UNIVERSITA POLITECNICA DELLE MARCHE

FACOLTA DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di un'applicazione mobile per la creazione di un ambiente di lavoro immersivo con realtà aumentata

Design and development of a mobile application for creating an immersive work environment with augmented reality

Relatore:
Prof. Storti Emanuele
Correlatrice:
Dott.ssa Marconi Sciarroni Monica

Laureando:
Marcianesi Luca

A.A 2023/2024

“C’è vero progresso solo quando i vantaggi di una nuova

Tecnologia diventano per tutti”

Henry Ford

Indice

Indice delle figure

Introduzione

Nella seconda metà dell'Ottocento nasceva l'industria e con il passare del tempo, grazie ad innumerevoli creazioni, siamo giunti all'Industria 4.0. Quest'ultima si basa sul concetto "Internet of Things" (IoT) che ha portato l'estensione di internet nell'ambito degli oggetti e dei luoghi reali. Non esiste una definizione universale di IoT ma se ne parla ogni volta che ci si riferisce all'utilizzo della rete internet, e delle sue capacità computazionali, estesa ad oggetti e sensori permettendo a quest'ultimi di generare, sfruttare e scambiare dati presenti in rete per migliorarne le prestazioni e limitare l'intervento umano non necessario. Questo è possibile costruendo una rete di sensori, formata da dispositivi elettronici capaci di ricavare dati da oggetti meccanici o dall'ambiente e comunicare tra loro attraverso internet. Il passo successivo, l'Industria 5.0, amplia il concetto di IoT in una nuova idea identificata dal nome "Internet of Everything" (IoE). Questa nuova direzione continua sulle orme della precedente ma con il nuovo obbiettivo di inserire al centro dello sviluppo l'uomo, valorizzandolo all'interno della nuova industria intelligente. Il caso di studio che verrà discusso fa parte del progetto "HOMEY: a Human-centric IoE-based Framework for Supporting the Transition Towards Industry 5.0", finanziato dall'Unione Europea – Next Generation EU tramite il bando PRIN 2022 del Ministero dell'Università e della Ricerca in cui collaborano l'Università di Pavia, l'Università Politecnica delle Marche e l'Università Bicocca di Milano. Il progetto si sviluppa dall'idea di mettere in comunicazione una rete di sensori attraverso un framework per la gestione dei dati con l'utente finale delle informazioni, presentandole attraverso un'esperienza immersiva utilizzando la realtà aumentata su strumenti smartphone, visori e permettere all'utente di interagire con il sistema a sua volta. Seguendo quest'idea, è possibile connettere l'uomo all'ambiente in cui si trova ed ai macchinari che deve utilizzare.

Essendo un progetto a cui partecipano tante entità, è utile comprendere quale siano gli obbiettivi per avere una visione generale del progetto.

Il progetto si pone tre obbiettivi:

1. Progettazione e realizzazione di un framework industriale per l'interazione tra i diversi attori coinvolti in una fabbrica.
2. Definizione e progettazione di un ambiente di lavoro immersivo sfruttando soluzioni di Realtà Aumentata.
3. Sfruttare i dati provenienti dai sensori per facilitare lo svolgimento delle mansioni, anche di gruppo, dei lavoratori.

Nella tesi verrà proposta una possibile strada per completare il secondo obbiettivo. Sarà presentato infatti un progetto che permette, attraverso la realtà aumentata, l'interazione tra l'uomo e la macchina in ambito industriale. Avendo chiaro l'obbiettivo della tesi ma anche quelli del progetto, è immediato capire che l'ambiente di lavoro immersivo dovrà comunicare con il framework del primo obbiettivo. Il progetto che verrà presentato avrà come parte centrale un'applicazione sviluppata per smartphone in grado di riconoscere un macchinario e, attraverso la realtà aumentata, mostrare all'utente i dati provenienti da un server remoto che si sostituisce al framework del progetto "HOMEY". L'utilizzo del server è un'opzione reale in quanto simula i dati prodotti da una rete IoE. Il server remoto, collocato nei server dell'Università Politecnica delle Marche, riceve attraverso la rete delle richieste che rielabora e restituisce delle informazioni.
Il presente lavoro di tesi è strutturato come di seguito: nel Capitolo 1 verrà illustrata l'analisi effettuata soffermandoci, in particolare, sul contesto in cui è stato svolto il lavoro, sugli obbiettivi da ottenere per considerare il progetto completo e sulla realtà aumentata di cui verranno discusse le potenzialità nell'ambito dello sviluppo mobile e mostrati esempi applicativi.
Nel Capitolo 2 verranno illustrati gli strumenti software utilizzati per completare gli obbiettivi prefissati soffermandosi sul come sono stati utilizzati nel progetto.
Successivamente nel Capitolo 3 verrà esposta la fase di progettazione. In questo capitolo verranno mostrate le scelte effettuate per le comunicazioni internet e per la creazione dell'applicazione. Dopo la fase di progettazione verranno presentati, nel Capitolo 4, le modalità di implementazione utilizzate per portare a compimento gli obbiettivi. La fase di implementazione è stata divisa in tre parti ognuna delle quali mostra il procedimento con

cui sono stati implementati i componenti dell'applicazione finale: server web e modello 3D e applicazione mobile.
Infine, nel capitolo delle conclusioni saranno esposte alcune considerazioni sul lavoro effettuato proponendo anche degli spunti per un possibile sviluppo futuro del progetto.

1. Analisi

In questo capitolo verrà illustrata la fase di analisi del progetto, che è importante svolgere prima di iniziare lo sviluppo del progetto poiché fondamentale per definirne gli obbiettivi e il percorso da seguire per conseguirli. Inoltre, in questo capitolo verrà anche illustrato il contesto applicativo di riferimento.

1.1. Contesto

Prima di iniziare il progetto era necessario individuare un caso di studio reale che rispettasse il contesto industriale in cui è inquadrato il progetto “HOMEY”. Questo ambiente è stato individuato nella fabbrica calzaturiera M.I.M con sede a Porto Sant’Elpidio, in provincia di Fermo. Il proprietario, dopo un breve incontro, ha acconsentito alla richiesta ed ha messo a disposizione la sua fabbrica. In Figura 1.1 viene mostrato uno dei macchinari, nello specifico un garbapunte, utilizzato per lo sviluppo dell’applicazione.

Figura 1.1 Garbapunte

Durante l’incontro il proprietario, dopo aver compreso l’obbiettivo del progetto, ha anche illustrato i vari macchinari che potevano essere adatti allo scopo. Per il progetto sono stati individuati tre macchinari, ognuno con le sue caratteristiche ed informazioni specifiche. I macchinari scelti sono un’applica puntali, una garba speroni e una garbapunte. Nel corso della tesi verranno mostrate diverse immagini illustrative che faranno riferimento proprio a quest’ultimo macchinario.

1.2. Obbiettivi

Come anticipato nell'introduzione, il progetto discusso in questo lavoro di tesi è inquadrato in un progetto più grande che si pone diversi obbiettivi. L'obbiettivo di questa tesi è definire e progettare un ambiente di lavoro immersivo sfruttando la realtà aumentata.
Dopo aver individuato il caso di studio specifico, è necessario sviluppare un'applicazione che, utilizzando la realtà aumentata, abbia le seguenti funzionalità:

1. Riconoscere uno o più oggetti: l'applicazione, attraverso la realtà aumentata, dovrà essere in grado, una volta inquadrato un macchinario, di riconoscerlo univocamente nell'ambiente.

2. Aggiungere informazioni all'oggetto specifico: dopo aver riconosciuto un macchinario, l'applicazione dovrà aggiungere sullo schermo, attraverso la realtà aumentata, informazioni aggiuntive utili all'utente dell'applicazione.

3. Ricevere le informazioni tramite internet anche in tempo reale: l'applicazione dovrà mostrare all'utente informazioni che provengono tramite internet. Queste informazioni possono essere metadati relativi ad un macchinario presenti su un server che non vengono modificati nel breve periodo oppure informazioni che vengono rilevate, attraverso dei sensori, dal macchinario e quindi devono essere aggiornate in tempo reale e mostrate all'utente.

4. Interagire con l'oggetto: l'applicazione mobile dovrà mettere a disposizione dell'utente la possibilità di interagire con il macchinario, rappresentato sullo schermo, attraverso una comunicazione bidirezionale con il server. L'interazione deve essere rilevata da un elemento mostrato con la realtà aumentata.

1.3. Realtà aumentata

Pilastro fondamentale del progetto è la realtà aumentata o AR, dall'inglese *augmented reality,* ed in questa sezione verrà approfondita questa tecnologia. La realtà aumentata è una tecnologia che è in grado di combinare il mondo reale e quello virtuale creando un'esperienza immersiva e interattiva che sovrappone elementi grafici, video o ologrammi al mondo fisico *[1]*. Questa tecnologia è ampiamente utilizzata nell'industria come strumento per interagire con gli strumenti industriali. La realtà aumentata ha i suoi natali nel 1968 quando un ricercatore statunitense di nome Ivan Sutherland la utilizzò in un lavoro in combinazione con degli occhiali *[2]*. Successivamente, nel 1992, Louis Rosenberg produsse Visual Fixtures: un sistema in grado di guidare i suoi utilizzatori nelle mansioni da svolgere attraverso delle istruzioni presentate sullo schermo.
Da allora questa tecnologia ha continuato a svilupparsi fino ad arrivare ai giorni odierni in cui viene applicata a tutti gli ambiti.
In questo elaborato viene esposto un caso di studio in cui l'ambiente applicativo è quello industriale. In particolare, la maggior parte delle applicazioni AR in ambienti industriali si focalizzano sui processi di assemblaggio manuale, manutenzione, formazione e nei controlli di qualità fornendo istruzioni di lavoro agli utenti su quali azioni compiere. Inoltre, la realtà aumentata è stata utilizzata anche per aumentare la sicurezza dei lavoratori. Infine, alcuni lavori si concentrano sull'elaborazione delle informazioni proponendo una gestione visiva dei dati utilizzando l'AR *[3]*.
Questa tecnologia permette di acquisire conoscenze, informazioni e visualizzare dati che, altrimenti, non potrebbero essere rivelate dall'uomo con precisione e senza mettere a rischio la sua sicurezza come, per esempio, la temperatura di un macchinario o il rumore interno di un motore. Gli elementi digitali inseriti utilizzando questa tecnologia possono essere di varie tipologie. Le più utilizzate sono: immagini, video e modelli 3D ma, grazie all'utilizzo della rete internet, è anche possibile l'elaborazione e visualizzazione dei dati provenienti dal mondo fisico. Allo stesso modo, è possibile aiutare l'utente a focalizzarsi sulle informazioni importanti già presenti nel mondo concreto ma immerse in un contesto con informazioni non necessarie.
Esistono molti tipo di realtà aumentata ma sono tutti riconducibili a due categorie principali: AR non basata sui marker e basata sui marker *[4]*.

AR non basata sui marker

Quando viene scansionato l'ambiente in cui si trova l'utente vengono aggiunte informazioni. A seconda del tipo di informazioni aggiunte, possiamo definire vari tipi di AR non basata sui marker:

1. **AR basata sulla localizzazione del dispositivo:** In questo caso le informazioni sono dipendenti dalla posizione del dispositivo che sta usando la realtà aumentata e al variare della posizione variano le informazioni mostrate all'utente.

Figura 1.2 Applicazione AR basata sulla posizione

In Figura 1.2 viene mostrato un esempio applicazione che, tramite la realtà aumentata e la posizione del dispositivo, mostra all'utente le attività commerciali nelle vicinanze.

2. **AR basata sulla proiezione:** le informazioni vengono aggiunte attraverso la proiezione sull'ambiente reale come utilizzando un proiettore. Un esempio di questo tipo di applicazioni è quella mostrata in Figura 1.3 in cui, grazie alla realtà aumentata, è possibile visualizzare un oggetto, magari da dover acquistare, nell'ambiente in cui è destinato.

Figura 1.3 Applicazione AR basata sulla proiezione

3. **AR "user-defined":** applicazioni sviluppate per risolvere problemi specifici. Un esempio di questo tipo di app sono quelle per effettuare le misurazioni di un ambiente, come quella mostrata in Figura 1.4.

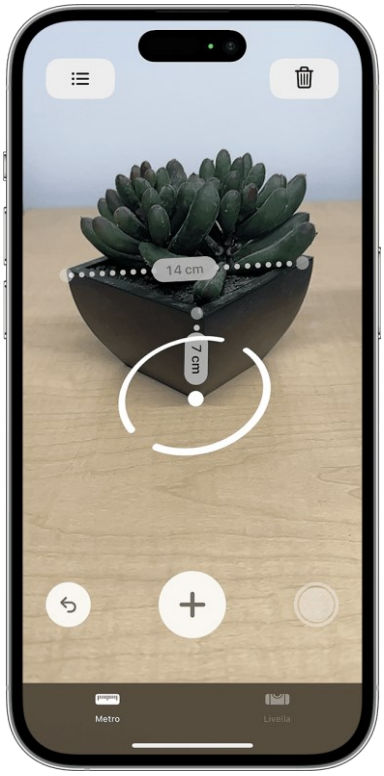

Figura 1.4 Applicazione AR per la misurazione

AR basata sui marker

Nel caso di AR basata sui marker, durante la scansione dell'ambiente il software è alla ricerca di un marker, che può essere un oggetto, un'immagine, un codice QR che è stato definito in fase di sviluppo. Una volta identificato l'elemento, il software di AR sovrappone all'elemento del contenuto multimediale sotto forma di testo, immagini, video, modelli 3D e molto altro come mostrato in Figura 1.5.

