**POLITECNICO DI MILANO**

**Corso di Laurea Magistrale in Ingegneria Informatica**
**Dipartimento di Elettronica, Informatica e Bioingegneria**


# Soluzione alla realizzazione di tour immersivi per dispositivi di Realtà Virtuale

**Relatore: Prof. Luciano Baresi**

**Tesi di Laurea di: Alex Piermatteo**
**Matricola: 804788**

**Anno Accademico 2014-2015**

# Ringraziamenti

Non sono particolarmente bravo nel fare discorsi ma arrivato alla conclusione di questo percorso di studi sento il bisogno di dire grazie ad alcune persone.
Per cominciare, il professor Luciano Baresi che mi ha lasciato completa libertà nello svolgimento della tesi e ha permesso questo lavoro che racchiude perfettamente le mie passioni.

Un ringraziamento davvero speciale va poi ai miei genitori, perchè hanno reso questo traguardo possibile ma soprattutto perchè mi hanno dato preziosi consigli di vita che porterò sempre con me.

Un immenso grazie ai miei amici dell'università, che hanno condiviso con me le soddisfazioni e le difficoltà di questo corso di studi. Con voi ho trascorso momenti indimenticabili e spero che ne passeremo molti altri insieme.

Grazie agli amici di Milano e di Pescara, con cui sono rimasto legato e che mi auguro di avere sempre al mio fianco. Nello specifico ringrazio Fabio, il fratello che non ho mai avuto.

Infine la ragazza che amo, Chiara. Tu mi hai sempre sostenuto, conosci ogni mio segreto e senza di te non sarei certamente la persona che sono ora.

Un ultimissimo ringraziamento va anche ai ragazzi del Mogamo, che con la loro ottima birra hanno alleviato le fatiche di questa carriera universitaria.

# Indice

# Elenco delle figure

# Sommario

Le possibilità aperte con l'utilizzo dei dispositivi mobili connessi ad internet come mezzi di informazione reperibile da ogni luogo e in qualsiasi momento, ha permesso all'utenza mondiale di diventare sempre più curiosa ma anche più esigente riguardo le informazioni di cui ha bisogno. Inoltre grazie anche all'avvento di nuove tecnologie e framework per il web molto potenti, l'attenzione dell'utente sembra essersi spostata verso il voler sentirsi parte di ciò che si ricerca in internet, favorendo le esperienze sempre più coinvolgenti ed interessanti.
Non a caso grandi aziende tecnologiche, come ad esempio Google, hanno cercato di portare spesso nuove idee per fornire informazioni agli utenti in maniera sempre più innovativa. E' ripartendo dalla voglia di innovare il modo di fare informazione, attraverso il web e i dispositivi di nuova generazione, che questo lavoro si è concentrato su un argomento attuale e sempre più in uso in questi ultimi anni, sto parlando dei *Tour Virtuali* e delle *esperienze immersive*. L'obiettivo non è solo quello di studiare le possibilità offerte attualmente per l'utenza comune dalle piattaforme che propongono l'esplorazione di panorami a 360 gradi e di tour virtuali. Il punto focale infatti è la documentazione dello sviluppo di un servizio nuovo, totalmente gratuito e di facile utilizzo che è stato prototipato per questo lavoro di tesi, e che permette la creazione di tour virtuali in totale autonomia con il semplice utilizzo di uno dispositivo mobile e di una web application.
Vi è poi una seconda parte del progetto molto importante che vede invece protagonista un dispositivo di nuovissima generazione e lo sviluppo della sua applicazione dedicata, il *Samsung Gear VR*, che è stato utilizzato come possibile client per la visualizzazione totalmente immersiva dei tour virtuali presenti sulla piattaforma realizzata. Questo secondo sviluppo non si pone

solo come semplice costruzione di un client alternativo al primo, ma tenta di portare innovazione e di fornire uno studio su quanto sia possibile uno scambio di dati, anche pesanti, con il server della piattaforma web costruita, per il download e la visualizzazione istantanea dei tour virtuali, offrendo però una buona usabilità senza rallentamenti. Interazione che il parco di applicazioni concorrenti visionate, al momento, non offrono.
Infine è stata svolta una valutazione qualitativa sul lavoro compiuto rapportandolo con i competitors attualmente in circolazione. Da questa analisi è emerso che la piattaforma realizzata è da considerarsi una valida proposta date le sue nuove funzionalità ed il supporto a dispositivi immersivi capaci di interfacciarsi con essa.

# Capitolo 1

# Introduzione

Le possibilità aperte con l'utilizzo dei dispositivi mobili connessi ad internet come mezzi di informazione reperibile da ogni luogo e in qualsiasi momento, ha permesso all'utenza mondiale di diventare sempre più curiosa ma allo stesso tempo anche più esigente riguardo le informazioni di cui ha bisogno. Non vi è più la sola necessità di una ricerca di dati testuali con l'unico obiettivo di apprendere qualcosa di nuovo ma adesso, grazie anche all'avvento di nuove tecnologie e framework per il web molto potenti, l'attenzione dell'utente si sta spostando verso il voler sentirsi parte di ciò che si ricerca in internet. A tal scopo, da sempre, uno dei metodi più utilizzati per il coinvolgimento degli utenti è stata proprio la comunicazione visiva che con l'aiuto di immagini e video ha avuto sempre un ruolo centrale.
Ma è in questo momento di grande quantità di stimoli visivi che si sta cercando di far evolvere e abbandonare le soluzioni tradizionali a favore di nuovi strumenti di comunicazione. Non a caso grandi aziende tecnologiche, come ad esempio Google, hanno cercato di portare spesso nuove idee per fornire informazioni agli utenti in maniera sempre più innovativa, basti pensare a Street View che permette esplorazioni di scenari virtuali da ogni parte del mondo. E' ripartendo da Street View e dalla voglia di innovare il modo di fare informazione, attraverso il web e i dispositivi di nuova generazione, che questo lavoro si è concentrato su un argomento attuale e sempre più in uso in questi ultimi anni, sto parlando dei *Tour Virtuali* e delle *esperienze immersive*.

