell'apposito capitolo, *Comunicazione*.

I dati video ricevuti dal client vengono decodificati ed elaborati all'interno di blocchi condizionali che gestiscono diverse situazioni.

4.2.1 Il ruolo del Multiprocessing

Nel codice viene utilizzato il modulo *multiprocessing* di *Python*, che viene utilizzato per eseguire processi separati in modo concorrente che possono richiedere una maggiore

⁵Sistemi informatici integrati in sistemi più grandi, eseguono specifiche funzioni e richiedono l'utilizzo di poche risorse.

potenza computazionale, cosa utile nel caso in cui si debbano effettuare operazioni di *computer visions* come nel caso dell'applicazione di HealthInTheCar.

A seconda della funzione scelta dal client, vengono avviati processi separati che gestiscono la richiesta relativa in parallelo al processo principale del server *websocket*. I processi sono principalmente due e corrispondono a due funzioni del pacchetto di HealthInTheCar: *calibration* e *detect*. Il loro avvio è dettagliato di seguito:

- ***calibration*:** il processo di calibrazione del viso viene avviato passando come argomenti due liste, *shared_queue*, e *image_change_queue*.

```
calibration_process =  
    Process(target=Calibrazione_m.calibration,  
            args=(shared_queue, image_change_queue))
```

- ***detect*:** il processo di rilevazione del viso viene avviato passando come argomenti tre liste, *shared_queue*, *zone_queue* e *heart_prediction_queue*.

```
detect_process =  
    Process(target=detect_funzionante.detect,  
            args=(shared_queue, zone_queue,  
                  heart_prediction_queue))
```

Nel codice sono presenti varie code (*queue*) che vengono condivise fra i processi, che permettono un uso degli stessi dati in modo concorrenziale. Grazie al modulo *queue* di *Python*, le code utilizzate garantiscono una comunicazione sicura e sincronizzata tra i diversi processi, eludendo a conflitti per l'accesso simultaneo ad uno stesso dato. Le code utilizzate sono le seguenti:

- ***shared_queue*:** la coda più importante, è utilizzata per la condivisione dei frame dello streaming video proveniente dal client; i frame vengono inseriti dal processo principale del server all'interno della *queue*, solo dopo che sono stati elaborati e convertiti in frame che possono essere letti dalle funzioni di *calibration* e *detect* del pacchetto di funzioni HealthInTheCar.

- ***image_change_queue***: la coda contiene l’immagine della zona attuale della calibrazione o, in alternativa, può indicare se si verifica un errore durante la calibrazione; i dati al suo interno possono essere inviati al client.
- ***zone_queue***: la coda contiene l’immagine della zona attuale della rilevazione del viso, il dato viene inviato al client in modo da essere visualizzata la zona corretta a schermo.
- ***heart_prediction_rate***: la coda contiene il valore del battito cardiaco dell’utente, il dato viene inviato al client in modo da essere visualizzato a schermo.

4.2.2 Conversione dei Frame

Il software HealthInTheCar è stato principalmente sviluppato per *Android*, il package *Flutter* che è incaricato di prendere lo stream video della camera del dispositivo *Android*, recupera i dati nel formato *YUV* (*YCbCr*), questo tipo di formato deve essere convertito in *RGB* in modo da essere correttamente letto dal pacchetto di funzioni di HealthInTheCar lato server, soprattutto per via dell’ampio uso della libreria di *OpenCV*.

La conversione dei dati inizia con la decodifica dei dati *YUV* dallo stream in entrata dal client, essi sono inizialmente contenuti in un array di byte di tipo *uint8*, questi vengono decodificati grazie alla libreria *NumPy*, tramite la seguente linea di codice:

```
yuv = np.frombuffer(data, dtype='uint8')
```

Una volta decodificato il frame in entrata, bisogna dividere le componenti del formato *YUV*: la *Y* rappresenta la luminanza dell’immagine, mentre la *U* e la *V* rappresentano le differenze di colore rispetto al verde; le componenti dello *YUV* vengono inserite in variabili differenti singolarmente. Le componenti *U* e *V*, inoltre, presentano una risoluzione dimezzata rispetto alla componente *Y*, pertanto devono essere eseguite delle operazioni di *upsampling*⁶ in modo da avere la stessa risoluzione della componente *Y*; l’*upsampling* avviene tramite una replica dei valori lungo le dimensioni appropriate.

⁶Tecnica che consiste nell’inserire punti dati aggiuntivi rispetto a quelli esistenti, in modo da aumentare la qualità o la dimensione di una immagine.

```

uv_start = width * height
y = yuv[:uv_start].reshape(height, width)
uv = yuv[uv_start:].reshape(height // 2, width)
u = uv[:, ::2]
v = uv[:, 1::2]

u = u.repeat(2, axis=0).repeat(2, axis=1)
v = v.repeat(2, axis=0).repeat(2, axis=1)

y = y.reshape((y.shape[0], y.shape[1], 1))
u = u.reshape((u.shape[0], u.shape[1], 1))
v = v.reshape((v.shape[0], v.shape[1], 1))

```

Una volta effettuato l'*upsampling*, le componenti separate vengono riunite in un unico array multidimensionale *yuv_array*, utilizzando il seguente comando:

```
yuv_array = np.concatenate((y, u, v), axis=2)
```

I valori nell'array vengono poi convertiti in *float32* per operazioni più precise e vengono normalizzati in modo che si trovino nell'intervallo corretto per il formato *YUV*: *Y* nell'intervallo [16, 235], *U* e *V* nell'intervallo [16, 240].

```

yuv_array = yuv_array.astype(np.float32)
yuv_array[:, :, 0] = yuv_array[:, :, 0]
    .clip(16, 235) - 16
yuv_array[:, :, 1:] = yuv_array[:, :, 1:]
    .clip(16, 240) - 128

```

A questo punto, viene applicata una matrice di conversione definita dalla costante *convert* per la conversione dell'immagine da *YUV* ad *RGB*. Sui valori risultanti deve essere applicata l'operazione di *clipping*, ovvero la limitazione dei valori di un immagine entro un intervallo specifico, in questo caso, l'intervallo [0, 255] per garantire che siano validi per il formato *RGB*; l'array viene riconvertito in *uint8* per una rappresentazione corretta dei valori dei pixel.

```

yuv_array = np.clip(yuv_array, 0, 255).astype(np.uint8)
rgb = np.matmul(yuv_array, convert.T)
    .clip(0, 255).astype('uint8')

```

Infine, l'immagine viene ruotata di 90° in senso antiorario utilizzando la funzione *cv2.rotate*, in modo che siano correttamente visualizzati dalle funzioni del server.

```
rgb = cv2.rotate(rgb, cv2.ROTATE_90_COUNTERCLOCKWISE)
```

A questo punto, il frame può essere correttamente inserito nella coda *shared_queue*, in modo da essere elaborata dai processi di *calibration* e di *detect*.

4.2.3 Calibrazione

Per quanto concerne la calibrazione lato server, essa inizia a seguito di un segnale arrivato dal client, il quale fa iniziare il processo di *calibration*.