Figura 1.5 Applicazione AR basata sui marker

I contenuti aggiunti sono agganciati al marker e ciò permette di spostare l'elemento o il dispositivo che ha effettuato la scansione e mantenere i contenuti aggiunti al mondo reale, in realtà aumentata, visibili fino a quando il marker è inquadrato dal dispositivo. Questo è il tipo di AR che verrà utilizzata nel progetto.

2. Software Utilizzati

Dopo una prima fase di analisi è necessario individuare gli strumenti adatti a convertire i requisiti e gli obbiettivi del progetto. In questo capitolo verranno mostrati e spiegati tutti gli strumenti software utilizzati e il loro scopo per la realizzazione dell'applicazione e il tipo di servizi online per reperire i dati.

Il punto di partenza è stato Vuforia Engine: una piattaforma di sviluppo per esperienze di realtà aumentata che supporta lo sviluppo su dispositivi come smartphone, tablet e visori e disponibile gratuitamente sul *sito di Vuforia[5].* Questa piattaforma è facilmente integrabile con il motore grafico *Unity[6]* che è stato scelto a sua volta. Sulla di base di queste decisioni sono stati selezionati i programmi che agiranno a supporto dello sviluppo dell'applicazione.

Seguendo il flusso di sviluppo del progetto in ordine cronologico, partendo dall'inizio, il primo software utilizzato è *Polycam[7]* per la scansione dell'oggetto. Successivamente, da tale scansione viene generato un file che verrà successivamente migliorato in un programma di modellazione 3D, in questo caso *Blender [8]*, e successivamente sarà la base da cui il programma Model Target Generator genera un modello da importare in Unity. In quest'ultimo, dopo aver aggiunto il file scaricabile dal sito di Vuforia, sarà possibile utilizzare il modello precedentemente creato ed iniziare lo sviluppo dell'app. Per la scrittura del codice e il controllo di versione sono stati usati rispettivamente *Visual Studio [9]* e *GitHub [10]*, quest'ultimo anche nella versione GitHub Desktop. Durante lo sviluppo, nella fase di comunicazione con servizi online per recuperare dati, sono stati utilizzati pacchetti di codice disponibili sul gestore di pacchetti *NuGet[11].* In particolare, sono stati usati i pacchetti Newtonsoft.json e WebSocketSharp-netstandard. In questa fase è stato usato anche il software *Postman[12],* che permette di effettuare chiamate HTTP *[13]* e visualizzare le relative risposte.

I software sono stati scelti sulla base delle attrezzature disponibili ma anche alla loro disponibilità di utilizzo. Tutti i programmi utilizzati, infatti, sono completamente gratuiti oppure mettono a disposizione di tutti delle funzionalità limitate ma sufficienti per questo progetto; inoltre, sono disponibili su dispositivi smartphone con Android 12 o superiore e pc Windows, che sono quelli utilizzati nel nostro caso.

2.1. Vuforia Engine

Vuforia Engine è una piattaforma di sviluppo per esperienze di realtà aumentata che permette a chi la usa di aggiungere funzionalità complesse di computer vision ad applicazioni per dispositivi comuni che funzionano con sistemi come Android, MacOs e UWP. Vuforia Engine è facilmente utilizzabile negli editor di sviluppo come Android Studio, Xcode, Visual Studio e Unity, che, come detto in precedenza, è stata la nostra scelta.
Sul sito di Vuforia, nella sezione downloads sottosezione sdk, è possibile scaricare un file specifico per Unity identificato dal nome “Add Vuforia Engine to a Unity Project or upgrade to the latest version”. Una volta terminato di scaricare quest’ultimo avremo un file di tipo unitypackage che dovrà essere importato in Unity.
Oltre a questo file, per usare Vuforia Engine è necessario registrarsi al sito di Vuforia e creare un account. Una volta creato un account, ed effettuato l’accesso, è possibile aggiungere al proprio account una “chiave di licenza” nella sezione “Licences” cliccando sul bottone GET BASIC e seguendo i passaggi indicati; questa chiava sarà necessaria come vedremo successivamente in Unity.
Vuforia Engine mette a disposizioni molte possibilità per il riconoscimento e tracciamento di immagini, oggetti e ambienti come:
Image Target utilizzato per riconoscere le immagini in due dimensioni
Model Target utilizzato per riconoscere oggetti in tre dimensioni
Area Target utilizzato per riconoscere ambienti come una stanza.

Nell’ambito del progetto è stato scelto di adottare l’opzione Model Target che è quella che meglio si adatta al contesto ma ciò non esclude che le altre potessero avere dei vantaggi ed essere utilizzate.

2.1.1. Model Target

L'opzione Model Target consente all'applicazione sviluppata con Vuforia Engine di riconoscere e tracciare oggetti nel mondo reale in base alla forma dell'oggetto. Questo tipo di tracciamento è adatto ad oggetti come elettrodomestici, giocattoli, veicoli ed apparecchiature industriali. Per creare un Model Target è necessario avere un modello 3D dell'oggetto, come un modello CAD 3D o una scansione 3D dell'oggetto creato da un'applicazione di terze parti. Per ottenere un buon risultato è consigliato seguire le *linee guida sulle caratteristiche dell'oggetto [14]*, scritte sul sito di Vuforia, come evitare oggetti deformabili e con superfici lucide ma ricercare elementi ricchi di particolati come mostrato nella Figura 2.1 e Figura 2.2.

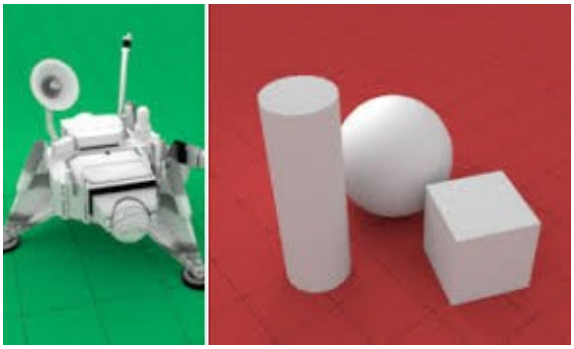

Figura 2.1 Linee guida oggetti complessi

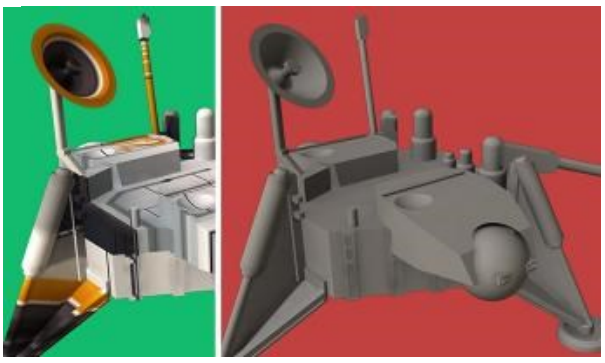

Figura 2.2 Linee guida oggetti lucidi

Nel caso non si possieda un modello 3D dell'oggetto, sul sito di Vuforia vengono suggerite delle applicazioni per crearlo; uno di questi è Polycam, di cui discuteremo in seguito.

Dal modello 3D dell'oggetto è facile ottenere il Model Target attraverso un software messo a disposizione gratuitamente da Vuforia chiamato Model Target Generator, approfondito nella prossima sezione.

2.2. Polycam

Polycam è un'applicazione per smartphone con un sito web collegato disponibile su dispositivi Android e permette di generare un modello 3D, tramite la realizzazione di video o foto.
Dopo aver scaricato l'app e creato un account è possibile iniziare a scansionare gli oggetti. Per realizzare una buona scansione è necessario inquadrare l'oggetto da tutte le angolazioni così da cogliere il maggior numero di particolari possibili. Una volta terminato di registrare il video dell'oggetto, o scattare le foto, il file verrà caricato e processato in automatico. Al termine dell'elaborazione il modello 3D sarà disponibile nella versione web di Polycam e potrà essere scaricato un file in formato gltf.

2.3. Blender

Blender è un programma multipiattaforma di modellazione, animazione, composizione di immagini tridimensionali e bidimensionali. In particolare, nel nostro caso di studio, è stato utilizzato per migliorare il modello 3D ottenuto da Polycam. Infatti, quest'ultimo può contenere delle parti indesiderate che non fanno parte dell'oggetto ma che sono state incluse nel processo di scansione. In Blender è possibile, dopo aver importato il file, eliminare le imperfezioni del modello attraverso la modalità di modifica per renderlo più simile alla realtà. Terminato il miglioramento del modello si può esportare ed è pronto ad essere utilizzato nel Model Target Generator. Questo passaggio attraverso Blender è necessario anche perché il modello risultante da Polycam non possiede coordinate spaziali richieste da Model Target Generator.

2.4. Model Target Generator

Model Target Generator è un'applicazione desktop, scaricabile dal sito di Vuforia nella sezione Downloads sottosezione Tools, che attraverso alcuni passaggi di configurazione permette di convertire un modello 3D in un dataset utilizzabile in Unity.
Model target Generator supporta modelli 3D in diversi formati e, nel nostro caso, è stato usato il formato gltf che è un formato standard per file di scene e modelli tridimensionali.
La schermata iniziale, mostrata in Figura 2.3, è composta dall'elenco dei singoli Model Target che sono stati creati e vengono identificati dal nome, data dell'ultima modifica, viste create e stato del modello.

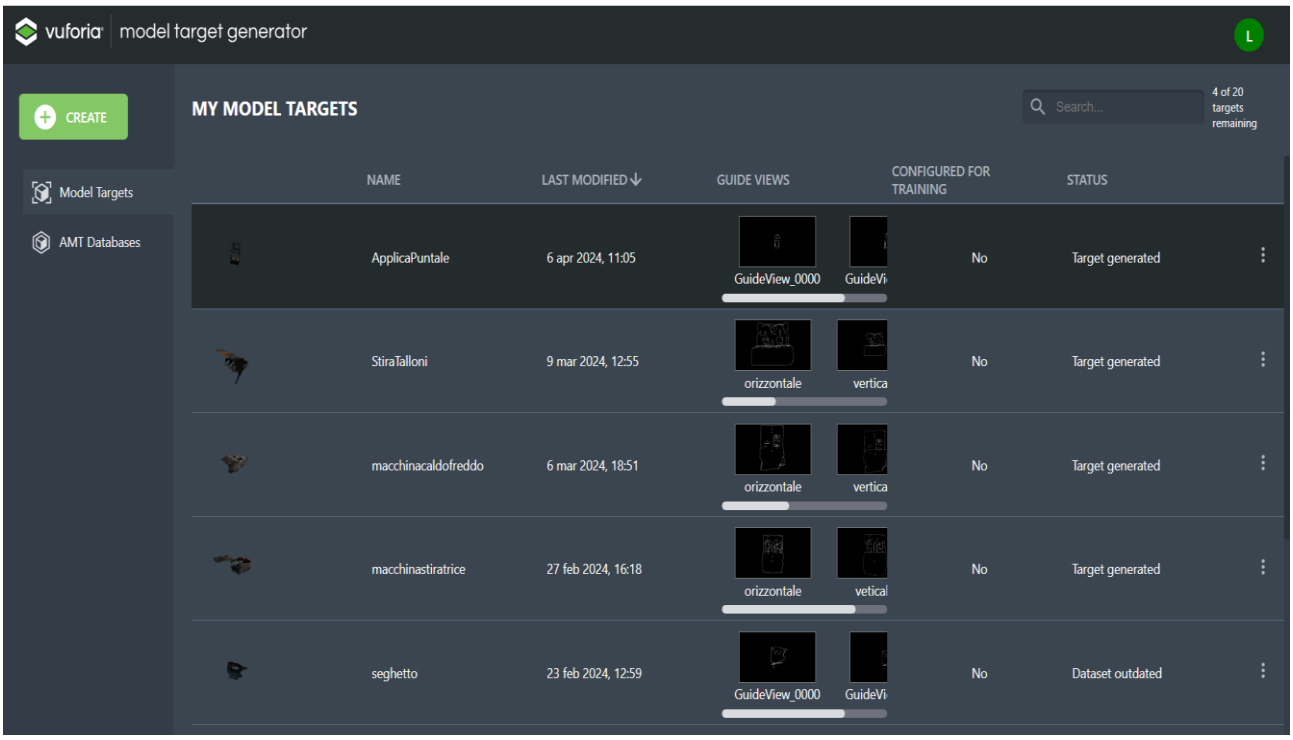

Figura 2.3 Schermata iniziale Model Target Generator

A sinistra è possibile anche passare all'elenco dei database creati che riporta le stesse informazioni dell'elenco dei Model Target.
In alto a sinistra è presente il tasto per iniziare una nuova creazione.