I Tour Virtuali sono l'evoluzione dell'immagine statica che poneva lo spettatore di fronte a un punto di vista predefinito e poco coinvolgente. L'obiettivo di questi invece è quello di rendere lo stesso spettatore padrone della scena e della navigazione all'interno di essa. Sono costituiti da una sequenza di video o di immagini a 360 gradi totalmente esplorabili, ma posso utilizzare anche altri elementi multimediali come suoni o testi di accompagnamento.
I Tour Virtuali hanno diverse possibili applicazioni pratiche ma ad oggi sono particolarmente usati nelle università, per l'apprendimento e nell'industria delle agenzie immobiliari, inoltre, anche grandi società stanno cominciando ad utilizzare questa tecnologia per incrementare la visibilità della propria azienda o semplicemente per attrarre nuovi clienti all'uso dei loro servizi. Questo perchè i tour virtuali si prestano molto bene per l'obiettivo di migliorare un business e secondo diversi studi il loro impiego porta a buoni risultati e vantaggi. In generale questi benefici derivano dal fatto che una foto panoramica a 360 gradi enfatiza meglio le particolarità di un luogo e produce da subito una buona impressione attraverso la visualizzazione. Entrando più nel dettaglio e studiando alcune casistiche specifiche sono emersi i seguenti risultati.

Nel campo universitario, recenti studi mostrano che gli studenti spesso prendono vantaggio da questi tour virtuali. YouVisit [24], una compagnia leader nel settore dei tour virtuali affiliata con 150 università e college in tutto il mondo, produce più di 2 milioni di visitatori l'anno sul proprio sito. Uno studio portato avanti proprio da YouVisit mostra come gli studenti alla domanda se reputassero importante avere come mezzo un tour virtuale per giudicare la scelta di un campus universitario, con il 36% abbiano risposto in maniera affermativa mentre un altro 40% li definisca come qualcosa di cui non si può fare a meno. Non a caso infatti tantissime università di prestigio in tutto il mondo come Yale, Harvard, Nuova Accademia delle Belle Arti di Milano o Cambridge hanno inserito all'interno del proprio sito dei tour virtuali. In accordo con il CEO di YouVisit Abi Mandelbaum infatti, le università e i college con cui la sua compagnia ha lavorato hanno riportato in media una crescita del loro numero di studenti stranieri del 35% ed un incremento delle visite in giornata all'anno del 30%.

Nel campo dell'educazione invece, importanti passi avanti nell'uso della Realtà Virtuale sono stati fatti con l'avvento dei *Virtual field trip*, vere e proprie gite scolastiche svolte direttamente in classe grazie a questa tecnologia innovativa. Un Virtual field trip per bambini delle elementari fornisce l'opportunità di esplorare e vedere posti e persone che tipicamente non si osserverebbero in una classica giornata di scuola. Gli studenti possono esplorare e viaggiare per i paesi senza realmente lasciare la loro classe. Soluzioni di questo tipo hanno un grosso impatto economico per le scuole poichè vengono abbattuti i costi di trasporto o di eventuali assicurazioni. Inoltre gli insegnanti hanno a disposizione una immensa quantità di informazioni da cui trarre spunto per l'insegnamento, sono nate infatti numerose piattaforme ed è facile trovare risorse semplicemente digitando *virtual field trip for elementary learners* sul web.

Anche le ricerche portate avanti da alcune delle migliori Agenzie immobiliari in tutto il mondo mostrano i benefici dei tour virtuali.

*Benefici per Real Estate*: I potenziali clienti possono giudicare una abitazione in una maniera più approfondita rispetto all'utilizzo di una normale fotografia poichè un tour virtuale rende più semplice da capire il posto in oggetto, inoltre i clienti che chiamano i proprietari di queste abitazioni sono spesso più informati e generalmente più interessati. Anche il proprietario ha il vantaggio di poter far visionare una casa 24h su 24 senza dover andare di persona e spendere soldi per le pulizie di visita.

*Benefici per Hotel/Resort*: I clienti decidono più in fretta se il tour virtuale ha fatto una buona impressione. E' infatti provato che il 73% delle persone che cerca un posto per le vacanze online visita almeno 3 o 4 siti web prima di decidere, con tour di questo tipo si ha la possibilità di rendere questa decisione più veloce.

Altre statistiche di queste agenzie hanno evidenziato che:

- Le case che avevano un tour tra le possibili visualizzazioni hanno ricevuto un 40% di visite in più;

- Il 75% degli utenti che sono interessati a case da comprare o affittare hanno dichiarato che preferiscono i tour alle classiche foto;

- Gli hotel che dispongono di un tour virtuale sul loro sito generano il 48% di prenotazioni in più rispetto a chi non ne ha uno, con un aumento dell'85% di booking online;
- Gli utenti che visitano siti che hanno al loro interno tour virtuali rimangono in media 3x volte in più sul sito web;
- I tour virtuali riducono il tempo perso per un affitto o per una vendita del 40%;

Il lavoro di tesi, come accennato in precedenza, si inserisce in questo contesto dei Tour Virtuali, con una attenzione particolare verso l'utenza comune, e quindi non professionale, interessata ad un utilizzo di questa tecnologia per la promozione di eventi o attività di vario genere. Attraverso lo studio delle piattaforme attualmente offerte si è infatti notato come non sia assolutamente facile per una persona comune realizzare da zero un nuovo tour virtuale da rendere disponibile per la visualizzazione sul web. Per questo motivo, come si vedrà in questo documento, la prima fase del progetto riguarderà lo sviluppo di un servizio nuovo, totalmente gratuito e di facile utilizzo che è stato prototipato e che permette la visualizzazione e sopratutto la creazione di tour virtuali in totale autonomia con il semplice utilizzo di uno dispositivo mobile di nuova generazione e di una web application.
Vi è però una seconda parte del progetto che vede invece protagonista un dispositivo di nuovissima generazione e lo sviluppo della sua applicazione dedicata, il Samsung Gear VR. La decisione di utilizzare un visore per la Realtà Virtuale è venuta dall'interesse che si sta mostrando attualmente per questo genere di prodotti. Il dispositivo in questione è stato utilizzato come possibile client per la visualizzazione totalmente immersiva dei tour virtuali presenti sulla piattaforma web realizzata. Questo secondo sviluppo non si pone solo come semplice costruzione di un client alternativo al primo, ma porta con sé la volontà di studiare quanto sia possibile uno scambio di dati, anche pesanti, con il server della piattaforma web costruita, per il download e la visualizzazione istantanea dei tour virtuali, offrendo però una buona usabilità senza rallentamenti.
La scelta del Samsung Gear VR, rispetto ad altri possibili dispositivi di Realtà Virtuale, è stata dettata da una maggiore performance offerta e dalla sua

capacità di essere utilizzato anche in mobilità grazie alla rete 4G del telefono che il dispositivo utilizza al suo interno per funzionare. Per lo sviluppo del suo applicativo, come si vedrà più avanti, si è scelto il framework di Unity, un IDE molto potente e ben supportato dalla comunità di sviluppatori.