Una volta che i frame provenienti dal client sono stati correttamente convertiti nel formato *RGB*, essi sono aggiunti alla coda *shared_queue*, in modo che la funzione del pacchetto HealthInTheCar li possa correttamente elaborare; una volta che l'elaborazione per la zona è stata terminata, il processo aggiunge alla coda *image_change_queue* una nuova zona per la calibrazione del viso, essa quindi permette al processo principale di entrare in un blocco *if*, all'interno del quale viene preparato ed inviato il messaggio da mandare al client riguardante la nuova zona da calibrare; in alternativa, la coda *image_change_queue* può contenere il codice per un errore, che indica di dover ripetere la calibrazione per la zona, anch'esso inviato al client in modo che sia correttamente visualizzato a schermo. A seguito dell'invio di uno dei due messaggi, per evitare problemi nella successiva zona da calibrare, viene svuotata la coda di *shared_queue*.

4.2.4 Rilevazione del Viso

Per quanto riguarda la rilevazione del viso lato server, così come per la calibrazione, essa inizia a seguito di un segnale arrivato dal client, il quale fa iniziare il processo di *detect*.

Convertiti ed aggiunti i frame correttamente nella coda *shared_queue*, i frame sono letti dalla funzione di *detect* del pacchetto di funzioni *HealthInTheCar*, la quale li elabora ed inserisce nella *zone_queue* la zona verso la quale lo sguardo dell’utente è rivolto, e nella coda *heart_prediction* il valore riguardante la frequenza cardiaca registrata. Le due code non vuote permettono al processo principale di entrare in un blocco *if*, all’interno del quale viene preparato ed inviato il messaggio da mandare al client riguardante la zona visualizzata dall’utente a la sua frequenza cardiaca, in modo da essere correttamente visualizzati a schermo. In modo da evitare di mandare più dati di quanto necessari, la zona visualizzata viene inviata al client se, e solo se, essa è diversa dalla precedente zona registrata.

Capitolo 5

Client

Sommario

5.1	Tecnologie	63
5.1.1	Dart	63
5.1.2	Flutter	66
5.2	Diagrammi di Design	68
5.2.1	Class Diagram	68
5.2.2	Sequence Diagram: Calibrazione del Viso	68
5.2.3	Sequence Diagram: Rilevazione del Viso	70
5.3	Implementazione	71
5.3.1	La Navigation Bar	71
5.3.2	Camera	71
5.3.3	Calibrazione	73
5.3.4	Rilevazione	76
5.3.5	Scelta della Lingua	78

Il client di HealthInTheCar è un applicazione sviluppata in *Flutter*, che permette la facile conversione del codice per utilizzarlo su sistemi *Android*, *iOS* e *Chrome*, tenendo conto però che l'applicazione è stata maggiormente testata su smartphone con sistema operativo *Android*.

5.1 Tecnologie

L'uso di *Flutter* è stato motivato da alcune cause, primo su tutti la richiesta di Teoresi S.p.a. di un'applicazione scalabile e che possa essere utilizzata su più dispositivi senza la riscrittura di codice totale, in quanto le uniche differenze sono nell'interfacciarsi con l'*SDK* specifico del sistema (eg: *Android*, *iOS*), inoltre, *Flutter* risulta particolarmente veloce grazie al suo metodo di compilazione, oltre ad essere scritto in *C++*, ciò permette al client di non essere il collo di bottiglia del sistema, spostando tutta la complessità al server; infine, *Flutter* permette lo sviluppo di un'interfaccia grafica fresca e semplice, in linea con la richiesta di un'app minimal e semplice da utilizzare.

5.1.1 Dart

Dart[31] è un linguaggio di programmazione open source orientato agli oggetti, ispirato a *Java* e *Kotlin*, ottimizzato per lo sviluppo lato client, per applicazioni web, mobili e desktop, che fa della sicurezza il suo punto di forza.



Figura 5.1: Logo di *Dart*.

Durante i tentativi di *Google* di migliorare lo sviluppo web, si accorsero che *JavaScript* presentava alcune carenze e limitazioni, pertanto nel 2011 presentarono per la prima volta *Dart*, che doveva colmare le lacune presenti in *JavaScript*, aiutando gli sviluppatori web. *Dart* ha subito numerose iterazioni e miglioramenti nel corso degli anni, con un focus sulla resa del codice più efficiente e sulla facilità di sviluppo.

Le caratteristiche principali di *Dart* sono:

- **Sintassi Chiara e Concisa:** *Dart* presenta una sintassi intuita e familiare, favorendo una scrittura di codice ordinata e manutenibile, in quanto ispirato a linguaggi come *Java*, *JavaScript* e *C++*, il ché lo rende particolarmente semplice da imparare.
- **Tipizzazione Statica Opzionale:** Il linguaggio presenza una flessibilità nella gestione dei tipi. *Dart* è *type safe*, ovvero utilizza un *type checking* statico, in modo da controllare se una variabile corrisponde sempre al suo tipo statico; inoltre, *Dart* è fornito di un *dynamic type* ed un controllo sui tipi a *runtime*, il ché può essere particolarmente utile per la sperimentazione dinamica del codice. La tipizzazione sia dinamica che statica permette agli sviluppatori di scrivere codice conciso e flessibile quando necessario nel primo caso, o garantire una maggiore sicurezza nella rilevazione degli errori nell seconda.
- **Null-Safety:** *Dart* è inoltre fornito di un controllo di *null-safety*, che gli permette di essere particolarmente sicuro, esso fa si che una variabile non possa essere null a meno che non si indichi che possa esserlo, trasformando possibili *runtime error* in *edit-time error*.
- **Gestione della Memoria Automatica:** Così come altri linguaggi di programmazione orientati agli oggetti come *Java*, anche *Dart* presenta un sistema di *garbage collection*¹, che libera la memoria degli oggetti non necessari e riduce il rischio di *memory leak*², in modo da aiutare gli sviluppatori nella scrittura di codice, permettendoli di concentrarsi sulla logica dello stesso piuttosto che sulla gestione manuale della memoria, migliorando l'affidabilità del software.
- **Asincronia e Await:** *Dart* supporta le operazioni asincrone, tramite parole chiavi come *await* ed *async*, consentendo agli sviluppatori di scrivere un codice reattivo che gestisce operazioni non bloccanti in modo efficiente; tale operazioni sono particolarmente utili per operazioni quali chiamate di rete, operazioni di *I/O* ed

¹Processo automatizzato per gestire la memoria dinamicamente.

²Situazione nella quale un programma utilizza memoria in maniera inefficiente, portanto ad uno spreco ed esaurimento di risorse.

altre ancora come la gestione di una *websocket*, in modo da rendere l'applicazione responsiva e fluida anche durante operazioni lunghe.

- **Ampia Libreria Standard:** Il linguaggio presenta una vasta gamma di funzionalità appartenenti alla libreria standard, in modo da rendere il linguaggio meno dipendente da librerie esterne, migliorando l'efficienza del codice.