Una volta avviata la creazione e selezionato il modello 3D da convertire in Model Target, si aprirà il pannello di configurazione da cui è possibile definire, oltre ad altri parametri come la scala e la colorazione, il metodo di riconoscimento dell'oggetto che può essere:

- **Vista Guidata**: quest'approccio si basa nello stabilire a priori quale sarà l'angolazione di riconoscimento dell'oggetto. Impostando una angolazione dell'oggetto, rispetto al dispositivo che dovrà riconoscerlo, Model Target Generator salverà, in un file png, lo "scheletro" dell'oggetto che dovrà essere sovrapposto all'oggetto reale per permettere l'aggancio in fase di esecuzione dell'app finale. Lo scheletro dell'oggetto è l'immagine di riferimento che Vuforia utilizza per il riconoscimento.

- **Vista Avanzata**: a differenza dell'approccio precedente in questo caso non è necessario impostare una posizione ma sarà il Model Target Generator, attraverso l'addestramento di un database con un processo di formazione tramite deep learning basato su cloud, a permettere all'oggetto di essere riconosciuto da una qualsiasi angolazione. Tale database può contenere anche più di un Model Target.

L'addestramento di un database richiede diverse ore e, con la versione gratuita, è disponibile un numero limitato di volte. Per questi motivi, come vedremo in seguito più approfonditamente, è stato scelto di utilizzare il Model Target generato con le viste guidate. Una volta creato un Model Target è sempre possibile modificare i parametri di configurazione ed aggiungere nuove viste ma l'applicazione Model Target Generator sottrarrà un altro Model Target al numero di quelli che è possibile creare.
Dall'elenco è poi possibile esportare il Model Target desiderato come vedremo in fase di implementazione.

2.5. GitHub e GitHub Desktop

GitHub è un servizio di hosting per progetti software. Quest'ultimo è un'implementazione dello strumento di controllo di versione Git e permette agli sviluppatori di salvare su cloud e condividere il codice online così da avere un backup in caso di problemi.
GitHub in particolare mette a disposizione anche un'applicazione desktop, chiamata GitHub Desktop, che rileva in automatico le modifiche sui file di progetto locali e gestisce i conflitti nel caso di lavori di gruppo tramite dei "commit" per salvare le ultime modifiche effettuate e dei "merge" per unire le modifiche fatte ai file precedenti.
GitHub permette anche di accedere a versioni precedenti del progetto; per fare ciò è utile sapere che ad ogni commit è associato un identificatore e grazie a questo è possibile tornare alla versione del codice al momento di quel commit.
Quando viene caricato un progetto su GitHub, viene creato un file chiamato gitignore che si occupa di gestire quei file che non devono essere caricati o aggiornati in un progetto. In questo caso aggiungendo Vuforia Engine a Unity è stato creato un file .tgz, un formato di file compresso, che è stato aggiunto nel gitignore per permettere il salvataggio su GitHub del progetto.

2.6. Postman

Postman è uno strumento software che consente di testare le API in un ambiente grafico di facile utilizzo, mettendo a disposizione un'interfaccia che, in questo progetto, ha permesso di testare le risposte dei programmi creati sul server dell'università che inviano i dati richiesti nel formato per lo scambio dati JSON. Un file di tipo JSON ha una struttura precisa e può contenere tre diverse componenti: un oggetto, una lista e un elemento che può essere a sua volta una stringa, un numero oppure un valore vero o falso.

2.7. Unity

Unity è un motore grafico multipiattaforma che consente lo sviluppo di videogiochi, applicazioni e altri contenuti interattivi. Questo editor supporta i linguaggi C++ e *C# [15]*. Come detto in precedenza, supporta lo sviluppo di applicazioni che sfruttano le potenzialità di Vuforia Engine. L'eseguibile dal sito contiene anche Unity Hub, l'applicazione responsabile della gestione dei progetti e delle varie versioni dell'editor che è possibile scaricare. La versione dell'editor utilizzata per lo sviluppo di questo progetto è stata la 2022.3.17f1.

Aprendo Unity Hub vengono visualizzati i progetti creati e la relativa versione dell'editor utilizzata. In alto a destra è presente il tasto per creare un nuovo progetto; dopo averlo premuto sarà possibile dare il nome al progetto e selezionare il template del progetto come mostrato in Figura 2.4. In questo caso è stato scelto 3D(built-in) che rappresenta un progetto 3D standard senza l'aggiunta di librerie specifiche.

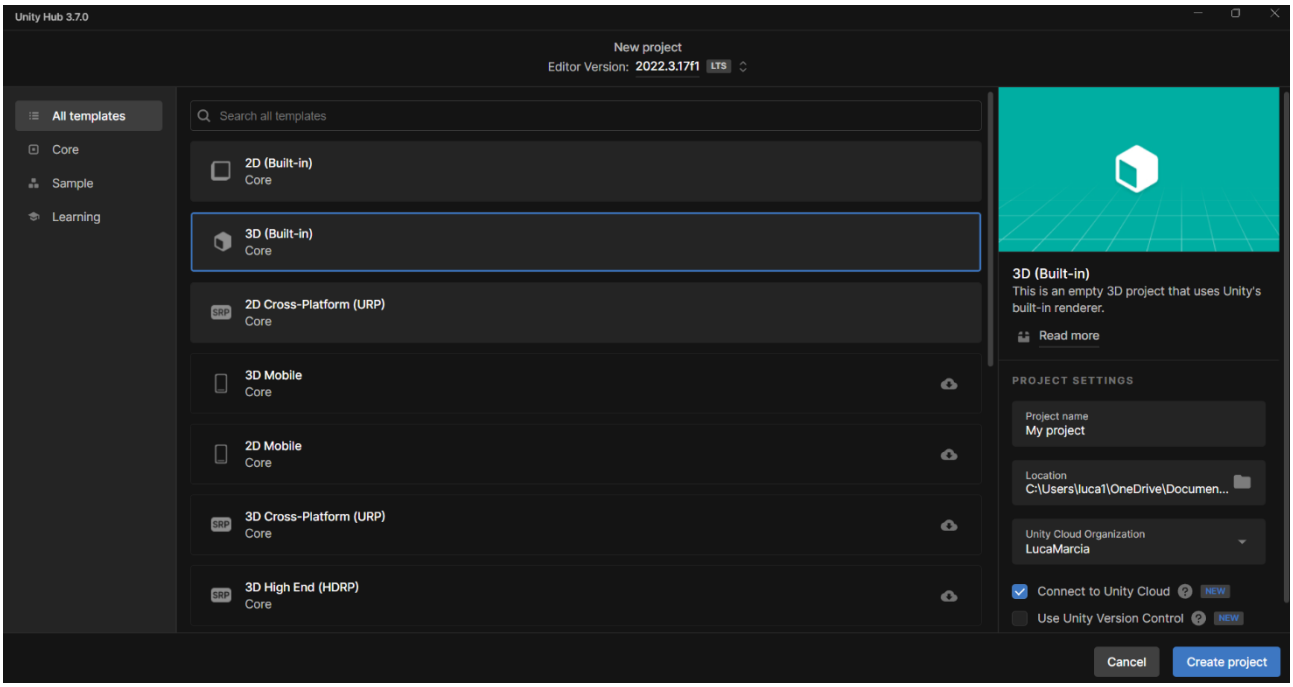

Figura 2.4 Schermata creazione progetto Unity

2.8. Visual Studio

Visual Studio è un ambiente di sviluppo multipiattaforma che supporta diversi linguaggi di programmazione, tra cui i più utilizzati sono C, Java e Python.
È disponibile in molte versioni ed in questo caso è stata usata la 2022 che, se richiesto, viene scaricata insieme ad Unity e si integra autonomamente. Grazie a quest'integrazione, Visual Studio è sincronizzato con la cartella del progetto Unity ed anche dei pacchetti aggiunti ad esso e, quindi, capace di vedere librerie e classi aggiunte al progetto e segnalare errori qualora si provino ad utilizzare classi non presenti nella cartella.
In questo progetto Visual Studio è stato utilizzato per sviluppare il codice utilizzato in Unity in linguaggio C#, per scrivere i due programmi che restituiscono i dati scritti in *Python [16]* e infine per strutturare i file JSON da cui uno di quest'ultimi recupera e rielabora i dati per poi inviarli.
Utilizzando Visual Studio sono stati anche eseguiti in fase di test i programmi destinati ad essere eseguiti sul server universitario.

2.9. NuGet

NuGet è un gestore di pacchetti, utilizzato principalmente per creare pacchetti e distribuire software. Nel progetto è stato integrato in Unity per utilizzare due pacchetti:

- *Newtonsoft.Json[17]*: un pacchetto che contiene delle classi, in linguaggio C#, che facilitano l'utilizzo di file di tipo JSON.
- *WebSocketSharp-netstandard[18]*: un pacchetto che contiene classi, in linguaggio C#, per instaurare comunicazioni con il protocollo webSocket, di cui discuteremo in seguito.

3. Progettazione

Individuati gli strumenti utili al progetto è ora il momento della progettazione. In questo capitolo vengono spiegate le scelte effettuate per l'interazione con i dati online e le scelte che riguardano l'applicazione, come aspetti di funzionamento e l'interfaccia.

3.1. Interazioni sulla rete internet

All'inizio della fase di progettazione, per memorizzare ed ottenere i dati online era stato valutato di utilizzare il servizio Google Firebase, che è facilmente integrabile nel codice scritto in C# e fornisce funzioni predefinite da adattare alla specifica situazione; purtroppo, in fase di test su dispositivi reali il kit di quest'ultimo creava conflitti con Vuforia Engine e dopo diversi tentativi, senza successo, è stato deciso di cambiare approccio.
Si è deciso, quindi, di creare ed eseguire due programmi in ascolto sul server dell'università.
Un programma è stato progettato per leggere dati da file JSON, rielaborarli e rispondere alle richieste tramite il protocollo di comunicazione HTTP, con un'interazione di tipo richiesta e risposta. Questo tipo di interazione è la più utilizzata quando si utilizza un motore di ricerca e consiste in un client che effettua una richiesta, per esempio una pagina web, e il server che, una volta ricevuta la richiesta, la processa e restituisce le informazioni in un formato che varia a seconda delle specifiche contenute nella richiesta; una volta ricevuta la risposta, la connessione tra il server ed il client si chiude.
Il secondo programma, invece, utilizza un protocollo diverso chiamato *websocket[19]*. Quando si utilizza il protocollo di comunicazione websocket, a differenza del HTTP, il client stabilisce una nuova connessione e rimane in attesa di ricevere i messaggi dal server che, durante questo tipo di connessione, si suppone arrivino con una determinata frequenza o comunque vengano inviati più di uno in un arco temporale di utilizzo. In questo caso è il client che decide di non ricevere più i messaggi e interrompe la connessione.
Il secondo programma creato genera un valore random, ogni cinque secondi, e lo invia, a chiunque si metta in ascolto, tramite il protocollo di comunicazione websocket. Nel caso specifico è proprio l'utente dell'applicazione che inizia la connessione con il server tramite un'interazione sullo schermo e da quel momento riceve nuove informazioni ogni cinque secondi fino a quando lo stesso utente, con un'altra interazione a schermo, decide di interrompere la connessione.
In Figura 3.1 viene mostrato un diagramma di queste interazioni.