Per quanto riguarda la stesura della tesi il lavoro è suddiviso tra i seguenti capitoli:

*Il secondo capitolo* spiega il contesto in cui il progetto di tesi si inserisce cercando di dare una panoramica al lettore il più esaustiva possibile. Successivamente vengono presi in esame i vari Tool e Servizi già esistenti per la creazione di tour virtuali e si continua discutendo ad alto livello sui vari dispositivi per la realtà virtuale presenti sul mercato e le motivazioni che hanno spinto a scegliere il Gear VR e Unity per lo sviluppo della sua applicazione nativa.

*Il terzo capitolo* spiega il servizio proposto con il prototipo, le differenze con lo stato dell'arte e gli obiettivi che si pone. Inoltre vi è una descrizione generale del sistema con un occhio di riguardo verso le tecniche che hanno permesso la comunicazione del server con client molto diversi tra loro come la Web application e il GearVR.

*Il quarto capitolo* spiega in dettaglio come è stata realizzata l'applicazione web che permette di caricare le immagini multimediali 360, di generare il tour virtuale e di arricchirlo con informazioni testuali in ogni scena. Inoltre vi è anche la descrizione di altre funzionalità quali ad esempio la gestione degli utenti o dei media caricati.

*Il quinto capitolo* descrive lo sviluppo portato avanti per il GearVR con la descrizione del framework di Unity e le sue caratteristiche. Nella seconda parte invece si discute dei problemi di debug, sulla velocità di caricamento delle scene riscontrati e le successive ottimizzazioni apportate.

*Il sesto capitolo* offre una visione generale di quanto prodotto con questo lavoro di tesi, vengono mostrati alcuni esempi significativi di utilizzo delle interfacce sviluppate, accompagnate da descrizioni esplicative.

*Il settimo ed ultimo capitolo* propone una valutazione qualitativa del lavoro svolto rapportandolo con i competitors attualmente in circolazione. Infi-

ne illustra delle possibili funzionalità aggiuntive che possono arricchire lo sviluppo di questo lavoro di laurea.

# Capitolo 2

# Stato dell'arte

In questo capitolo si cercherà di spiegare nel dettaglio il contesto generale in cui si inserisce il progetto di tesi, mostrando le varie forme che un tour virtuale può presentare con riferimenti ad alcuni esempi significativi. Subito dopo si andrà ad analizzare lo stato dell'arte dei servizi e tool atti alla creazione di questi tour virtuali sottolineando i loro pregi e difetti. Infine si passerà in rassegna il parco dei dispositivi per la Realtà Virtuale e degli SDK necessari al loro sviluppo.

## 2.1 Le tipologie di tour virtuali

Al fine di poter capire come impostare un servizio per la visualizzazione di tour virtuali è stato necessario prima di tutto studiare quali sono gli standard tecnologici attualmente disponibili in materia, inoltre è stato importante concentrarsi anche sullo studio delle interfacce utente e di come questi visualizzatori apparissero dall'esterno e quali funzionalità avessero. Tutto ciò è servito al fine di trarre delle linee guida utili a definire un modo di operare corretto nella progettazione e di creare un prototipo intuitivo per qualsiasi tipologia di utilizzatore.

### 2.1.1 Web & Application

Per molte attività di business, un tour virtuale deve poter essere accessibile da ogni luogo. E' per questo motivo che una delle migliori soluzioni per

la loro esplorazione è sempre stata una applicazione web. Tuttavia un tour utile e ben fatto non deve mostrare solo una serie di foto panoramiche, una migliore esperienza la si fornisce solo se questo viene accompagnato da una varietà di contenuti multimediali come video, testi, foto o anche suoni.
Vi sono molti modi per collezionare insieme tutti questi formati, qualche anno fa la soluzione più accreditata è stata l'uso di software come Adobe Flash ma negli ultimi tempi ci si sta spostando verso l'utilizzo di tecnologie più leggere e performanti quali Javascript + HTML5 [22]. Questo recente standard web permette infatti di sviluppare dei siti internet che assomigliano a delle vere e proprie applicazioni, fruibili anche attraverso il browser presente sui dispositivi mobile. Attraverso questo approccio la logica applicativa solitamente si trova su un server remoto e viene riproposta nel browser solamente la logica di presentazione.


*Figura 2.1: Tour Virtuale Web Italo*

Un esempio di tour virtuale multimediale, per il business, che utilizza tecnologia in Flash è quello presente sul sito di *italo.it*. Il tour offre la possibilità di esplorare tutti gli ambienti del treno, con una panoramica sulle tipologie di vagoni e servizi che la compagnia ferroviaria offre durante i suoi viaggi. Prima di ogni scena esplorativa della carrozza c'è un video introduttivo della stessa. I contenuti sono completi in quanto vi sono anche foto e testi che accompagnano l'utente durante la navigazione degli hotspot. C'è anche un selettore di scene da cui è possibile facilmente cambiare

ambientazione e decidere cosa si vuole effettivamente conoscere. Il punto debole risiede però nella tecnologia Flash in quanto rende il tour lento nei caricamenti e non disponibile per smartphone e tablet.