Dart permette di compilare il codice con due tecnologie diverse:

- **Compilatore *Just-in-Time (JIT)*:** il codice *Dart* viene compilato in tempo reale durante l'esecuzione del programma, utilizzabile per il *debugging*³ di codice e per funzioni come quella di *hot reload*, che permette di applicare le modifiche al codice in tempo reale durante l'esecuzione del software, senza dover riavviare l'applicazione. Con il *JIT*, il codice sorgente viene tradotto in codice macchina mentre è l'applicazione in esecuzione sul dispositivo dell'utente.
- **Compilatore *Ahead-of-Time (AOT)*:** *Dart* viene compilato in codice nativo prima dell'esecuzione dell'applicativo; questo tipo di compilatore permette la compilazione di un linguaggio ad alto livello (come il *C++*) in un linguaggio binario dipendente invece dal sistema, facendo a meno della presenza di un interprete o di una macchina virtuale⁴; per il web, invece, il compilatore di *Dart* trasforma il codice in *JavaScript*. Questo tipo di compilazione è particolarmente utile per prestazioni ottimali e per ridurre i tempi di avvio dell'applicazione, soprattutto su dispositivi mobile e sistemi *embedded*.

Dart consente, inoltre, l'utilizzo di pacchetti di terze parti tramite il pacchetto *pub*: simile a *npm* di JavaScript o *pip* di Python, il pacchetto *pub* permette agli sviluppatori di importare ed utilizzare librerie, framework *UI*, *API* esterne ed altro ancora, facilitando lo sviluppo di applicazioni complesse.

Dart è utilizzato per lo sviluppo di una grande varietà di applicazioni, ed è proprio la sua capacità di sviluppare applicazioni *cross-platform*⁵ in maniera semplice ed

³Processo di individuazione, analisi e risoluzione di bug, problemi all'interno del codice.

⁴Software che crea un ambiente virtuale che simula una macchina fisica indipendente da quella realmente in uso.

⁵Capacità di un'applicazione di funzionare su piattaforme diverse.

efficace ad averlo reso popolare, grazie anche a *Flutter*, esso può essere utilizzato per la creazione di app *Android* ed *iOS* con una singola *codebase*⁶, inoltre permette lo sviluppo di applicazioni efficienti Desktop per *Windows*, *macOS* e *Linux*, oltre che per applicazioni web, oltre ad essere l'ideale per applicazioni di *IoT*⁷ grazie alle sue capacità di asincronia.

5.1.2 Flutter

Flutter[6][7] è un framework open-source sviluppato da *Google* per *Dart*, per lo sviluppo di applicazioni a partire da un'unica *codebase*. *Flutter* ha acquisito rapidamente notorietà e popolarità grazie alle sue capacità di creare interfacce grafiche fluide, complesse e fresche, in modo semplice e veloce per diverse piattaforme, quali ad esempio *Android*, *iOS* e *Chrome*.

Inizialmente *Flutter* era *Sky*, progetto di *Google* con l'obiettivo di semplificare lo sviluppo di app mobile *cross-platform*, fu presentato la prima volta nel 2015 e, nel corso degli anni, ci furono vari cambiamenti ed iterazioni, che lo portarono ad essere pubblicato come *Flutter* nel 2018.

Le caratteristiche principali di *Flutter* sono:

- **Widget Reattivi:** *Flutter* ha come mattoni fondanti i *widget*, elementi *UI* pre-definiti o personalizzati che compongono l'interfaccia grafica del software. I *widget* possono rispondere dinamicamente agli eventi, rendendo l'interazione con l'utente ricca e reattiva.
- **Hot Reload:** aiutato anche da *Dart*, linguaggio su cui è sviluppato il Framework di *Flutter*, è presente la funzione di *hot reload*, che permette agli sviluppatori di controllare l'effetto del loro codice in tempo reale senza dover riavviare l'applicazione, rendendo la programmazione di interfacce *UI* più efficiente.
- **Framework UI Dichiarativo:** *Flutter* possiede un approccio dichiarativo per la costruzione dell'interfaccia utente, ciò significa che gli sviluppatori descrivo-



Figura 5.2: Logo di *Flutter*.

⁶L'insieme di tutto il codice di un software.

⁷Internet of Things, si riferisce alla rete di dispositivi fisici come veicoli o elettrodomestici.

no l’aspetto dell’interfaccia a seconda dello stato dell’applicazione, spostando il focus sulla logica dell’applicazione piuttosto che sulla gestione dello stato dell’interfaccia.

- **Capacità Cross-Platform:** *Flutter* possiede un’architettura multi-piattaforma, il che consente di utilizzare una singola *codebase* per lo sviluppo di applicazioni su diverse piattaforme, riducendo lo sforzo del programmatore e definendo solo le singole differenze per le piattaforme.
- **Personalizzazione Estrema:** i *widget* permettono un alto grado di personalizzazione, in modo da creare interfacce *UI* uniche ed originali, che si adattano alle esigenze degli utenti e dei requisiti dell’applicazione stessa, senza peccare per prestazioni o fluidità.

Lengine di *Flutter* è scritto principalmente in *C++*, il ché gli permette un supporto per il *rendering* di basso livello utilizzando librerie grafiche di *Google*, oltre che interfacciarsi con *SDK*⁸ specifici per piattaforma come quelli di *Android* e *iOS* per implementare funzioni relative a file, rete, supporto a *plug-in* nativi ed altro ancora.

⁸Software Development Kit, insieme di librerie e funzioni che permettono allo sviluppatore di interfacciarsi con una specifica piattaforma.

5.2 Diagrammi di Design

Per la comprensione della struttura dell'architettura delle classi del client, e di alcune funzioni specifiche dell'applicativo, si mostreranno in questa sezione alcuni diagrammi di design software: *class diagram* e *sequence diagram*. Per la realizzazione dei diagrammi è stato utilizzato il tool *Visual Paradigm*[32].

5.2.1 Class Diagram

Viene mostrato di seguito il *class diagram* di analisi delle entità del software, non tutti i metodi, attributi e classi sono stati inseriti, ma solo quelli fondamentali alla comprensione dell'architettura dell'applicativo, preferendo la facilità di lettura.