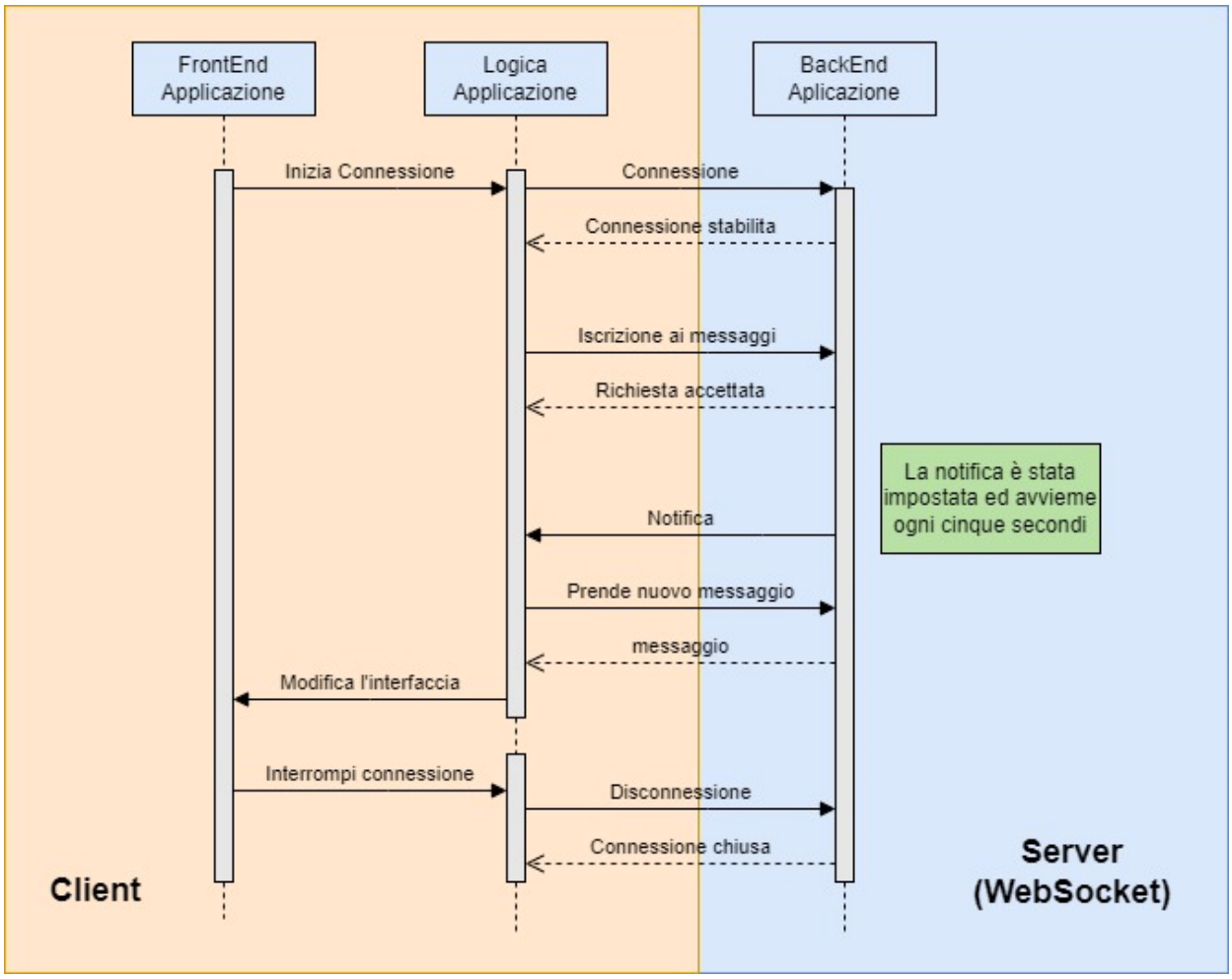

Figura 3.1 Diagramma connessione websocket

Nei programmi creati sono stati utilizzati due protocolli di comunicazione diversi in base alla natura del dato che devono restituire. In particolare, il protocollo HTTP è utilizzato quando le informazioni vengono richieste una volta e non devono essere aggiornate nel breve periodo come, per esempio, il modello del macchinario. Al contrario, il protocollo websocket è più efficiente quando l'informazione richiesta deve essere aggiornata in tempo reale. In ambito industriale, per esempio, la temperatura di un macchinario può variare repentinamente e l'operatore ha bisogno di conoscere lo stato del macchinario in tempo reale per poter effettuare le operazioni corrette.

3.2. Struttura dell'applicazione

L'applicazione è stata strutturata in quattro scene. La prima, mostrata in Figura 3.2, ha la funzione di schermata iniziale dell'app. Questa schermata è stata progettata per essere semplice ed intuitiva prima che accattivante per rispettare l'idea di un'applicazione utilizzata dai lavoratori. È stato inserito solo il necessario per la navigazione nelle altre scene. Sono stati inseriti due "dropdown menu" in cui nel primo sono indicate le stanze della fabbrica mentre nel secondo, dopo aver selezionato la stanza, sono elencati i macchinari presenti nella stanza scelta.

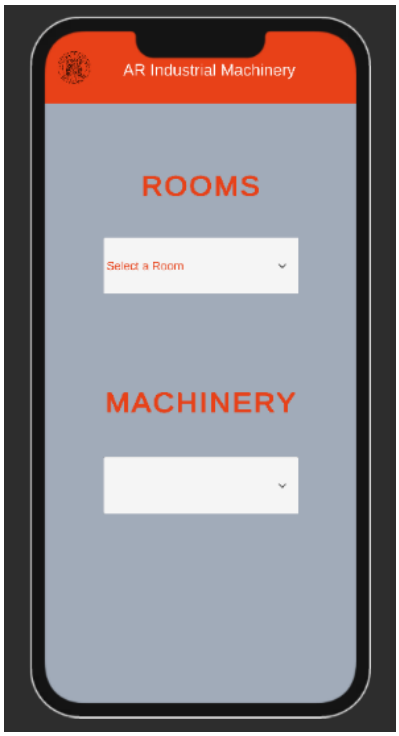

Figura 3.2 Simulatore con schermata iniziale

Nel momento che l'utente seleziona la stanza e il macchinario desiderato, l'applicazione apre in automatico la schermata corrispondente al macchinario scelto. Il funzionamento della schermata iniziale è mostrato in Figura 3.3.

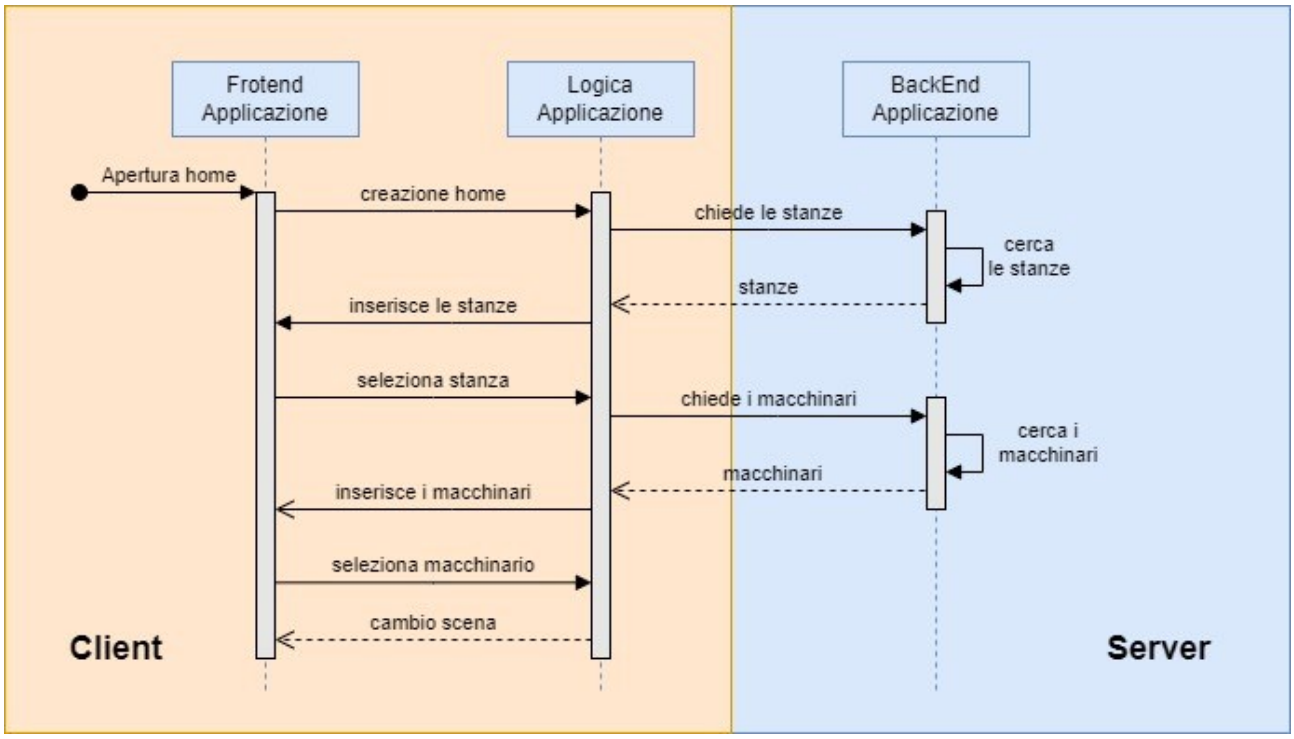

Figura 3.3 Diagramma funzionamento schermata principale

Dalla schermata iniziale si naviga verso una delle schermate dei macchinari. Le scene dei macchinari sono uguali tra loro con l'unica differenza che ognuna è caratterizzata da un Model Target e quindi all'apertura mostrerà una sagoma diversa a seconda del macchinario.

Figura 3.4 Simulatore con scena di un macchinario

Facendo riferimento alla Figura 3.4, ogni scena relativa ad un macchinario ha la seguente composizione:

- In alto a sinistra la freccia verso sinistra, che altro non è che un bottone per tornare alla schermata principale;
- In alto a destra il simbolo delle informazioni, che anch'esso è un bottone per aprire un pannello con le informazioni relative al macchinario;
- Al centro l'immagine di riferimento del macchinario, che dovrà essere allineato al macchinario reale per il riconoscimento;
- In basso un bottone, che accende/spegne la connessione al programma che utilizza il protocollo websocket.

Il funzionamento delle scene è mostrato nella Figura 3.5. In questa figura è stato omesso la parte di comunicazione tramite il protocollo websocket perché già illustrata nella Figura 3.1.

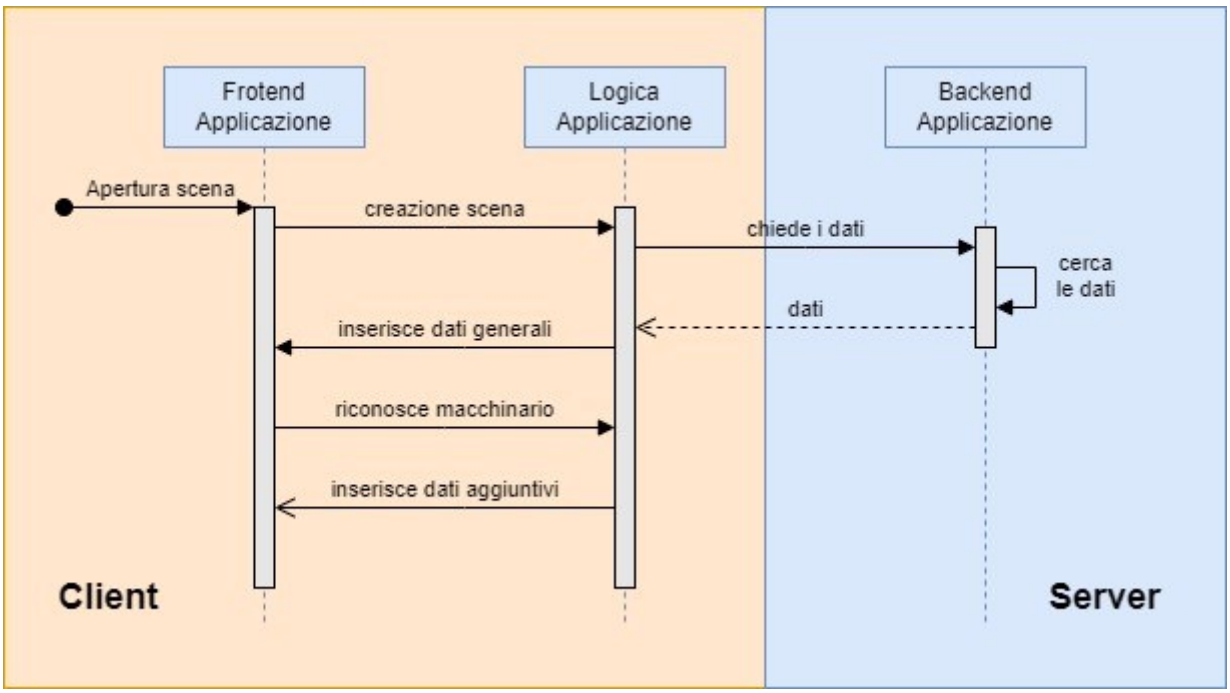

Figura 3.5 Diagramma funzionamento scene con Model Target

4. Implementazione

Finita la fase di progettazione e definiti tutti gli aspetti del progetto, è il momento di passare alla fase di sviluppo dell'applicazione.

Lo sviluppo può essere diviso in tre fasi principali:

- Creazione del Model Target
- Creazione API
- Sviluppo dell'applicazione

Le prime due fasi sono indipendenti tra loro mentre l'ultima unisce le prime due, aggiungendo elementi grafici per rendere l'esperienza dell'utente gradevole. Al termine di questo capitolo l'applicazione sarà terminata e nel capitolo successivo, oltre alle conclusioni, verranno anche proposte idee per una futura versione dell'app.

4.1. Creazione del Model target

In questa sezione verrà mostrata dall'inizio alla fine la creazione di un Model Target che poi potrà essere utilizzato dal Vuforia Engine all'interno di Unity.
Dopo aver scelto l'oggetto da cui si vuole ottenere il corrispondente Model Target, seguendo le linee guida accennate nella sezione 2.1, è necessario, se non si possiede un modello CAD 3D, creare un modello 3D con applicazioni come Polycam.

4.1.1. Scansione dell'oggetto

Aprendo l'applicazione Polycam su uno smartphone, nella schermata iniziale, verrà mostrato l'elenco delle scansioni effettuate, come si vede in Figura 4.1.