Altro tour online di recente costruzione è quello messo in piedi per *Expo Milano 2015* [19] che permette ai visitatori di esplorare un modello virtuale dell'area che ospita l'Esposizione Universale. L'applicazione è basata sulla piattaforma 3DEXPERIENCE, software proprietario sviluppato da Dasault Systéms, azienda leader nel settore della modellazione e progettazione 3D per la realtà virtuale. Il tour è ricco di approfondimenti multimediali: video interviste, immagini e photogallery che descrivono i concept e i progetti architettonici dei Paesi partecipanti e dei Cluster, le mostre ospitate nelle aree e gli eventi delle strutture. Tutti i contenuti sono consultabili e condivisibili anche sui social da web, oppure in mobilità da smartphone e da tablet. Una volta caricata l'applicazione web, si apre una mappa 3D in cui è possibile scegliere da quale padiglione o area cominciare l'esplorazione o in alternativa c'è un selettore di scene, che resterà presente per tutto il tour, da cui sarà possibile cambiare ambientazione in ogni momento.

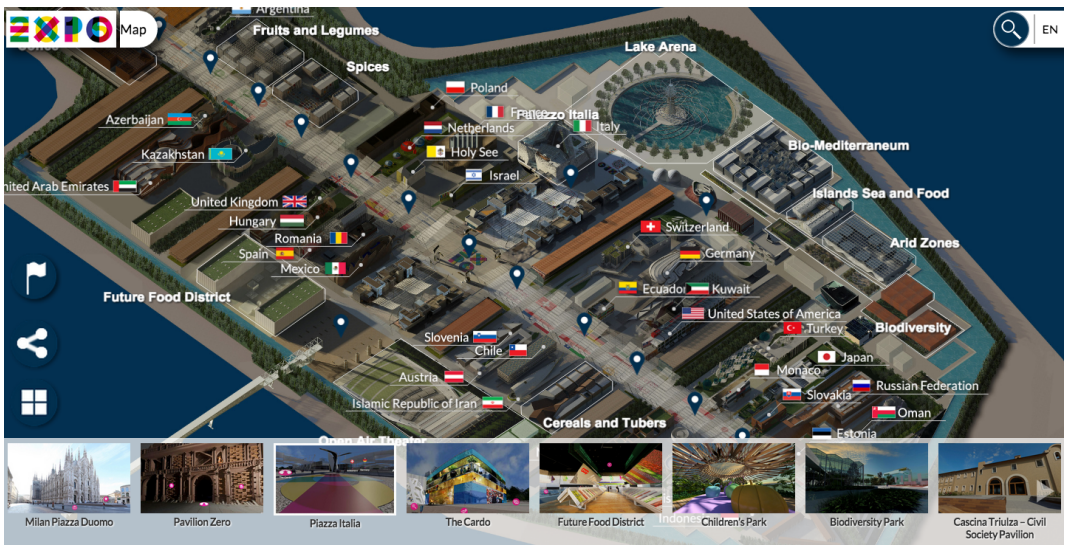


*Figura 2.2: Mappa 3D Expo Milano 2015 e selettore scene*

L'interfaccia per la navigazione dei panorami è molto ricca di contenuti e funzionalità quali ad esempio selezione dei suoni, rotazione automatica del panorama o condivisione sui social di ogni scena. Per quanto riguarda

la multimedialità, come già accennato in precedenza, il tour offre numerosi hotspot informativi diversi che, a differenza del tour presente su italo.com, portano in primo piano sullo schermo tutti i testi, i video e le foto legate al padiglione su cui ci si sta informando, senza però abbandonare la scena. Le tecnologie utilizzate per il visualizzatore sono l'HTML5 e il Javascript, le immagini delle scene non sono dei semplici panorami equirettangolari ma sono un collage di foto quadrate più piccole che messe insieme costruiscono l'intero ambiente in 360.


*Figura 2.3: Expo Milano 2015, visualizzatore web tour virtuale*

### 2.1.2 La realtà immersiva

La realtà virtuale fa parte di quel ramo dell'informatica che ha l'obiettivo di creare dei mondi simulati. Questa può essere definita anche immersiva se da un lato riesce ad immergere lo spettatore al suo interno ma dall'altro gli permette di interagire con il mondo circostante grazie all'utilizzo di dispositivi specifici che cercano di rendere l'esperienza il più reale possibile. Un esempio è dato dai visori di nuova generazione, semplici occhiali in cui gli schermi vicini agli occhi annullano il mondo reale dalla visuale dell'utente. Il visore può inoltre contenere dei sistemi per la rilevazione dei movimenti, in modo che girando la testa da un lato, ad esempio, si ottenga la stessa azione anche nell'ambiente virtuale. Con l'introduzione sul mercato dell'Oculus

Rift [15] e di tutti i seguenti dispositivi di realtà virtuale, anche i tour hanno risentito del cambiamento e sono nate applicazioni che sfruttano queste nuove tecnologie in modo da fornire all'utente un'esperienza sempre più stimolante e coinvolgente.

Un esempio di tour virtuale completamente immersivo è quello offerto dalla *Matterport*, una società con sede in California che sfrutta nuove tecnologie per la creazione di tour virtuali con modellazione 3D degli ambienti. Per modellazione 3D si intende che i panorami dei loro tour non sono delle semplici immagini equirettangolari ma sono dei veri e propri modelli tridimensionali in cui l'utente viene posto all'interno con la possibilità di esplorarli. Hanno sviluppato un applicativo, di nome Matterport VR showcase, nativo per GearVR che permette la visita di tre differenti ambienti: il Grand Canyon, la Mirus Gallery di San Francisco ed il loro ufficio nella Silicon Valley. L'applicazione non ha nessuna interconnessione con il web ed ha tutte le informazioni e i modelli 3D direttamente incorporati al suo interno, questo la rende molto veloce seppur non dinamicamente scalabile all'aggiunta di nuovi tour virtuali.

*Figura 2.4: Esempio puntamento hotspot Matterport VR Showcase*

All'avvio è possibile scegliere tra uno dei tre scenari disponibili e una volta selezionatone uno si apre il visualizzatore 360 da cui è possibile poi navigare il modello. E' sufficiente infatti puntare con il mirino, posto al cen-

tro dello schermo, uno degli hotspot sospesi all'interno della scena. Così facendo ci si potrà spostare poco più avanti nell'ambiente e continuando in questo modo è possibile esplorare tutto il tour messo a disposizione. Non c'è il bisogno di caricare una foto dopo l'altra per la visualizzazione in quanto il modello tridimensionale è unico e ci si sposta semplicemente al suo interno.