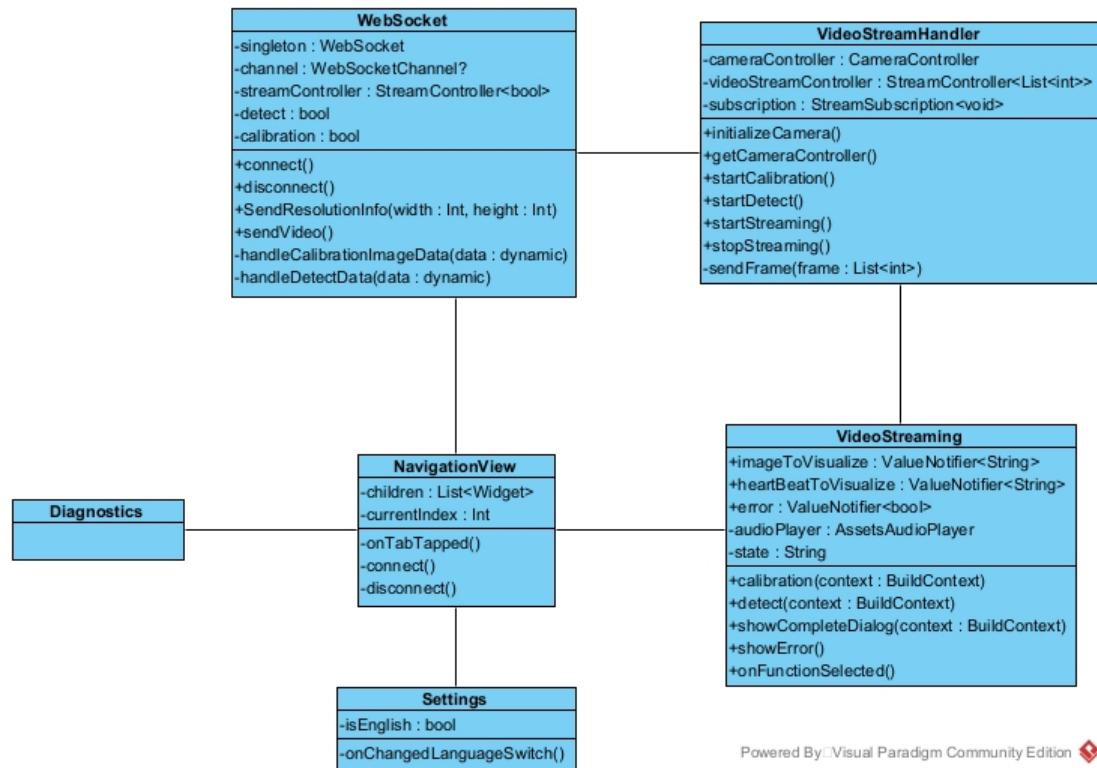


Figura 5.3: Il diagramma di design descrive le classi principali, con l'aggiunta di metodi ed attributi, del client.

5.2.2 Sequence Diagram: Calibrazione del Viso

Di seguito il *sequence diagram* di design relativo alla funzione di calibrazione del viso, con la specifica che non tutti i metodi ed attributi sono stati inseriti ma solo quelli

fondamentali alla comprensione, preferendo la facilità di lettura.

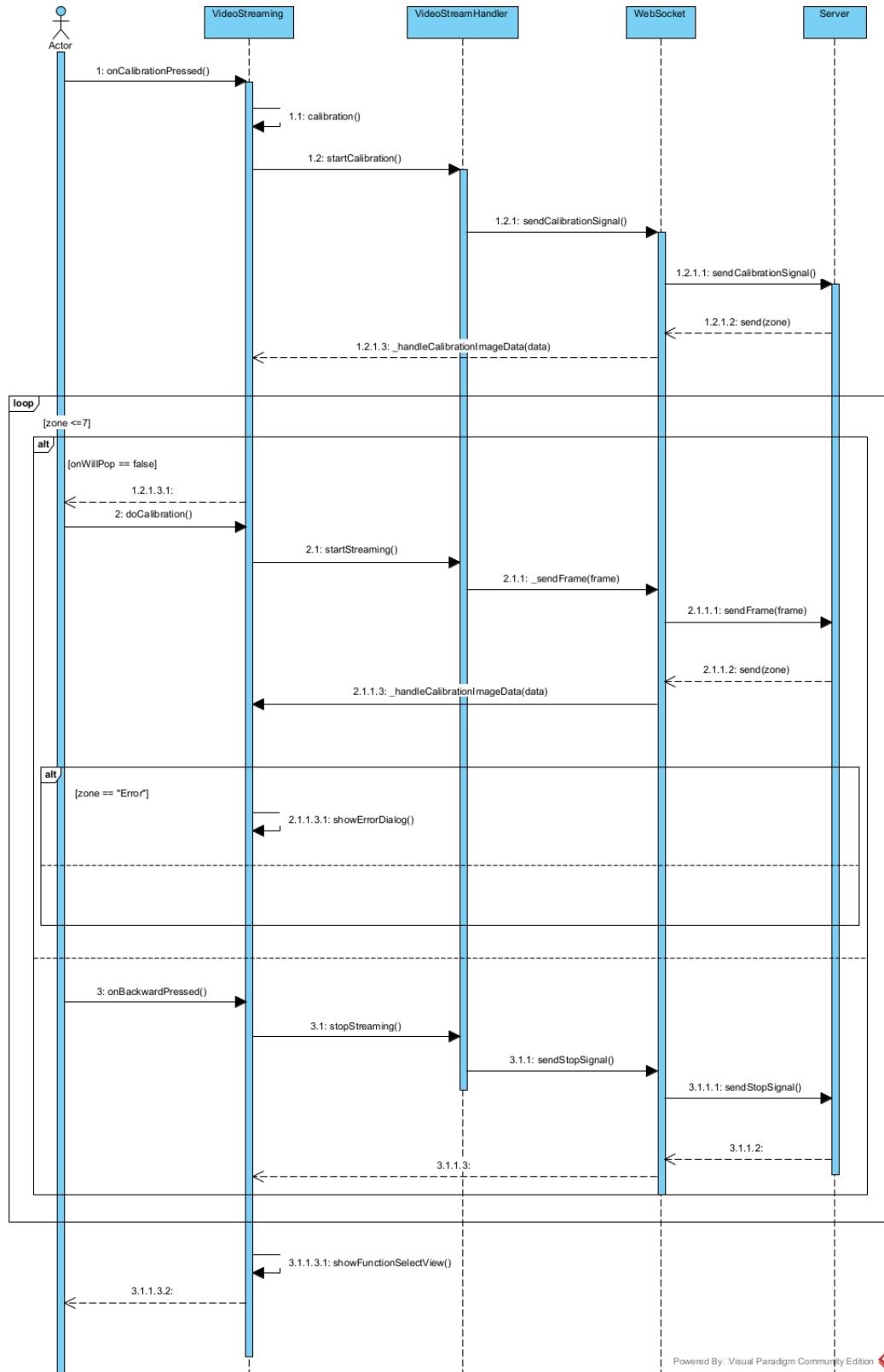


Figura 5.4: Il *sequence diagram* di design descrive la funzione di calibrazione del viso lato client.

5.2.3 Sequence Diagram: Rilevazione del Viso

Di seguito il *sequence diagram* di design relativo alla funzione di rilevazione del viso, con la specifica che non tutti i metodi ed attributi sono stati inseriti ma solo quelli fondamentali alla comprensione, preferendo la facilità di lettura.