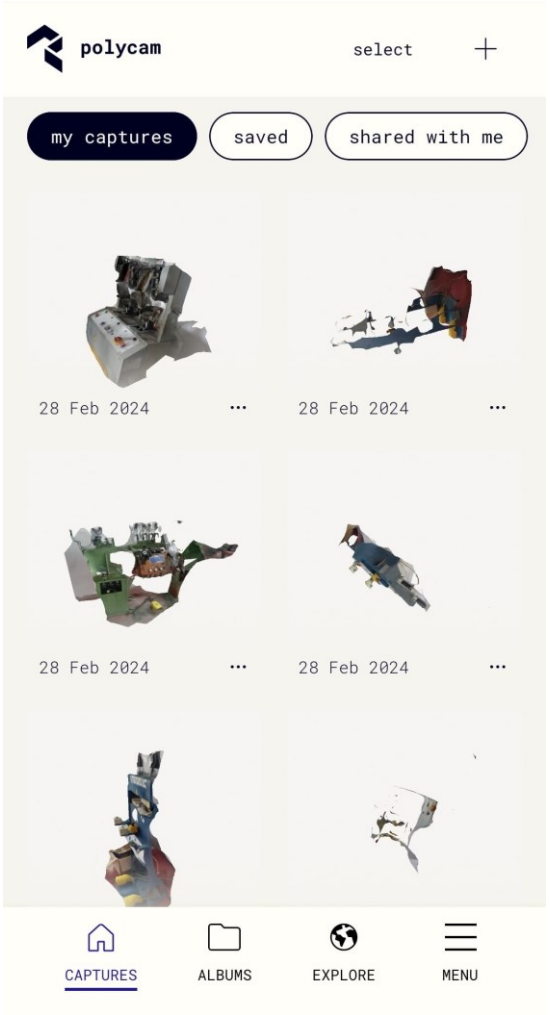

Figura 4.1 Elenco scansioni effettuate polycam

Per iniziare una nuova scansione è sufficiente premere sul tasto “+” in alto a destra. Così facendo, si aprirà una schermata simile alla videocamera ma con bottoni specifici dell’app. A questo punto si inizia a registrare un video dell’oggetto oppure a scattare delle foto. Una volta terminato, sarà sufficiente premere il tasto “Done” e, all’apertura della successiva schermata, far partire il caricamento del file e il conseguente processamento. Al termine di questa fase di elaborazione è disponibile un’anteprima del risultato, come mostrato in Figura 4.2.

Figura 4.2 Esempio di scansione polycam

Utilizzando poi la versione web dell’applicazione è possibile scaricare sul computer, scegliendo tra i vari formati, il file della scansione. Nel progetto è stato usato il formato gltf, utilizzato successivamente anche nel Model Target Generator.

4.1.2. Miglioramento scansione e posizionamento spaziale

Il file ottenuto da Polycam non è ancora pronto per generare il Model Target ma necessita di essere migliorato con un software di modellazione 3D, nel nostro caso Blender.
Infatti, dopo essere stato scaricato, il file presenta due problemi diversi:

- Coordinate del contenuto
- Scansione con elementi indesiderati

Questi due problemi non sono trascurabili poiché, se al momento della generazione del Model Target l'oggetto non ha coordinate spaziali definite, l'applicazione Model Target Generator genera un errore, mostrato in Figura 4.3, e ne impedisce la creazione.

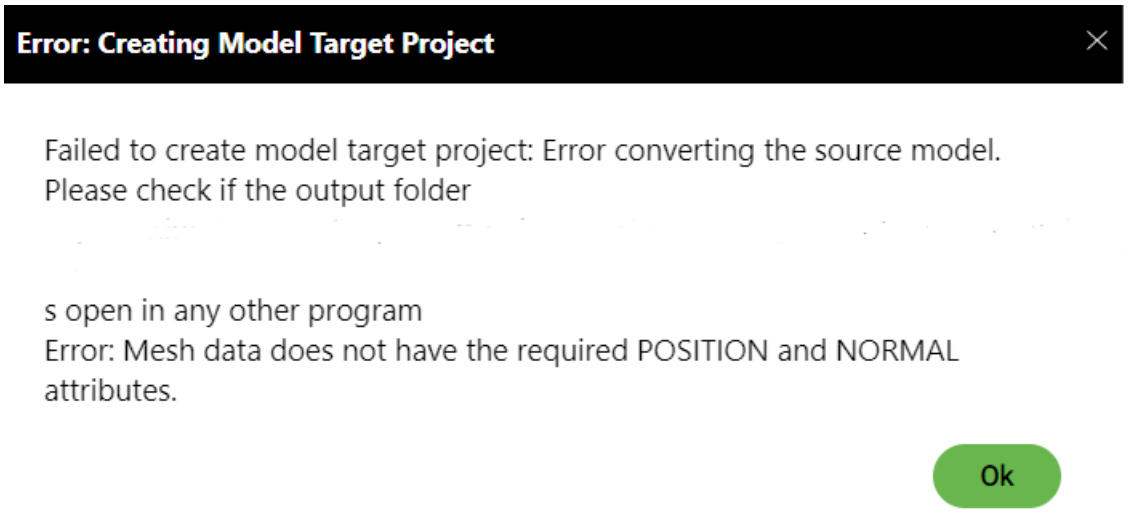

Figura 4.3 Errore per il mancato posizionamento spaziale

Questo primo problema si risolve autonomamente importando il file in Blender e poi esportandolo, mantenendo il formato; questo perché, quando viene aggiunto, ad esso viene assegnato di default la posizione per gli assi x, y, z nel punto (0,0,0).
Il secondo, a differenza del primo, non è un problema bloccante, cioè non impedisce la creazione del Model Target, ma ne crea uno con elementi indesiderati che potranno interferire sia nella fase di sviluppo dell'applicazione finale e sia nella fase di esecuzione per il riconoscimento dell'oggetto, in quanto verranno ricercati quegli elementi che potrebbero non essere presenti nello stesso oggetto spostato in un altro ambiente.
Naturalmente ogni scansione può avere più o meno bisogno di essere migliorata. Per

migliorare la scansione, dopo aver importato il file in Blender, è stata utilizzata la modalità di “Modifica Oggetto” selezionando le parti indesiderate ed eliminandole, tramite il tasto della tastiera “Canc” e l’opzione vertici.
Di seguito, in Figura 4.4, è mostrato il risultato del lavoro effettuato in Blender con un confronto tra prima e dopo il miglioramento.

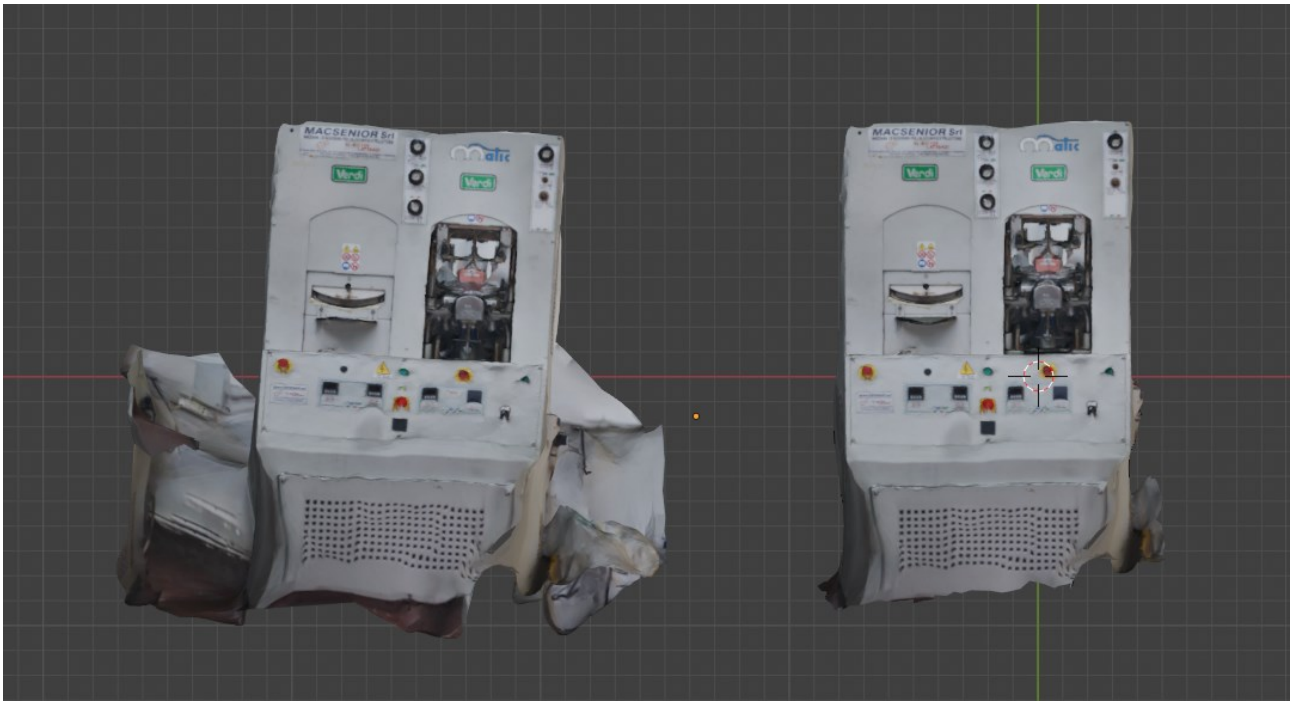

Figura 4.4 Confronto prima e dopo il miglioramento in blender

4.1.3. Creazione del Model Target

Dopo aver corretto i problemi citati nella sezione precedente, si può passare alla creazione del Model Target tramite l'applicazione Model Target Generator.
Come anticipato nel capitolo dei software utilizzati, all'apertura dell'applicazione in alto a sinistra è presente il tasto "Create" che, una volta cliccato e selezionato Model Target, mostrerà una schermata in cui si chiede di inserire il modello. Una volta selezionato il file discusso in precedenza, verrà mostrata una schermata con tutti gli aspetti di configurazione del Model Target quali:

- Vettore up modello
- Unità modello
- Colorazione
- Complessità
- Ottimizza monitoraggio
- Viste guida

Vettore up modello: definisce l'orientamento del modello importato e varia per ogni modello.

Unità modello: in un modello, di solito, non sono codificate le unità di misura; quindi, è necessario inserirle in modo che siano il più fedeli possibili all'oggetto reale.
Nel progetto uno dei macchinari scelti è alto 1,70 metri e largo 0,95 metri quindi, in questa fase, è stata assegnata l'unità di misura "metri" ai valori rilevati dal software, che possono essere errati, come mostrato in Figura 4.5.

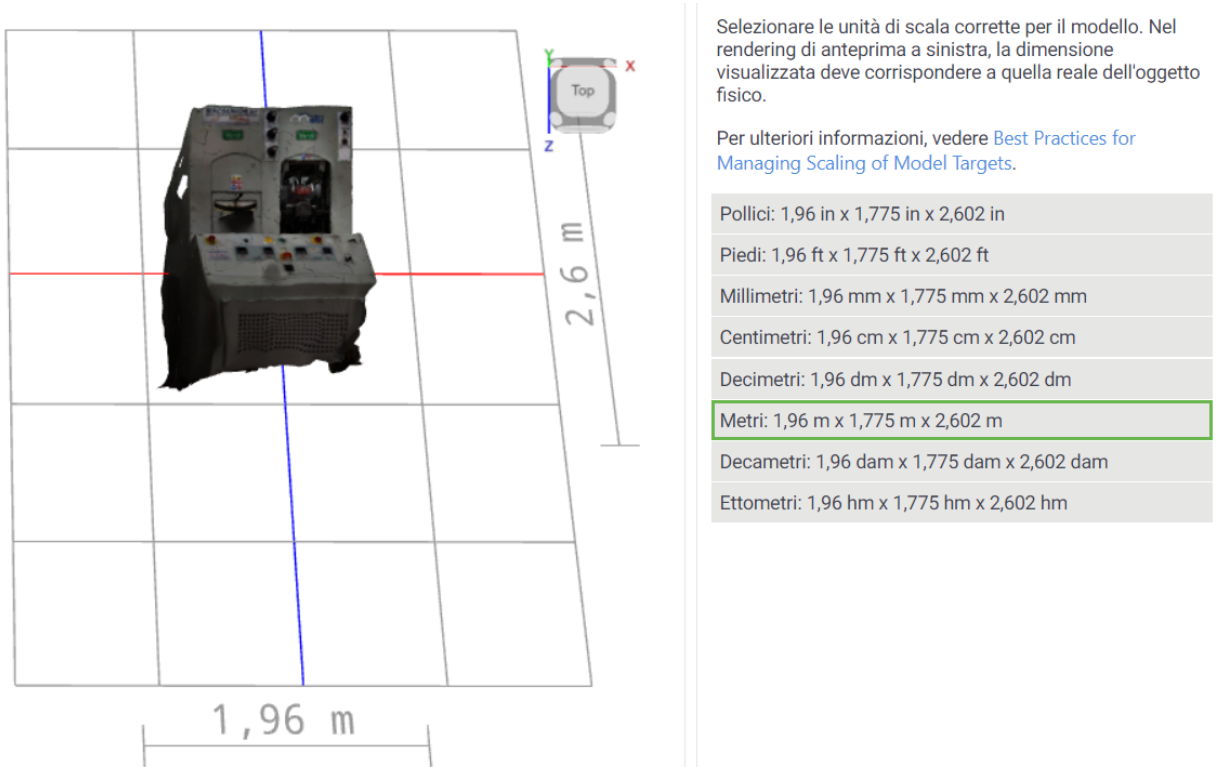

Figura 4.5 Pannello configurazione dimensioni oggetto

Colorazione: Nel nostro caso, il modello contiene anche l'aspetto cromatico dell'oggetto; Model Target Generator lo rileva e imposta la configurazione ad "aspetto realistico".

Complessità: L'applicazione rileva se il modello è abbastanza dettagliato per la generazione del Model Target. In caso negativo, impedisce la creazione.

Ottimizza monitoraggio: Mette a disposizione un parametro di configurazione dipendente dalle dimensioni dell'oggetto per ottimizzare il monitoraggio. Nel progetto è stato impostato a "Default", utilizzando modelli di oggetti di grandi dimensioni.