*Figura 2.5: Esempio puntamento hotspot Matterport VR Showcase*

## 2.2 Le tipologie di immagini utili per un tour virtuale

Una volta identificate le tipologie più significative di tour virtuali, il passo successivo è stato quello di capire come poterne realizzare uno da zero, in totale autonomia, avendo poi la possibilità di averlo online sempre disponibile. Prima ancora però, è stato necessario identificare quali varietà di panorami fossero compatibili ed idonei con questo scopo. Le foto panoramiche, o chiamate anche foto a 360 gradi, esistono da più di un secolo. Vi sono esempi di questo genere di immagini risalenti infatti a più di 130 anni fa. Ai giorni nostri però hanno avuto un grosso incremento del loro utilizzo nella tecnologia grazie alla facilità con cui possono essere prodotte. Un qualsiasi telefono di fascia media infatti è adesso capace di fare foto panoramiche di discreta qualità, e cosa più importante su internet questo genere di immagine è diventato di facilissima reperibilità.

In termini di formato vi sono tre principali tipologie di panorami:

*Panorama parziale*, sono la prima tipologia che è stata creata. Vengono assemblate foto di grandezza normale una di fila all'altra per creare un'immagine continua con un angolo di visuale maggiore rispetto a quello che si potrebbe fotografare con una lente normale. Non vengono utilizzate nel campo dei tour virtuali perchè non completano interamente una scena.

*Panorama cilindrico*, è possibile utilizzarla per un tour virtuale poichè completano un giro a 360 gradi del panorama circostante. Tuttavia non catturano al loro interno le informazioni relative alla parte superiore ed inferiore della realtà che vogliono immortalare. Il procedimento di creazione è lo stesso di quelle a panorama parziale con la differenza che in queste la prima e l'ultima foto di cui è composta si sovrappongono.

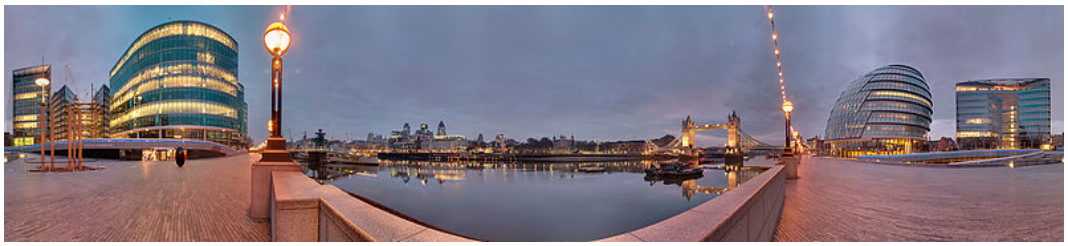

*Figura 2.6: Esempio di panorama cilindrico*

*Panorama sferico (o equirettangolare)*, è la tipologia più completa ed utilizzata in assoluto. Sono perfette per la creazione di tour virtuali in quanto permettono di guardare anche sotto e sopra nella scena. Possono essere costruite sia grazie a particolari fotocamere con lenti a 360 gradi che scattano un'unica foto o anche con la sovrapposizione di immagini separate e poi renderizzate ad hoc.

*Figura 2.7: Esempio di panorama sferico o equirettangolare*

## 2.3 Catalogazione dei servizi e tool esistenti per costruire un tour da zero

Una volta identificato che il panorama equirettangolare è quello più idoneo alla visualizzazione degli scenari, il punto fondamentale diventa capire come si possa realizzare un tour virtuale da zero sfruttando immagini di questo tipo. In questo paragrafo vengono quindi presi in esame i migliori servizi e tool che permettono questo genere di azione e se ne studieranno punti di forza e debolezza.

### 2.3.1 Servizi online

Utilizzando Google come fonte primaria di ricerca di questi servizi, sono moltissimi i siti trovati che permettono di costruire tour virtuali affidandosi ad aziende di piccola o media grandezza che si prendono carico di scattare foto panoramiche e di costruire tour virtuali personalizzati in cambio di un pagamento. Questi però non sono stati presi in considerazione da questo lavoro di tesi in quanto non soddisfano il requisito primario che è quello di costruire questo genere di informazione digitale in completa autonomia.

Per questo motivo la ricerca ha toccato solo quei servizi che permettono di generare un tour virtuale su piattaforme accessibili a chiunque.

Uno dei più significativi in assoluto si chiama *360cities*, ed è il più grande archivio multimediale, geo-referenziato, di foto panoramiche ed interattive create da un network di migliaia di utenti fotografi specializzati da tutto il pianeta. La forza è nella loro comunità che ogni giorno aumenta la galleria multimediale con nuove foto di ottima qualità a 360 gradi. Grazie alla collaborazione con fotografi professionisti, fornisce anche servizi specializzati per la pubblicità, sviluppo di applicazioni e giochi o anche creazione di tour e panorami per l'editoria e film.
L'interfaccia per la navigazione dei panorami è molto minimale, vi sono solo un radar in basso che mette in evidenza la porzione dell'immagine che si sta guardando e delle frecce che mostrano la direzione in cui si può proseguire il proprio tour, con relativa distanza in metri del punto in cui ci troviamo. Nonostante la scarsa presenza di altri elementi multimediali di contorno il tour si apprezza per l'ottima qualità delle immagini e le performance del visualizzatore che può essere facilmente inserito all'interno di altri siti web per mostrare i panorami presenti in piattaforma. Il visualizzatore web inoltre sfrutta le librerie in HTML5 di krPano (Paragrafo 2.3.2).