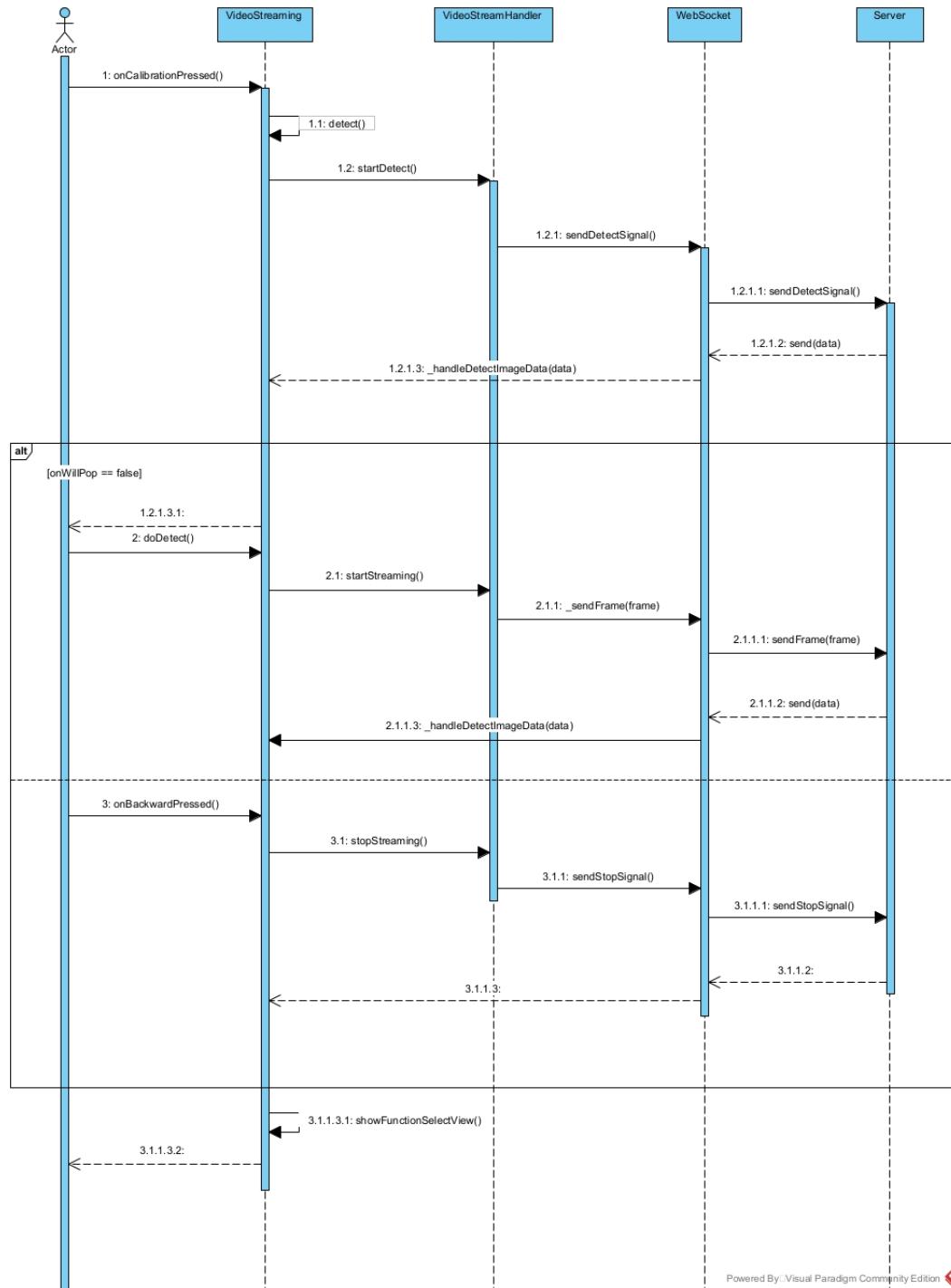


Figura 5.5: Il *sequence diagram* di design descrive la funzione di rilevazione del viso lato client.

5.3 Implementazione

In questa sezione sarà dettagliata l'implementazione del client, escludendo il ruolo della *websocket* e della comunicazione, la quale è già stata discussa nell'apposito capitolo, *Comunicazione*.

Il client è incaricato di interfacciarsi con l'utente finale, che utilizza due funzioni principali, ovvero la calibrazione e la rilevazione del viso, le quali usano la camera del dispositivo in uso per mandare al server uno streaming video che sarà elaborato per ottenere delle risposte da visualizzare a schermo.

5.3.1 La Navigation Bar

La navigation bar, o *navbar*, è un elemento fondamentale per le applicazioni mobile, che permette di navigare fra le varie schermate principali di un'applicazione; per quanto concerne HealthInTheCar, la *navbar* permette di accedere alla schermata di diagnostica, di menu principale, nel quale si può scegliere fra calibrazione e rilevazione del viso, e di impostazioni.

La *navbar* è stata implementata utilizzando il *widget* di *Flutter BottomAppBar*, composto di due *IconButton*, uno per la diagnostica e l'altro per le impostazioni, in composizione con un *FloatingActionButton*, che invece è stato posto al centro della precedente, in modo da simulare un'unica *navbar*, che invece permette l'accesso alla schermata di menu. Se la schermata visualizzata è associata all'icona appartenente alla *BottomAppBar* essa presenta un colore diverso in modo da rendere più intuitivo l'utilizzo, mentre se è stata selezionata la schermata del *FloatingActionButton* viene modificata l'icona dello stesso. Gli *IconButton* sono *widget* che permettono di creare icone cliccabili come bottoni, ognuno di essi è fornito pertanto di una funzione *onPressed*, la quale modifica la schermata da visualizzare ed aggiorna le icone della *navbar*.

5.3.2 Camera

Il ruolo della camera all'interno del software è fondamentale, il quanto è l'elemento di *I/O* che permette l'acquisizione dello streaming video che il client dovrà inviare al

server tramite la *websocket*. *Flutter* utilizza il pacchetto *camera* per interfacciarsi con la fotocamera di dispositivi *Android* ed *iOS*, permettendo di creare un'unica *codebase* per entrambi i sistemi operativi, oltre a fornire impostazioni per la personalizzazione della risoluzione, il formato dell'immagine, la modalità di messa a fuoco, l'esposizione ed altro ancora.

La camera viene inizializzata nel seguente metodo:

```
Future<void> initializeCamera() async {
    final cameras = await availableCameras();
    final frontCamera = cameras.firstWhere(
        (camera) => camera.lensDirection ==
            CameraLensDirection.front,
        orElse: () => cameras.first,
    );

    // Usa la fotocamera frontale
    _cameraController = CameraController(frontCamera,
        ResolutionPreset.low,
        imageFormatGroup: ImageFormatGroup.nv21,
        enableAudio: false);

    await _cameraController.initialize();

    // Ottieni le informazioni sulla risoluzione
    //int cameraWidth =
    //    _cameraController.value.previewSize!
    //    .width.toInt();
    //int cameraHeight =
    //    _cameraController.value.previewSize!
    //    .height.toInt();

    // Invia le informazioni sulla risoluzione al server
```

```

//_webSocket
// .sendResolutionInfo(cameraWidth, cameraHeight);
}

```

Presente nella classe *VideoStreamHandler*, esso ricerca e rende disponibile per l'utilizzo la camera frontale del dispositivo; il metodo viene chiamato nella classe *VideoStreaming* prima di usare la camera, in modo da non incappare in errori o eccezioni; inoltre, nel metodo è presente una parte di codice commentato, la quale può essere utile per inviare al server i dati riguardanti la risoluzione della camera, in modo da permettere al server di effettuare operazioni che richiedono la modifica dei frame dello streaming video, infatti nella classe *WebSocket* è presente un metodo che permette l'invio di tali dati al server di risoluzione tramite un file *JSON*⁹ se prima identificati.