Viste guida: In questo step della configurazione, Model Target Generator da all'utente la possibilità di scegliere il metodo di riconoscimento. Le opzioni messe a disposizione sono due:

- Crea vista guida
- Crea vista avanzata

Nel progetto sono state utilizzate le viste guida e, nella sezione successiva, verrà approfondito il processo di creazione con alcune accortezze utilizzate.

4.1.4. Creazione delle viste

La generazione delle viste è l'ultimo passaggio di configurazione necessario per creare il Model Target. La Figura 4.6 mostra la schermata adibita alla creazione.

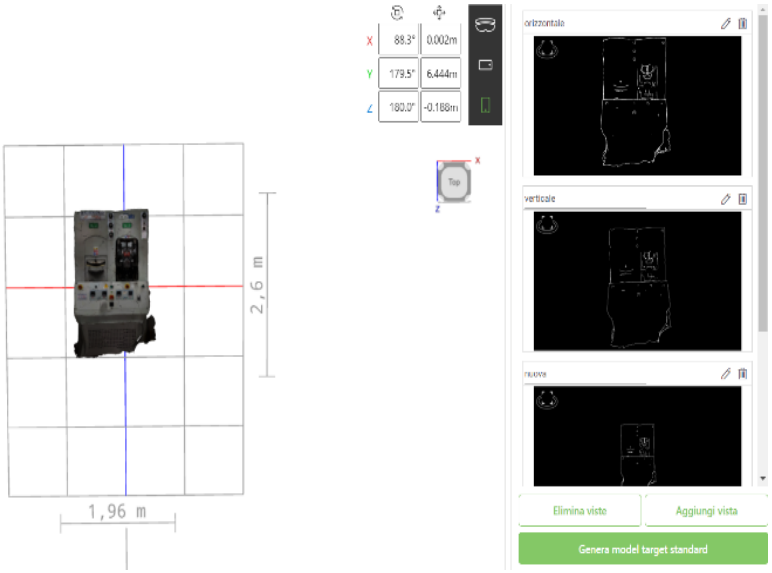

Figura 4.6 Pannello creazione vista guidata

Al centro, vicino alle coordinate x, y, z, è possibile selezionare il tipo di dispositivo destinato ad utilizzare la vista, come occhiali smart o smartphone, con a disposizione a sua volta le modalità di utilizzo orizzontale e verticale. Per creare una nuova vista è sufficiente cliccare sul tasto in basso a destra "Aggiungi vista" e, una volta inquadrato l'oggetto con l'angolazione desiderata, premere il tasto "Crea vista guidata" nella schermata successiva. Nel progetto sono state create più di una vista per entrambe le modalità di utilizzo dello smartphone e per distanze dall'oggetto diverse. Proprio l'aspetto della distanza dall'oggetto è importante per l'applicazione finale destinata all'utente: infatti, se nella vista l'oggetto è troppo vicino o troppo lontano, ciò renderà impossibile all'utente, in fase di utilizzo, riconoscere gli oggetti.

Figura 4.7 File esempio vista guidata

Poiché è possibile, una volta create le viste e generato il Model Target, avvicinare l'oggetto se è troppo distante, e non allontanarlo, è consigliabile creare le viste con l'oggetto molto distante, come mostrato in Figura 4.7, e poi modificare il file png, presente nella cartella generata con il Model Target, relativo alla vista scelta. Il file da modificare, in un qualsiasi editor di foto, è quello che contiene le linee dell'oggetto e non quello con tutti i dettagli.

4.1.5. Aggiungere il Model Target in Unity

Una volta terminata la creazione delle viste, il Model Target creato viene aggiunto in automatico all'elenco delle creazioni effettuate. Per importare il Model Target in Unity bisogna, dall'elenco delle creazioni, cliccare sui tre puntini presenti sulla destra relativi al modello scelto. Si aprirà un menù in cui selezionare "Open destination folder"; una volta fatto, si aprirà la cartella generata da Model Target Generator con la sottocartella "dataset", in cui è presente il file in formato unitypackage. Ora è sufficiente, con Unity aperto con il progetto desiderato, fare doppio click sul file e premere "Import" sulla schermata che si aprirà in Unity. In alternativa, da Unity è possibile importare il file, tramite Import Assets.

4.2. Creazione API

Dopo la creazione del Model Target, una parte molto importante del progetto è la ricezione e l'invio di dati tramite internet. Nel progetto le informazioni utilizzate sono predefinite, contenute in file nel formato di tipo JSON, o vengono generate casualmente. L' aspetto importante, però, è la possibilità di ottenere dati online e visualizzarli, attraverso la realtà aumentata, sull'oggetto. Per far ciò, è stato necessario creare due programmi, eseguiti sul server dell'università, che, a richiesta, restituiscano dati. Entrambi i programmi sono scritti in Python e fruttano librerie esterne come *Flask[20]*, per connessioni HTTP, e Websockets, per connessioni websocket.

Il primo programma è responsabile della risposta alle chiamate inviate con il protocollo HTTP e del salvataggio dei dati provenienti dall'applicazione. Questo programma mette a disposizione dell'utente diverse chiamate parametrizzate, come mostrato in Figura 4.8.

```
app = Flask(__name__)
api = Api(app)

#Route machinary
api.add_resource(homey.GetRooms, '/listroom')

api.add_resource(homey.GetAllMachinery, '/allmachinery')

api.add_resource(homey.GetMachineryByRoomId, '/machineryByRoom')

api.add_resource(homey.GetMachineryById, '/machinery')

api.add_resource(homey.ChangeState, '/changeState')

api.add_resource(homey.GetTasksById, '/getTaskById')


if __name__ == '__main__':
    app.run(host='192.168.104.56', port=5004)
```

Figura 4.8 Lista funzioni del programma con connessione HTTP

Le richieste restituiscono dati rielaborati, in formato JSON, con una logica definita a priori come, per esempio, in Figura 4.9, ottenuti da file statici, anch'essi di tipo JSON.

```python
#select all the machinery
class GetAllMachinery(Resource):
    def get(self):
        result = []
        with open("machinery.json", 'r') as file:
            data = json.load(file)
            for room in data["rooms"]:
                machine = [obj for obj in  room["machinery_list"]]
                result += machine



        return jsonify(result)
```

Figura 4.9 Esempio funzione del programma con connessione HTTP

Il secondo programma, invece, utilizza il protocollo websocket e, dopo l'instaurazione di una connessione con il client, invia ogni cinque secondi un messaggio, in formato JSON, contenente la stringa "temperature" seguita da un valore random generato dal programma stesso, come mostrato in Figura 4.10.

```python
stop_event = Event()


async def handler(websocket):
    data = await websocket.recv()
    print(data)

async def echo(websocket):
    #async for message in websocket:
    #    await websocket.send(message)
    try:
        while True:
            message = {'temperature': random.randint(20, 200)}
            print(message)
            await websocket.send(json.dumps(message))
            # Sleep for 5 seconds (adjust the interval as needed)
            stop_event.wait(5)
    except websockets.exceptions.ConnectionClosed:
        print(f"Connection with {websocket.remote_address} closed.")

async def main():
    async with serve(echo, "192.168.104.56", 2000):
        try:
            await asyncio.Future() # run forever
        except KeyboardInterrupt:
            server.close()
            await server.wait_closed()
            print("WebSocket server gracefully closed.")

asyncio.run(main())
```

Figura 4.10 Codice programma sul server per connessione websocket

In fase di test dell'applicazione, il primo programma è stato eseguito anche sul pc locale così da facilitare lo sviluppo. In questa fase, l'applicazione era eseguita nel simulatore presente in Unity così tramite l'uso dell'indirizzo 127.0.0.1, o comunemente localhost, è stato possibile testare l'app senza continue modifiche sul server universitario, che richiede del tempo per essere riavviato.

4.3. Sviluppo dell'applicazione

Dopo aver illustrato la creazione del Model Target e lo sviluppo dell'approccio ai dati online, è ora il momento di vedere nel dettaglio lo sviluppo dell'applicazione finale.
Il questo capitolo verrà spiegato più nel dettaglio come utilizzare il Model Target in Unity, come visualizzare scritte ed immagini in realtà aumenta, come utilizzare i dati provenienti dal web e tutto ciò che è stato fatto nell'applicazione finale.

4.3.1. Impostazioni per la build

Prima di iniziale a parlare dello sviluppo è necessario, però, dedicare una sezione alle impostazioni con cui viene creato il file eseguibile dell'applicazione. Questo passaggio è stato necessario principalmente per diminuire i tempi di creazione dell'app ma anche per altre ragioni che vedremo in seguito. Queste impostazioni sono modificabili direttamente dall'editor Unity recandosi in alto a sinistra e cliccando su "File" e successivamente su "Build Settings". Nella nuova schermata è stata abilitata la modalità "Development Build" e, dopo aver selezionato la piattaforma su cui l'applicazione è destinata ad essere eseguita, nel nostro caso Android, è stata eseguita la modifica con il tasto "Switch Platform" in basso a destra. Fatto questo passaggio, è il momento di modificare le impostazioni; per fare ciò è sufficiente cliccare sul tasto "Player Settings", andare nella sezione "Other Settings" ed effettuare le seguenti modifiche.
La prima modifica è la rimozione dell'"Auto Graphics Api" e della "Graphics API" chiamata "Vulkan" se presente per migliorare la fluidità dell'applicazione.
Le altre modifiche riguardano, invece, come l'applicazione viene costruita e l'accesso ad Internet. Nel primo caso è stato scelto come "Scripting Backend" "IL2CPP" e, di conseguenza, per contenere il tempo di build, selezionata l'architettura "ARM64" invece della "ARMv7". L'ultima modifica è l'attivazione del permesso di poter stabilire la connessione su HTTP anche in maniera non sicura ma solo in fase di sviluppo, che è la nostra situazione. Entrambe le modifiche effettuate sono mostrate in Figura 4.11.

Figura 4.11 Impostazioni di build

Infine, è necessario sapere che In Unity le diverse schermate di un'applicazione sono chiamate "scene", ed ogni scena, dopo essere stata creata, deve essere aggiunta all'applicazione nella schermata di build premendo il tasto "Add Open Scenes".

4.3.2. Disposizione statica delle schermate

La schermata principale e alcune parti delle schermate dei macchinari sono statiche, ovvero non variano dopo il riconoscimento del macchinario. Per fare ciò, Unity mette a disposizione un elemento chiamato "canvas". Questo elemento altro non è che il riquadro in cui inserire, nel nostro caso, gli elementi statici, che nella schermata principale sono i due menù di selezione mentre nelle scene dei macchinari i vari tasti per tornare indietro, le informazioni e la connessione. Dopo aver inserito il riquadro nella gerarchia degli oggetti è necessario configurarlo. Nel pannello di configurazione è stata impostata la "Render Mode" su "Screen Space – Overlay" e la "Reference Resolution" sui valori "1920x1080", come mostrato in Figura 4.12.

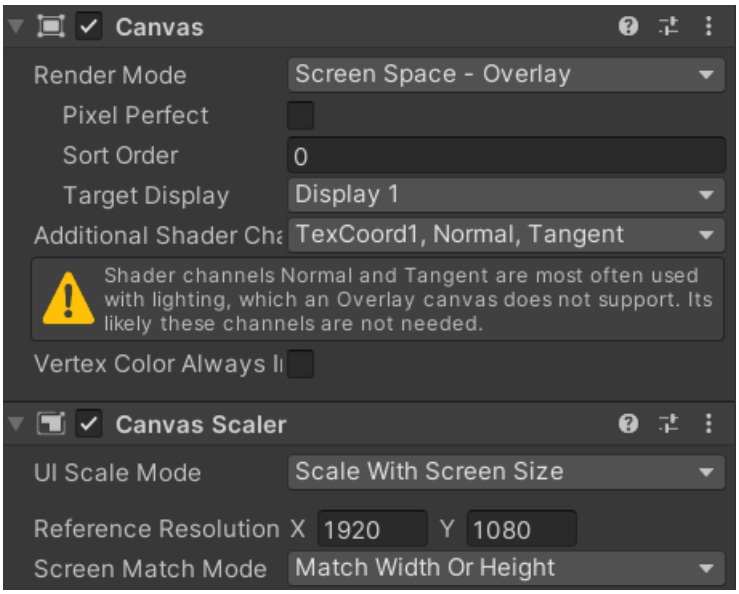

Figura 4.12 Configurazione canvas

Fatto ciò, è ora possibile inserire gli elementi desiderati, che dovranno essere figli del riquadro nella gerarchia e posizionati, di conseguenza, come in Figura 4.13.