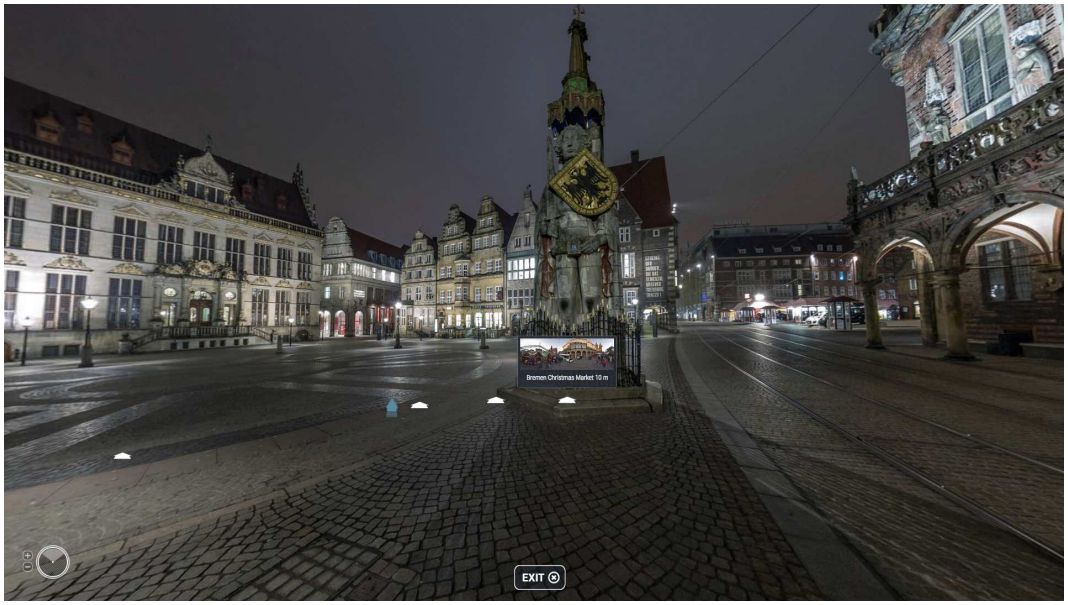


*Figura 2.8: 360cities, visualizzatore web tour virtuale*

L'applicazione di 360cities per dispositivi mobile è anch'essa molto interessante poichè oltre a fornire una migliore gestione della ricerca dei panorami e tour presenti sul sito offre un visualizzatore che permette di osservare l'interno della scena grazie al giroscopio del dispositivo. La parte davvero interessante tuttavia sta nel fatto che i panorami sono tutti *Google Cardboard Ready*[9]. E' infatti possibile cliccare sul simbolo dedicato e il panorama passerà in modalità bioculare per essere inserito all'interno del Cardboard ed offrire così una esperienza immersiva. In questa modalità non è però possibile navigare un tour virtuale ma solo osservare la scena selezionata.

*Figura 2.9: 360cities, mobile in visualizzazione per Google Cardboard*

Nonostante l'ottimo servizio e le migliaia di utenti che lo utilizzano, in 360cities costruire un tour virtuale da zero non risulta particolarmente semplice. Ha regole sulla tipologia e la qualità delle foto molto restrittive, come la grandezza minima, presenza di sbavature nella foto o semplicemente errori nei metadata presenti nella immagine di partenza. Inoltre non è possibile caricare le scene in autonomia ma vanno tutte inviate via mail e fatte approvare con tempi di attesa che possono variare da pochi minuti a qualche giorno. Non è poi possibile editare il proprio tour virtuale in quanto questo viene generato automaticamente dalle foto inviate e che vengono unite in base alla loro geo-localizzazione. Per questo motivo non vi è nean-

che la possibilità di personalizzare la posizione degli hotspot all'interno di una scena dato che non sono inseriti dall'utente. 360cities ha una sua versione free che permette di caricare immagini e di inserire panorami sui propri siti alle condizioni descritte sopra, ma è possibile registrarsi come fotografi professionisti e diventare autonomi nelle pubblicazioni pagando una quota che può arrivare a 179 euro/anno.

*YouVisit* è un altro servizio molto importante per la creazione dei tour virtuali, infatti come è stato anticipato nell'Introduzione (Capitolo 1) di questo lavoro di tesi, la compagnia è leader nel settore della costruzione di tour e realtà virtuale. Hanno collaborazioni con le migliori università del mondo per la realizzazione dei loro prodotti ed utilizzano un software proprietario per la visualizzazione dei contenuti multimediali scritto anche questo in HTML5 e compatibile quindi con tutti i dispositivi mobile. Come 360cities da un anno a questa parte anche i loro panorami sono tutti visualizzabili tramite dispositivi VR (Virtual Reality), ma anche nel loro caso in realtà immersiva non è possibile visionare un tour vero e proprio ma solo osservare una scena selezionata.
I servizi completi per la realizzazione di un tour virtuale o della produzione delle foto panoramiche necessarie, sono a pagamento, tuttavia esiste una versione free del loro 'Tour Builder' che permette di fare l'upload dei propri file multimediali e costruire una versione base di tour senza hotspot ma con un selettore di scene laterale che unisce i panorami. Una volta completato l'editing è possibile o condividere il lavoro o anche inserirlo all'interno di altre pagine web.

### 2.3.2 Software

Oltre ai servizi online che direttamente permettono l'inserimento più o meno difficoltoso delle proprie foto panoramiche all'interno delle loro piattaforme, esistono alcuni software specifici che forniscono la possibilità di modificare le proprie foto per renderle compatibili con i visualizzatori a 360 o costruire direttamente tour virtuali di vario genere. Il problema è che spesso il loro utilizzo non è particolarmente immediato in quanto l'utente si ritrova a dover gestire del codice da modificare o inserire autonomamente all'inter-

no di un proprio spazio web. Inoltre, ad oggi, nessuno di quelli davvero performanti è risultato gratuito.

Il software più importante tra i tool attualmente online è sicuramente *krPano* [10], un progetto nato nel 2010 portato avanti da un singolo sviluppatore austriaco di nome Klaus Reinfeld. Anche questo applicativo nasce come semplice player per tour virtuali ma nel tempo, rispetto agli altri, è diventato molto utilizzato grazie ai frequenti update sul software. Il punto di forza risiede nel fatto che il visualizzatore per tour virtuali, che ha una sua versione sia in Flash sia in HTML5, è molto versatile grazie alle numerose tipologie di panorami che supporta ed inoltre ha delle alte prestazioni nel caricare immagini di scenari molto pesanti.
Questo perchè insieme al player è possibile scaricare (ed eventualmente acquistare con una licenza a parte di 129 euro) *Krpanotool*: una serie di utili strumenti per l'elaborazione dell'immagine. I tools di krPano sono molto diversi e consentono per esempio di trasformare una foto da immagine cubica a equirettangolare e viceversa. L'interesse maggiore tuttavia va rivolto a *kMaketiles*: lo strumento che ha reso kRpano il player più performante per immagini ad alta risoluzione. kMaketiles è un tool per il tiling, ovvero l'operazione di ritaglio delle immagini ad alta risoluzione che vengono suddivise in tante piccole parti (in inglese tiles) per rendere l'immagine disponibile su più livelli e scaricabile attraverso il web con tempi di attesa molto più bassi.