Attraverso i metodi *startStreaming()* e *stopStreaming()* della classe *VideoStreamHandler*, la quale è un'interfaccia di controllo della camera del dispositivo e del suo stato di streaming utile alla classe *VideoStreaming*, fa sì che si possa iniziare lo streaming dei frame video mandandoli alla *websocket*, la quale impacchetta i dati e li manda a sua volta al server.

5.3.3 Calibrazione

Per quanto concerne la funzione di calibrazione, nella classe di *VideoStream* il processo di calibrazione viene avviato tramite la pressione del pulsante apposito nel menu iniziale. La pressione del pulsante modifica lo stato della schermata in *calibration* mostrando la schermata di calibrazione; la pressione del pulsante invia, tramite una chiamata al metodo *startCalibration()* della classe di *VideoStreamHandler*, un segnale alla *websocket*, la quale comunica al server che è iniziata la funzione di calibrazione; a seguito dell'invio di quest'ultimo, inizierà lo streaming video dal client al server. Il client riceve quindi le zone da visualizzare a schermo per la calibrazione tramite *websocket*, esse sono modificate nella schermata di *VideoStream* tramite un *ValueNotifier*, ovvero una classe di *Flutter* che permette di gestire uno stato reattivo all'interno di un'applicazione, a seconda della modifica del dato al quale è associato. Il codice del

⁹JavaScript Object Notation, linguaggio leggero e leggibile dalle macchine, utilizzato per lo scambio di messaggi tramite rete o *API*.

ValueNotifier permette di usare come *widget* un *ValueListenableBuilder*, che permette di rendere reattiva la *UI*:

```
ValueListenableBuilder(  
    valueListenable: VideoStream.imageToVisualize,  
    builder: (BuildContext context,  
              String imageToVisualize, Widget? child) {  
        return Image(  
            image: AssetImage(  
                "assets/images/Z$imageToVisualize.png"));  
    },  
)
```

Il *ValueNotifier* associato alla variabile *imageToVisualize* indica l'immagine della zona da far visualizzare all'utente. La variabile viene modificata dalla *websocket* una volta ricevuti i dati dal server:

```
void _handleCalibrationImageData(dynamic data) {  
    if (data == "Error 01") {  
        print("Error 01");  
        VideoStream.error01.value = true;  
    } else {  
        oldImg = data;  
        print("Ricevuto cambio img");  
        VideoStream.imageToVisualize.value = data;  
    }  
}
```

Gli errori invece hanno un *ValueNotifier* specifico, ad esempio:

```
VideoStream.error01.addListener(() {  
    if (VideoStream.error01.value) {  
        showError();  
    }  
})
```

```
    }  
});
```

Al codice 01 corrisponde l'errore di calibrazione, che fa ripartire la calibrazione per la zona attuale.

Una volta che una zona è stata calibrata, inoltre, viene riprodotto un suono per notificare l'utente della corretta calibrazione:

```
_audioPlayer.open(Audio(_correctAudioPath));  
_audioPlayer.play();
```

Lo stesso avviene per gli errori.

La calibrazione si definisce terminata una volta che sono state calibrate tutte le zone richieste dal server:

```
VideoStream.imageToVisualize.addListener() {  
    if (VideoStream.imageToVisualize.value == "6") {  
        disconnect();  
        showCompletedDialog(context);  
    } else {  
        showCorrect();  
    }  
};
```

Successivamente, si ritorna alla schermata di scelta di funzione e viene mostrato un *dialog* che notifica all'utente che la calibrazione è terminata con successo:

```
showDialog(  
    context: context,  
    builder: (BuildContext context) {  
        return AlertDialog(  
            title: Text('calibrationCompleted'.tr),  
            actions: [
```

```

TextButton(
    onPressed: () {
        Navigator.of(context).pop(); // Chiudi il dialog
    },
    child: const Text('Ok'),
),
],
);
},
);

```

5.3.4 Rilevazione

La classe di *VideoStream* permette l'avvio non solo della funzione di calibrazione, ma anche di rilevazione del viso tramite la pressione del pulsante apposito nel menu iniziale. Così come per la calibrazione, la pressione del pulsante modifica lo stato della schermata in *detect*:

```

setState(() {
    _state = "detect";
});

```

mostrando la schermata di rilevazione del viso; la pressione del pulsante invia, tramite una chiamata al metodo *startDetect()* della classe di *VideoStreamHandler*, un segnale alla *websocket*, la quale comunica al server che è iniziata la funzione di *detect*; a seguito dell'invio di quest'ultimo, inizierà lo streaming video dal client al server. Per ogni frame dello streaming video inviato al server, il client riceverà la zona dell'abitacolo verso la quale l'utente volgeva lo sguardo, in modo che il client possa correttamente visualizzarle a schermo, infatti esse sono mostrate nella schermata di *VideoStream* tramite un *ValueNotifier* associato alla variabile *imageToVisualize*, indicando l'immagine della zona. A differenza della calibrazione però, questa funzione permette di visualizzare anche il battito cardiaco dell'utente, che è anch'esso visualizzato e modificato tramite un *ValueNotifier*:

```

ValueListenableBuilder(
    valueListenable: VideoStream.heartBeatToVisualize,
    builder: (BuildContext context,
        String heartBeatToVisualize,
        Widget? child) {
    return Text(heartBeatToVisualize,
        style: Theme.of(context)
            .textTheme.labelMedium);
},
),

```

La variabile viene modificata dalla *websocket* una volta ricevuti i dati dal server:

```

void _handleDetectData(dynamic data) {
    if (data == "Error 02") {
        print("Error 02");
    } else if (data == "Error 03") {
        print("Error 03");
    } else if (data.toString().startsWith('Z')) {
        print("Ricevuto predizione zona");
        if (data == "Z10") {
            VideoStream.imageToVisualize.value = "ZORIGINAL";
        } else {
            VideoStream.imageToVisualize.value = data;
        }
    } else {
        print("Ricevuta Frequenza Cardiaca");
        VideoStream.heartBeatToVisualize.value = data;
    }
}

```

5.3.5 Scelta della Lingua

Per quanto concerne l'implementazione della modifica della lingua dell'applicazione, piuttosto di far sì che la lingua del software cambi a seconda della lingua del dispositivo, è stato scelto di invece utilizzare uno switch per la modifica dall'italiano all'inglese, in modo che sia facilmente utilizzabile su tutti i dispositivi. La modifica della lingua è stata facilmente implementata grazie al framework *GetX* di *Flutter*: esso permette di gestire la localizzazione dei testi in maniera semplice ed efficiente, permettendo di utilizzare un file all'interno del quale si identificano i testi da visualizzare per lingua a seconda di quella scelta. Il comando

```
Get.updateLocale(locale)
```

di *GetX*, chiamato ogni volta che viene cliccato lo switch, permette la modifica immediata della lingua dell'applicazione, che ha impatto diretto sull'interfaccia utente, ciò include i titoli, descrizioni, buttoni ed altri elementi di testo presenti nel software.