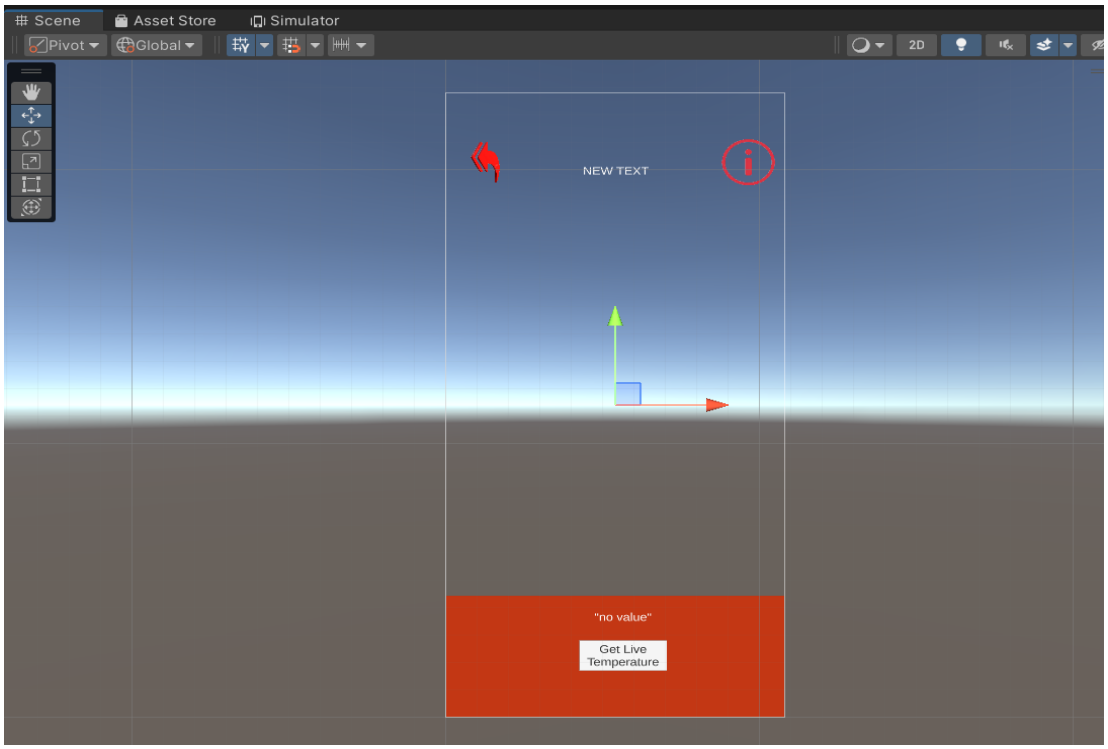

Figura 4.13 Interfaccia statica con canvas

Gli elementi grafici inseriti sono tutti relativi alla sezione “UI” presente tra le opzioni quando si inserisce un oggetto nella gerarchia. Se un oggetto deve avere un comportamento dinamico, a quest’ultimo deve essere aggiunto un file di codice nella sezione “Inspector” oppure è possibile aggiungere un oggetto vuoto che viene configurato per gestire i file di codice necessari. In questo caso, ogni codice dovrà avere un parametro visibile all’esterno a cui assegnare l’elemento da controllare, come mostrato in Figura 4.14.

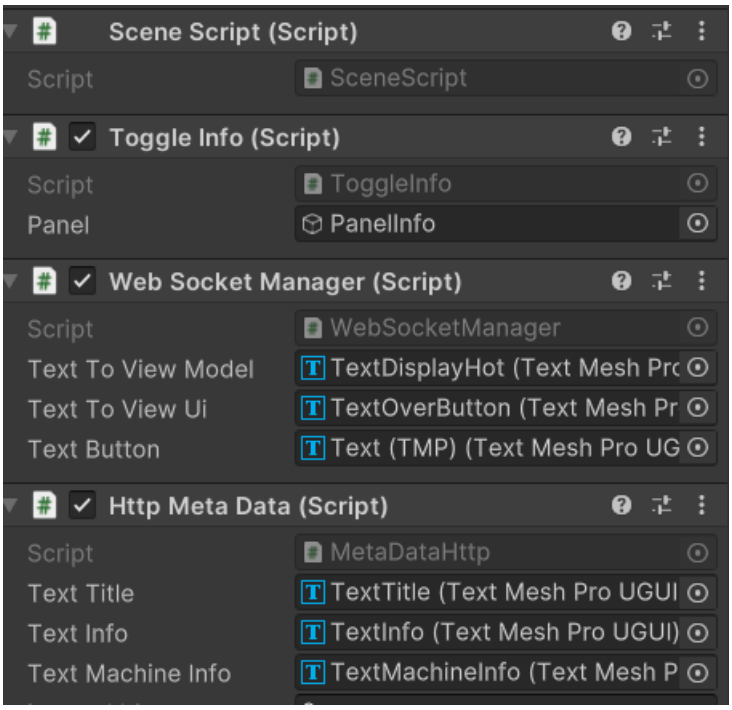

Figura 4.14 Esempio oggetto contenitore per file di codice

Infine, se l'elemento da controllare è un bottone sono predefinite delle funzioni da configurare come, per esempio, la "OnClick" che viene eseguita quando rileva l'interazione dell'utente sullo schermo.

4.3.3. Utilizzo Vuforia Engine

Le scene dei macchinari sono quelle relative alla realtà aumentata e, per fare ciò, si servono di Vuforia Engine. Per utilizzare Vuforia Engine è necessario, dopo aver importato il file contenente il Model Target, eliminare la “Main Camera” ed aggiungere l’“AR Camera”, presente nella sezione Vuforia Engine, nella gerarchia degli oggetti presenti nella parte sinistra di Unity, come mostrato in Figura 4.15.

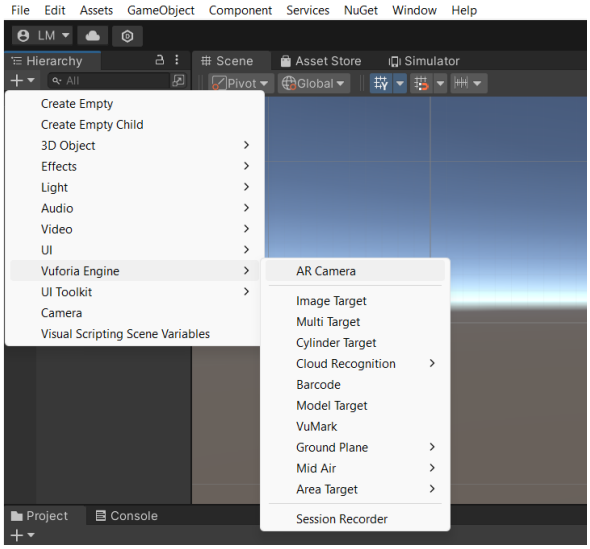

Figura 4.15 Aggiunta AR camera

L’AR Camera deve essere configurata: è sufficiente selezionarla nella gerarchia e nel pannello Inspector, che si apre sulla destra, e cercare la voce “Open Vuforia Engine Configuration”. Cliccando su quest’ultima si apre un altro pannello in cui deve essere inserita, nella casella “App Licence Key”, la chiave creata precedentemente, come illustrato nella Sezione 2.1.
Facendo ciò, l’AR Camera funziona ed è arrivato il momento di aggiungere il Model Target.

4.3.4. Collegamento online

Come detto in precedenza, uno degli obbiettivi del progetto è-la visualizzazione di dati recuperati da risorse sul web, in questo caso dal server dell'università, in realtà aumentata sul macchinario riconosciuto dall'applicazione. In questa sezione verrà mostrato il metodo utilizzato sia per le connessioni con protocollo HTTP che per quelle con il protocollo websocket.

Per le connessioni HTTP è stata utilizzata una libreria standard di Unity chiamata "UnityEngine.Networking" che mette a disposizione la classe "UnityWebRequest", che contiene metodi per instaurare la connessione, gestire gli errori ed accedere alla risposta del server. La funzione creata è mostrata in Figura 4.16.

```
//method that get the metadata from an api call and put on the ui element
1 riferimento
IEnumerator getMetaData()
{
    using (UnityWebRequest request = UnityWebRequest.Get(_urlsetState + internalId ))
    {

        yield return request.SendWebRequest();
        if (request.result == UnityWebRequest.Result.ConnectionError || request.result == UnityWebRequest.Result.ProtocolError)
        {
            Debug.Log(request.result);
        }
        else
        {
            string json = request.downloadHandler.text;
```

Figura 4.16 Funzione per la connessione HTTP

Poiché una funzione di questo tipo può essere temporalmente lunga e dispendiosa, l'applicazione potrebbe non eseguirla quando previsto e creare un'incongruenza nella visualizzazione dei dati. Per evitare questo problema, molti linguaggi di programmazione mettono a disposizione le coroutine, che indicano al processore di eseguire queste funzioni in contemporanea. In C# le funzioni di questo tipo devono essere inserite come argomento alla funzione "StartCoroutine" e implementare l'interfaccia IEnumerator, come mostrato in Figura 4.17.

```
// Start is called before the first frame update
Messaggio Unity | 0 riferimenti
void Start()
{
    StartCoroutine(getMetaData());
}

//method that get the metadata from an api call and put on the ui element
1 riferimento
IEnumerator getMetaData()
```

Figura 4.17 Utilizzo coroutine

Per la connessione al server tramite protocollo websocket non sono disponibili librerie standard in Unity, quindi, è stato necessario utilizzare un pacchetto esterno chiamato "WebSocketSharp-netstandard" presente sul NuGet.

Questo pacchetto mette a disposizione una libreria chiamata WebSocketSharp che contiene la classe "WebSocket" per gestire la connessione. Importante in questo caso è la funzione "OnMessage", che viene invocata ogni volta che il server invia un nuovo messaggio al client. Il codice è mostrato in Figura 4.18.

```
//create the object for the web socket connection
WebSocket _ws = new WebSocket("ws://193.205.129.120:63425");

// Start is called before the first frame update
Messaggio Unity | 0 riferimenti
void Start()
{
    //attach a listener to the web socket
    _ws.OnMessage += (sender, e) =>
    {
        //on new message save the new data (after a json parser)
        _newdata =  JsonUtility.FromJson<ResponseWebSocket>(e.Data);
    };

}
```

Figura 4.18 Funzione per la connessione websocket

I messaggi ricevuti dal server sono in formato JSON e, nel caso della connessione websocket, rispettano lo standard. Infatti, sono racchiusi tra le parentesi graffe, che identificano l'oggetto da mappare, e quindi è stato possibile utilizzare la libreria standard di Unity chiamata "UnityEngine" che contiene la classe "JsonUnity", utilizzata come in Figura 4.18.

Nelle connessioni HTTP, invece, la risposta, per come è stata definita sul programma creato in precedenza, è comunque in formato JSON però è racchiusa tra le parentesi quadre, e ciò vuol dire che identifica una lista. In questo caso, è stato necessario ricorrere al gestore NuGet per l'utilizzo di una libreria esterna chiamata "Newtonsoft.Json", che mette a disposizione la classe "JsonConvert" che permette di mappare questi file JSON, come mostrato in Figura 4.19.

```
//use jsonconverter because the result is not an object but a list
_list = JsonConvert.DeserializeObject<List<ResponseMachinery>>(json);
```

Figura 4.19 Esempio utilizzo JsonConvert

In entrambi gli approcci, per poter utilizzare i dati è anche necessario far conoscere al compilatore come devono essere mappati. Per far ciò, indipendentemente dall'approccio, sono state create delle classi che corrispondono alla risposta del server. Un esempio di queste classi è mostrato in Figura 4.20.

```
public class ResponseMachinery
{

    public string brand;
    public string deployed_at;
    public string description;
    public string dimension_HxWxD_cm;
    public string elettrical_features_Hz;
    public string elettrical_features_V;
    public int internal_id;
    public string machine_type;
    public string name;
    public string serial_number;
    public string weight_Kg;
}
```

Figura 4.20 Classe esempio risposta web

4.3.5. Utilizzo Model Target

Dopo averlo importato, il procedimento per aggiungere il Model Target alla gerarchia di oggetti è analogo all'AR Camera. Per la sua configurazione bisogna selezionare, nel pannello Inspector, il Database e Model Target desiderati tra quelli importati. Fatto questo, per visualizzare, in fase di esecuzione, la vista di riferimento del macchinario bisogna impostare la voce "Guide View Mode" in "GuideView2D" e la voce "Guide View" con il nome della vista creata precedentemente come mostrato in Figura 4.21.

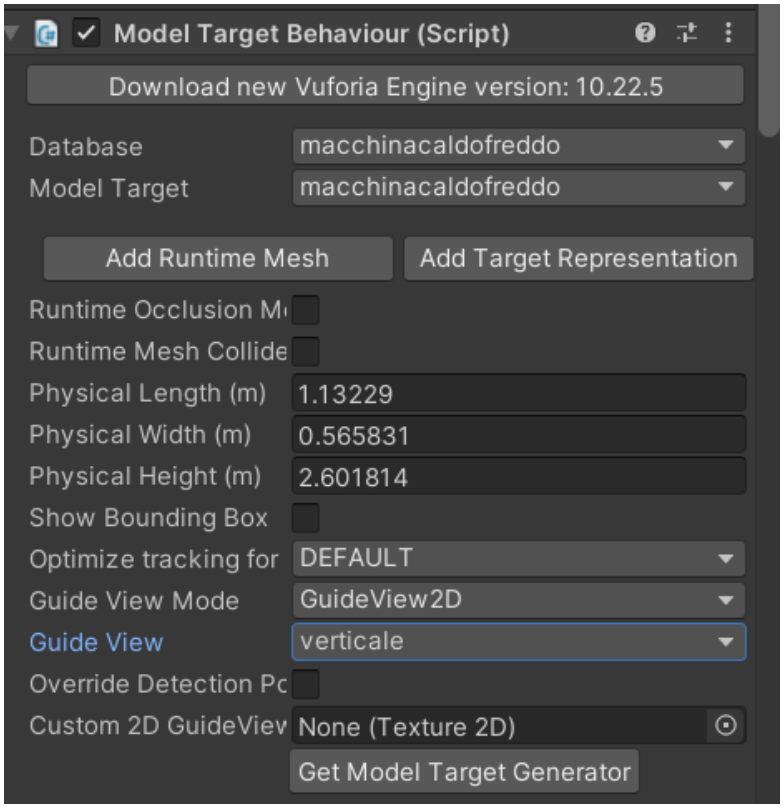

Figura 4.21 Configurazione Model Target

Al termine della configurazione è disponibile nella scena il modello, come illustrato in Figura 4.22. Essendo il Model Target un oggetto esterno agli elementi statici della schermata, non è visibile al centro del riquadro precedentemente creato ma è inserito in basso a destra in dimensioni molto ridotte. Per visualizzarlo basterà selezionarlo sulla gerarchia e premere il tasto “f” della tastiera. Nonostante in fase di sviluppo la sagoma non sia visibile al centro del riquadro, questa verrà posizionata al centro dello schermo in fase di esecuzione.