Altro tool interessante all'interno di krPano è sicuramente quello legato alla creazione dei tour virtuali. E' infatti possibile costruire un tour da zero facendo uso degli script presenti. Basta far partire l'esecuzione di tali script passando come parametri le immagini che si intende usare per il tour, in automatico questo genererà una cartella con al suo interno una serie di file necessari al funzionamento. Tra questi vi è un file .xml dove al suo interno vi sono tutti i parametri del tour come il numero degli hotspot e la loro posizione, le immagini presenti e le loro info, le varie modalità di visualizzazione e altri settaggi.


*Figura 2.10: krPano, editor software*

Per modificare il tour appena generato viene inserito all'interno anche un editor, che con l'utilizzo di una UI grafica non fa altro che modificare questo file .xml per la gestione di hotspot e l'inserimento di altre funzionalità che di base non vengono inserite automaticamente. Il software è quindi in definitiva molto potente ed ha come uniche problematiche il fatto che non può essere utilizzato senza una regolare licenza in versione gratuita e che alla realizzazione del tour virtuale questo non sia hostato da nessuna parte ma debba essere caricato separatamente sul proprio sito web.

## 2.4 I dispositivi VR: tipologie e la scelta del Gear

Dallo studio dei diversi servizi e tool presenti sul mercato in questo momento per la creazione di tour virtuali e dalle analisi svolte sulle loro funzionalità è emerso che il mondo della Realtà Virtuale sta prendendo sempre più importanza. Questo anche grazie alla buona risposta dell'utente che è interessato ad un maggior coinvolgimento immersivo con l'uso delle tecnologie VR. Ai fini di questo lavoro di tesi, che si pone l'obiettivo di portare innovazione in questo campo legato ai tour virtuali, è sembrato quindi scontato dover affiancare alla propria piattaforma online anche un dispositivo di Realtà Virtuale per la visualizzazione immersiva dei tour. Per fare que-

sto però era necessario studiare quali visori fossero al momento disponibili, come fossero costruiti e quali vantaggi e svantaggi portassero con loro, in modo da scegliere il più adatto ai fini del progetto.

Essenzialmente vi sono due grandi macroaree legate a questi dispositivi, quelli a basso costo che si possono quasi costruire in autonomia con materiali semplici come cartone, un paio di lenti e del velcro, e quelli che invece costano centinaia di euro giustificati da una maggiore attenzione ai materiali e alla qualità dell'hardware inserito al proprio interno. Da un primo punto di vista, avere un dispositivo a basso costo accessibile a tutte le fasce di utente può portare a notevoli vantaggi legati alla distribuzione del proprio applicativo software. L'applicazione o il servizio in questione potrebbe venire usata da un vasto numero di persone interessate a spendere una piccola cifra, nell'ordine dei 20 dollari, per provare una nuova esperienza immersiva.
Le soluzioni a basso costo possono quindi essere allettanti poichè all'apparenza entrambe le tipologie svolgono le stesse funzioni ma scendendo nel dettaglio vi sono differenze più importanti da considerare.

Per iniziare, la prima importante differenza tra dispositivi come i Google Cardboard e quelli di fascia più alta come il Samsung Gear VR risiede nel sensore di tracking. Nonostante entrambe le soluzioni per funzionare necessitano di un telefono all'interno, nel caso del Gear VR il dispositivo riceve dal sensore una spinta di informazioni più dettagliate non indifferente che portano ad un impatto significativo sulla qualità dell'esperienza. Infatti il sensore è più veloce nell'aggiornare gli input ed è anche più preciso al minimo cambiamento di posizione. Secondo le recensioni di chi ha usato spesso i Google Cardboard, questi risultano ottimi per una prima visualizzazione ma alla lunga il tracking comincia a perdere fluidità di calcolo rendendo l'esperienza utente meno coinvolgente.
Stesso discorso vale per i sensori del trackpad: nei dispositivi VR non vi è la possibilità di toccare lo schermo del telefono con le dita. Nel Gear VR infatti sono stati posti a lato dei pulsanti ed un trackpad che servono proprio a questo scopo. Google è l'unica, tra le aziende che hanno adottato una soluzione a basso costo, che al momento invece ha ovviato a questo problema inserendo un magnete sul lato del suo Cardboard che funge da bottone.

Una volta che infatti viene mosso il magnete il telefono, che utilizza il magnetometro, riceve un input e considera il movimento come un click.
Per queste ragioni appena discusse, in questo lavoro di tesi, è stata preferita una soluzione più ad alto livello legata quindi alla possibilità di avere una esperienza più precisa e con più ampie potenzialità di sviluppo. Tutto questo anche se a discapito del parco di dispositivi compatibili in quanto il Gear VR, a differenza dei Google Cardboard, supporta i soli Galaxy Note 4 e il Galaxy S6 al suo interno.


*Figura 2.11: Samsung Gear VR e Oculus Rift DK2*

Facendo un confronto invece tra i dispositivi VR che necessitano di un device esterno come schermo rispetto a quelli classici collegabili al PC, sono state trovate altre differenze che adesso verranno spiegate di seguito.
Tra i dispositivi mobile è stato preso in considerazione il Samsung Gear VR in quanto il più avanti a livello tecnologico e soprattutto l'unico attualmente disponibile in Italia, per i dispositivi invece classici collegabili al PC la scelta non poteva che ricadere sull'Oculus Rift DK2, dispositivo che nel suo campo attualmente non ha rivali. Cominciando dal software va detto che i sistemi sono totalmente equiparabili in quando è stata proprio la Rift a fornire la tecnologia per la realtà virtuale a Samsung, è per questo motivo non vi sono grosse differenze sotto questo punto di vista se non nella grafica

dell'interfaccia interna.
Il Gear VR grazie al Galaxy Note 4 al suo interno possiede però un display più definito e dispone del supporto ai suoni senza la necessità di dover per forza collegare un auricolare esterno come succede per l'Oculus. A livello di prezzo però la versione per PC ha un costo più accessibile di 280 euro mentre il Gear VR costa 199 euro più l'obbligo, se non lo si ha già a disposizione, di comprare il telefono compatible ad un costo di circa 549 euro. Tuttavia la cosa più importante che pone il Gear VR in vantaggio è la sua possibilità di avere il Wireless e la rete 4G incorporate, che permette di poter usare il dispositivo in totale mobilità e non vincolandolo ad un Computer a cui doverlo collegare per farlo funzionare in rete.