Capitolo 6

Valutazione delle Performance

Sommario

6.1	Implementazione del conteggio del frame rate	80
6.2	Il Test ed i risultati	81
6.2.1	Hardware utilizzato	81
6.2.2	Calibrazione del viso	82
6.2.3	Rilevazione del viso	83

La frequenza dei fotogrammi, o *frame rate*, è l’unità di misura per la valutazione delle prestazioni di un sistema di elaborazione o acquisizione video, ovvero la capacità di catturare fotogrammi ad una certa frequenza. Un frame rate più alto è sinonimo di maggiore fluidità nell’esperienza visiva, a differenza di un frame rate più basso che invece è associato ad una riproduzione video meno fluida, rendendo tale parametro fondamentale per l’esperienza utente.

In questo capitolo sarà mostrata una stima delle performance dell’elaborazione video lato server tramite un test che utilizza un sistema di conteggio del frame rate: si inizierà con la presentazione dell’implementazione del metodo di conteggio, per poi passare all’hardware utilizzato per la valutazione ed infine i risultati del test.

6.1 Implementazione del conteggio del frame rate

Il processo di conteggio del *frame rate* al fine di valutare le prestazioni del software è eseguito lato server in un *thread*¹ separato da quello principale, che accede alla coda condivisa dei fotogrammi, *shared_queue*, registrandone la frequenza in arrivo dal client. Il thread *frame_counter*, conta la frequenza dei frame elaborati con la frequenza di un secondo; una volta che l’intervallo di tempo è terminato, viene calcolato il *frame rate* come rapporto tra il numero di fotogrammi registrati e la durata dell’intervallo di tempo:

```
while time.time() - start_time < period:  
    try:  
        frame = shared_queue.get()  
        frame_count += 1  
        total_frames += 1  
  
    except queue.Empty:  
        pass  
  
frames_per_second = frame_count / period
```

¹Sequenza di istruzioni eseguita in parallelo all’interno di un processo.

Inoltre, l'algoritmo salva anche il valore massimo e minimo di frame rate, oltre che effettuare una media del frame rate per secondo, in modo da poter effettuare operazioni di valutazione performance.

6.2 Il Test ed i risultati

Con l'obiettivo di fornire dei dati di monitoraggio riguardanti la stima del frame rate per l'elaborazione video dell'applicativo, sono stati effettuati dei test sulle funzioni di calibrazione e di rilevazione del viso. I test in questione sono stati realizzati tramite l'utilizzo del codice di conteggio del frame rate durante un uso reale del software. In entrambi i casi, sono stati eseguiti 20 test per funzione con la finalità di fornire, per ogni esame, il valore di frame per secondo, o *fps*, minimo e massimo, oltre alla media degli *fps*, che per la calibrazione del viso prende in considerazione l'intero processo della funzione, mentre per la rilevazione prende un intervallo di tempo di 90 secondi. Nelle tabelle dei risultati, inoltre, sono presenti anche i valori riguardanti la media degli *fps* della versione originale delle funzioni di HealthInTheCar, precedenti alla modifica ed all'ottimizzazione.

6.2.1 Hardware utilizzato

Le operazioni di *computer vision* richiedono un certo dispendio di risorse, per questo motivo si ritiene sia utile passare ad una descrizione dell'hardware utilizzato, in quanto le prestazioni del software potrebbero variare a seconda dei dispositivi usati.

Per l'esecuzione del test di valutazione delle performance sono stati utilizzati due dispositivi hardware, uno per il server ed il secondo per il client.

Il dispositivo lato server è un computer con le seguenti specifiche:

- **Sistema Operativo:** Windows 11 64-bit
- **Processore:** 12th Gen Intel(R) Core(TM) i5-12400F 2.50 GHz
- **Memoria:** 16 GB 3600 Mhz DDR4

- **Scheda Grafica:** Nvidia GeForce RTX 3060 Ti 8 GB GDDR6

Il dispositivo lato client è uno smartphone Android, *OnePlus 11*, con le seguenti specifiche:

- **Sistema Operativo:** OxygenOS 14.0 Android 14
- **Processore:** Qualcomm Snapdragon 8 Gen 2 Mobile Platform
- **Memoria:** 8 GB LPDDR5X
- **Scheda Grafica:** Adreno 740

6.2.2 Calibrazione del viso

La seguente è la tabella relativa alle statistiche del frame rate della calibrazione del viso:

Test	FPS Minimi	FPS Massimi	FPS Medi	FPS Medi Prec. Vers.
1	19	28	22.6	15.43
2	17	26	24.46	12.875
3	13	27	20.193	13.92
4	15	24	19.884	15.01
5	14	23	20.21	12.94
6	16	26	22.875	15.219
7	13	25	22.857	14.34
8	14	28	23	16.023
9	17	26	21.846	15.7
10	18	27	23.09	13.56
11	13	28	22.5	13.58
12	15	23	19.49	15.2
13	15	25	21.8	14.783
14	16	26	22.01	13.293

Test	FPS Minimi	FPS Massimi	FPS Medi	FPS Medi Prec. Vers.
15	13	24	19.894	15.146
16	13	23	20.04	14.6
17	15	24	19.571	14.435
18	14	28	22.2	15.235
19	13	25	20.055	15.484
20	13	28	19.7	16.23
Media	14.8	25.7	21.41	14.65

Tabella 6.1: Tabella relativa alle statistiche del frame rate della calibrazione del viso.

Da come si può denotare dalla tabella, è stato registrato un miglioramento nel frame rate medio di **6.75** fps.

6.2.3 Rilevazione del viso

La seguente è la tabella relativa alle statistiche del frame rate della rilevazione del viso:

Test	FPS Minimi	FPS Massimi	FPS Medi	FPS Medi Prec. Vers.
1	24	32	26.6	17.54
2	22	27	24.645	16.342
3	17	31	24.368	18.5
4	19	29	26.26	16.35
5	24	28	25.485	15.98
6	21	26	24.724	16
7	21	28	24.645	17.235
8	23	32	27.857	16.87
9	22	30	24.575	16.115
10	21	29	24.6	18.23

Test	FPS Minimi	FPS Massimi	FPS Medi	FPS Medi Prec. Vers.
11	25	28	26.571	16.652
12	19	31	23.292	16.847
13	18	27	23.472	17.223
14	22	27	24.529	16.36
15	21	28	25.631	18.01
16	20	32	25.173	17.545
17	22	31	24.785	15.765
18	22	29	24.846	18.397
19	21	28	24.061	17.687
20	18	30	25.272	16.08
Media	21.1	29.15	25.06	16.986

Tabella 6.2: Tabella relativa alle statistiche del frame rate della rilevazione del viso.