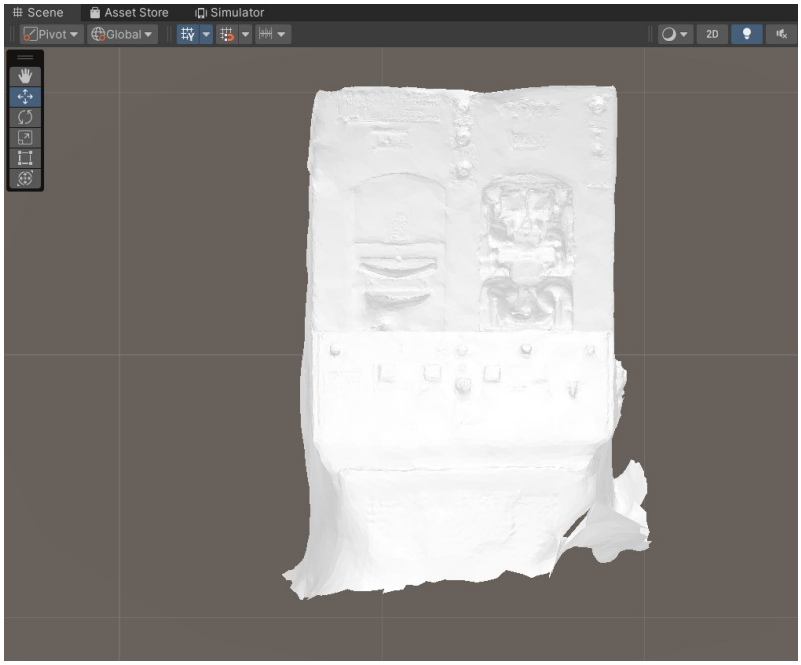

Figura 4.22 Model Target vuoto in Unity

Nel progetto l’obbiettivo è mostrare informazioni aggiuntive del macchinario in realtà aumentata; quindi, sono stati aggiunti nella gerarchia gli elementi che devono essere visualizzati. Questi elementi, per essere visualizzati dopo il riconoscimento, devono essere figli del Model Target nella gerarchia. Gli elementi sono degli oggetti 3D. Nel progetto sono stati usati i piani predefiniti in Unity, con impostata un’immagine specifica per ognuno. Un esempio del modello con gli elementi aggiuntivi è mostrato in Figura 4.23.

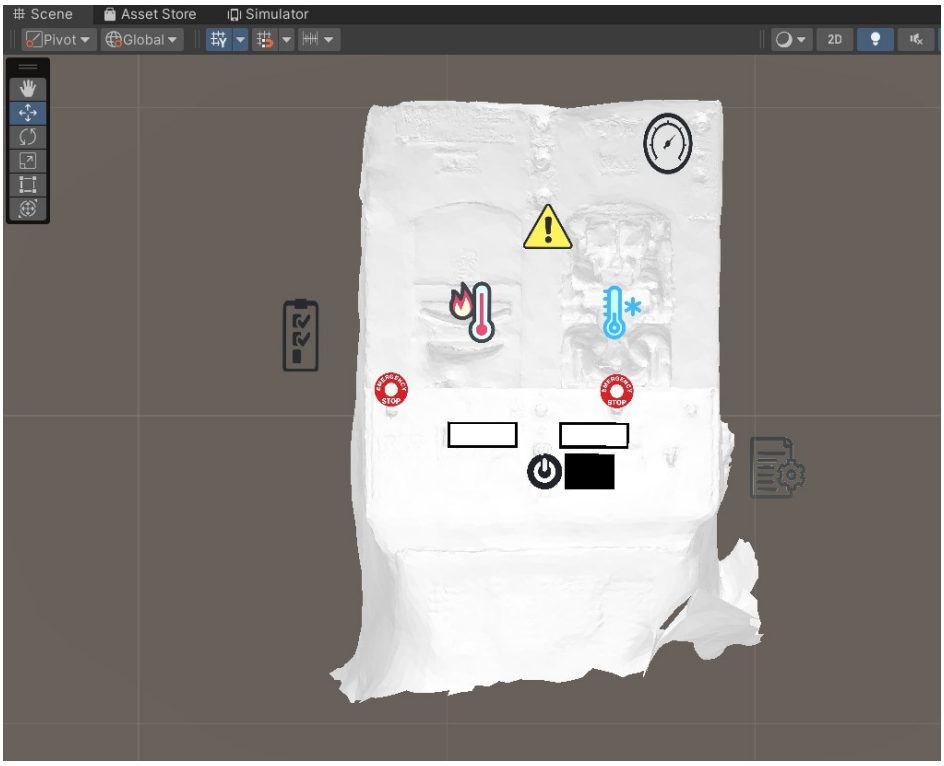

Figura 4.23 Model Target con elementi aggiuntivi

Aggiunti alcuni elementi al modello in scena, è possibile vedere i primi risultati del lavoro fino ad ora effettuato. La Figura 4.24 mostra un'istantanea acquisita dall'applicazione in cui è visibile la scena prima del riconoscimento del modello del macchinario.

Figura 4.24 Screenshot applicazione pre-riconoscimento

Invece, nella Figura 4.25, si vede cosa mostra l'applicazione una volta che il macchinario è stato riconosciuto: è scomparsa l'immagine di riferimento e sono apparsi gli elementi aggiunti al modello.

Figura 4.25 Screenshot macchinario scansionato

Ogni elemento grafico, se attivato, mostra un'informazione aggiuntiva predefinita o proveniente da una richiesta online. Nella Figura 4.26 vengono mostrate le informazioni contenute negli elementi con cui l'utente può interagire. Gli elementi vengono resi visibili grazie ad uno script che rileva l'interazione sul piano e modifica la visibilità degli elementi

nascosti. Tramite questa interazione, utilizzando il bottone con il simbolo dell'interruttore, è possibile inviare una richiesta HTTP che memorizza sul server un messaggio di stato.

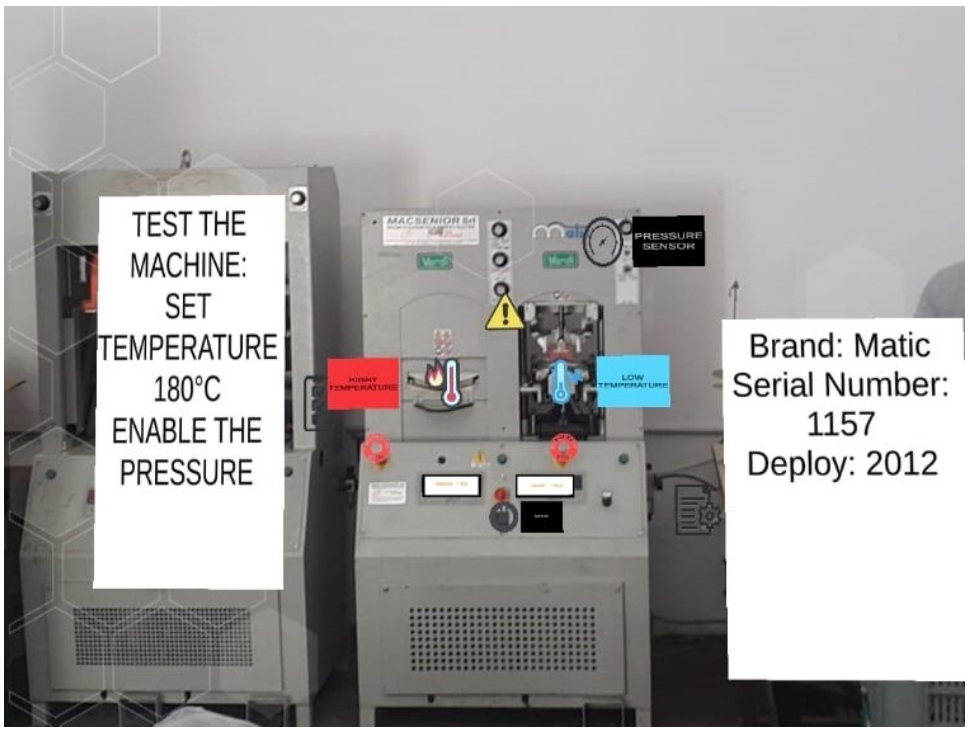

Figura 4.26 Screenshot informazioni visibili

Conclusioni

In questa tesi è stato proposto un possibile processo di progettazione e sviluppo di un'applicazione mobile che sfrutta le funzionalità di realtà aumentata per trasformare un ambiente di lavoro comune in un ambiente di lavoro immersivo in cui il lavoratore, sfruttando la tecnologia, ha la possibilità di comunicare al meglio con esso, ampliando le sue conoscenze con le informazioni aggiuntive provenienti dalla rete. Il punto di partenza è stato l'analisi del progetto "HOMEY" e dei suoi obbiettivi, per poi effettuare un'analisi del contesto e della tecnologia da utilizzare.

Successivamente sono stati elencati i software utilizzati per sviluppare il progetto nelle sue varie componenti. Grazie a questi strumenti, avendo chiare le loro funzioni, è stato possibile iniziare la progettazione dell'applicazione in tutte le sue componenti. L'applicazione è stata suddivisa in due componenti principali: l'applicazione mobile e il server che comunica con essa. Infine, in fase di implementazione, sono state mostrate tutte le componenti implementate a partire da zero. Infatti, è stata mostrata la creazione del modello, lo sviluppo dell''applicazione e come essa interagisce con il server.

Essendo una proposta di soluzione per obbiettivo proposto dal progetto "HOMEY", essa lascia infinito spazio per essere ripresa e modificata. L'aspetto visivo può essere sicuramente migliorato o cambiato per rendere alcuni elementi dell'interfaccia grafica più piacevoli all'esperienza dell'utente mentre, per quanto riguarda le funzionalità, si può ampliare la parte di invio di dati dall'applicazione al server.

Complessivamente, lo sviluppo di un ambiente di lavoro immersivo può dirsi completato con successo. Gli obiettivi prefissati sono stati tutti portati a compimento trovando, nel momento in cui si presentava un problema, sempre una soluzione per risolverlo. Inoltre, il progetto è stato funzionale ed istruttivo a livello personale per approfondire la conoscenza su una nuova tecnologia che si sta sviluppando sempre di più in molti ambiti e che in futuro sarà cruciale. Conoscere una tecnologia di questo tipo è un importante vantaggio in un mondo in continua evoluzione.

Bibliografia

[1] Yue Yin, Pai Zheng, Chengxi Li, Lihui Wang. “A state-of-the-art survey on Augmented Reality-assisted Digital Twin for futuristic human-centric industry transformation”. Robotics and Computer-Integrated Manufacturing, 81: 102515, Elsevier, 2023

[2] Ivan E. Sutherland, “The Ultimate Display”, Information Processing Techniques, In 1965 International Federation for Information Processing Congress, pages 506-508, 1965

[3] Luís Fernando de Souza Cardoso, Flávia Cristina Martins Queiroz Mariano, Ezequiel Roberto Zorzal. “A survey of industrial augmented reality”. Computers & Industrial Engineering,139: 106159, Elsevier, 2020

[4] Tipi di realtà aumentata: https://www.geopop.it/cose-la-realta-aumentata-come-funziona-e-qualche-esempio-di-applicazione/

[5] Vuforia: https://developer.vuforia.com/

[6] Unity: https://unity.com/

[7] Polycam: https://poly.cam/library

[8] Blender: https://www.blender.org/

[9] Visual Studio: https://visualstudio.microsoft.com/it/

[10] GitHub: https://github.com/

[11] Nuget: https://www.nuget.org/

[12] Postman: https://www.postman.com/

[13] HTTP: https://it.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[14] Linee guida sulle caratteristiche dell'oggetto: https://developer.vuforia.com/library/model-targets/model-targets-supported-objects-cad-model-best-practices

[15] C#: https://it.wikipedia.org/wiki/C_sharp

[16] Python: https://www.python.org/

[17] Newtosoft: https://www.newtonsoft.com/json

[18] WebsocketSharp: https://github.com/sta/websocket-sharp

[19] Websocket: https://it.wikipedia.org/wiki/WebSocket

[20] Flask: https://flask.palletsprojects.com/en/3.0.x/