Da come si può denotare dalla tabella, è stato registrato un miglioramento nel frame rate medio di **8.074** fps.

Conclusioni e Sviluppi Futuri

L'applicazione HealthInTheCar è frutto di analisi, sviluppo ed implementazione di un programma client-server commissionato da Teoresi S.p.A. per il tirocinio.

L'obiettivo del presente progetto era quello di fornire un'applicazione demo che utilizzasse delle funzioni di *computer vision* già preesistenti di proprietà dell'azienda dedite al monitoraggio dello stato del conducente alla guida, in modo che ne fosse semplice l'utilizzo da parte di un dispositivo mobile Android.

Per la realizzazione del progetto si è dovuto per prima cosa analizzare tutti i casi d'uso della stessa, esaminando nello specifico le funzioni di calibrazione e rilevazione del viso per il monitoraggio del battito cardiaco con annessa creazione di diagrammi per aiutarne la comprensibilità e lo studio, per poi passare alla progettazione di un'interfaccia grafica tramite *mockup* semplice ed intuitiva, con particolare attenzione nel rendere l'applicazione quanto più gradevole all'utilizzo possibile; si è passato, successivamente, ad effettuare uno studio sull'architettura del sistema e sui linguaggi di programmazione adatti da utilizzare per usufruirne di tutti i vantaggi, i quali sono stati *Python* lato server, scelta facilitata anche dalla questione che le funzioni di *computer vision* erano già sviluppate in tale linguaggio, oltre che dalla libreria di *OpenCV* fondamentale per la conversione dei frame video, e lato client *Flutter*, framework per la realizzazione di interfacce grafiche basato su *Dart*; inoltre, c'è stato uno studio sulla tecnologia di comunicazione giusta tra client e server che potesse fornire uno streaming video con latenza quasi nulla, ovvero una *websocket*, ed infine uno studio sulla conversione dei dati video in modo che potessero essere leggibili dalle funzioni preesistenti.

Il software risultante è un applicazione reattiva ed intuitiva, che riesce a fornire la calibrazione e la rilevazione del viso dell’utente, con annessa rilevazione del battito cardiaco, in tempi brevi, con uno streaming video elaborato lato server con un frame rate stabile e sufficiente ad un utilizzo comodo dell’applicazione finale, migliorato rispetto a precedenti iterazioni delle funzioni di back-end, come mostrato dai test effettuati.

Per quanto concerne futuri aggiornamenti, l’applicazione è stata sviluppata in maniera da essere scalabile sia lato server, grazie all’architettura impostata che permette di modificare le funzioni di *computer vision* senza intaccare lo streaming proveniente dalla *websocket* ed aggiungerne di nuove grazie alla sua modularità, che lato client, grazie all’utilizzo di *Flutter* il software può essere facilmente messo in funzione su altri dispositivi con pochi accorgimenti, oltre che fornire uno spazio dove visualizzare in futuro la diagnosi completa del conducente con dati a schermo e una possibile implementazione di un indice di sonnolenza dello stesso.

Bibliografia

- [1] Teoresi S.p.A. *Teoresi S.p.A.* URL: <https://www.teoresigroup.com/it/>.
- [2] K Jarrod Millman e Michael Aivazis. “Python for scientists and engineers”. In: *Computing in science & engineering* 13.2 (2011), pp. 9–12.
- [3] Python Software Foundation. *Python*. URL: <http://https://www.python.org/>.
- [4] Jim R Parker. *Algorithms for image processing and computer vision*. John Wiley & Sons, 2010.
- [5] OpenCV. *OpenCV*. URL: <https://opencv.org/>.
- [6] Eric Windmill. *Flutter in action*. Simon e Schuster, 2020.
- [7] Google LLC. *Flutter*. URL: <https://flutter.dev/>.
- [8] Ian Fette e Alexey Melnikov. *The websocket protocol*. Rapp. tecn. 2011.
- [9] Gary C Kessler. “An overview of TCP/IP protocols and the internet”. In: *InterNIC Document, Dec 29* (2004), p. 42.
- [10] Renata Janocová. “Evaluation of software quality”. In: *IMEA 2012* (2012), p. 24.
- [11] Roberto Polillo et al. *Facile da usare-Una moderna introduzione all’ingegneria della usabilità*. Apogeo, 2010.
- [12] Business Case. “Use Case Diagram”. In: *Glossary Diagram Details Tabs, Diagram Details Tabs Diagram Name, Property Fields For Diagram Diagram Property Fields, Property Fields For Diagram Discriminator ()*.
- [13] Hatice Koç et al. “UML diagrams in software engineering research: a systematic literature review”. In: *Proceedings*. Vol. 74. 1. MDPI. 2021, p. 13.

- [14] Figma. *Figma*. URL: <https://www.figma.com>.
- [15] Theresa-Marie Rhyne. “Applying color theory to digital media and visualization”. In: *Proceedings of the 2017 CHI conference extended abstracts on human factors in computing systems*. 2017, pp. 1264–1267.
- [16] Rakesh Gupta. “Color therapy in mental health and well being”. In: *International journal of all research education and scientific methods (IJARESM), ISSN* (2021), pp. 2455–6211.
- [17] Google Fonts. *Inter*. URL: <https://fonts.google.com/specimen/Inter>.
- [18] Haroon Shakirat Oluwatosin. “Client-server model”. In: *IOSR Journal of Computer Engineering* 16.1 (2014), pp. 67–71.
- [19] Paul H Lysaker et al. “Metacognition and schizophrenia: the capacity for self-reflectivity as a predictor for prospective assessments of work performance over six months”. In: *Schizophrenia research* 122.1-3 (2010), pp. 124–130.
- [20] David Hubel e Torsten Wiesel. “David Hubel and Torsten Wiesel”. In: *Neuron* 75.2 (2012), pp. 182–184.
- [21] David H Hubel. “Exploration of the primary visual cortex, 1955–78”. In: *Nature* 299.5883 (1982), pp. 515–524.
- [22] Margaret S Livingstone e David H Hubel. “Psychophysical evidence for separate channels for the perception of form, color, movement, and depth”. In: *Journal of Neuroscience* 7.11 (1987), pp. 3416–3468.
- [23] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.
- [24] Allen Hanson. *Computer vision systems*. Elsevier, 1978.
- [25] Ian Fette e Alexey Melnikov. *Rfc 6455: The websocket protocol*. 2011.
- [26] Aymeric Augustin e contributors. *websockets Documentation in Python*. URL: <https://websockets.readthedocs.io/en/stable/>.
- [27] Google LLC. *websocketchannel Documentation in Flutter*. URL: https://pub.dev/packages/web_socket_channel.

- [28] Eli Bressert. “SciPy and NumPy: an overview for developers”. In: (2012).
- [29] NumPy. *NumPy*. URL: <https://numpy.org/>.
- [30] The Linux Foundation. *PyTorch*. URL: <https://pytorch.org/>.
- [31] Google LLC. *Dart*. URL: <https://dart.dev/>.
- [32] Visual Paradigm. *Visual Paradigm*. URL: <https://www.visual-paradigm.com/>.