L'idea del progetto vede l'utilizzo dei dispositivi mobile per la visualizzazione dei contenuti virtuali, per la creazione delle foto panoramiche ed anche del loro upload sulla piattaforma. Per questo motivo la scelta di un dispositivo utilizzabile in totale mobilità, che implementa lo stesso telefono con cui si realizzano i tour virtuali anche per funzionare, ha posto il Gear VR come la scelta più idonea per il lavoro di tesi.

## 2.5 GearVR: SDK e possibili framework a confronto

Lo sviluppo software di applicativi per dispositivi VR non è di certo banale. Vi sono da tenere in considerazione molti fattori relativi alla stereoscopia delle immagini, al rendering, alla latenza dei frame ed altre accortezze sempre relative al fatto che l'utente è al centro di un ambiente che deve sembrare il più immersivo e realistico possibile. Ad aumentarne la difficoltà, è da tenere anche in considerazione che, nel caso specifico del Gear VR, si sta sviluppando per un dispositivo mobile e per questo motivo si devono considerare molti fattori legati all'utilizzo della batteria, alle prestazioni limitate del dispositivo, anche se il Note 4 a dire il vero è un dispositivo molto performante, e soprattutto alla gestione della memoria RAM che le applicazioni per dispositivi di Realtà Virtuale stressano particolarmente. Per venire in aiuto agli sviluppatori che intendono approcciare questo mondo attraverso

il visore di Samsung, sono state messe a disposizione da Rift due tipologie di framework molto diversi tra loro, che presentano alcuni vantaggi e svantaggi con il loro utilizzo. Questi SDK permettono di gestire in automatico alcune delle accortezze di cui è si è parlato in precedenza liberando così lo sviluppatore da diverse problematiche e fornendo invece molte funzionalità che agevolano lo programmazione verso queste piattaforme.

Il primo framework [14] messo a disposizione è relativo allo *sviluppo nativo* su piattaforma Android. Questo SDK è relativamente rudimentale, ma è anche più vicino all'hardware e permette di implementare esperienze di realtà virtuale ad altissime prestazioni senza il sovraccarico che alcuni ambienti più elaborati come Unity comportano. Non è molto ricco di funzinalità ma fornisce le infrastrutture di base per partire con un proprio sviluppo di realtà virtuale ad alte prestazioni. Per il suo utilizzo viene fornita una libreria di partenza scritta in C++ [18] che al suo interno ha tutte le funzioni che gestiscono l'hardware, inoltre Rift fornisce anche un Template di base da cui partire per lo sviluppo in questa modalità o per prendere semplicemente spunto. Per funzionare oltre al classico SDK Android [8] è necessario scaricare anche l'NDK, un'altra libreria che serve a fare da ponte tra il linguaggio Java in cui è scritta l'app, con la libreria VrLib in C++ che fornisce l'azienda. Il meccanismo di interconnessione è semplice: il file Java *VrTemplate/src/oculus/MainActivity.java* gestisce il caricamento della libreria nativa VrLib, subito dopo questa chiama *nativeSetAppInterface()* per consentire al codice C++ di registrare la sottoclasse della VrAppInterface che definisce a sua volta l'applicazione. Da quel momento grazie a questa interfaccia è possibile mettere in contatto App e libreria.
I problemi di questa libreria sono legati alla disinformazione e al poco supporto che la accompagna in quanto, nonostante Rift abbia un documento riguardante il setup iniziale per un progetto di tipo nativo, non ha poi nessun'altra documentazione relativa alle funzioni che questa libreria fornisce. L'unico modo per lavorarci e sfruttarla è quello di aprire il codice in C++ e leggere ogni classe cercando di interpretare ogni singola funzione e la sua gerarchia di chiamata. Anche cercando in rete non si trovano risultati legati ad un suo utilizzo da parte degli utenti che si sentono scoraggiati nell'utilizzarla proprio per questo motivo.

| Source Files | Classes / Types | Description |
|---|---|---|
| App.h App.cpp AppLocal.h AppRender.cpp | VrAppInterface App AppLocal | Virtual application interface and local implementation. (Native counterpart to VrActivity). |
| PlatformActivity.cpp | | Native implemnetation for the PlatformActivity. Also implements several other menus. |
| TalkToJava.cpp TalkToJava.h | TalkToJava TalkToJavaInterface | Thread and JNI management for background Java calls. |
| Input.h | VrInput VrFrame | Input and frame data passed each frame. |
| KeyState.cpp KeyState.h | KeyState | Keypress tracking. |

*Figura 2.12: Esempio di alcune classi presenti nella VrLib di Oculus*

A differenza della VrLib però Rift fornisce anche un secondo SDK molto importante legato alla implementazione con *Unity* [20]. Unity è un Game Engine cross-platform sviluppato da Unity Technologies e utilizzato principalmente per lo sviluppo di video games per PC, console oppure dispositivi mobile. Il suo Game Engine è costruito su Mono, una implementazione open-source del .NET Framework ma i suoi linguaggi di programmazione possono essere legati anche al Javascript, C# o il Boo (linguaggio ispirato al Python). Su questo grande IDE (Integrated development environment) di sviluppo Rift ha costruito un 'integration package' di applicazioni di esempio, classi di supporto e oggetti chiamati Prefabs che possono essere inseriti già perfettamente operativi durante lo sviluppo. Un esempio di Prefab è la OVRCameraRig che inserita nel progetto permette di osservare l'ambiente circostante in modalità bioculare stereoscopica senza la necessità di fare altro lavoro aggiuntivo.
Il package per funzionare, esattamente come l'SDK nativo, necessita dell'installazione dell'Android SDK in quanto l'applicazione che viene fuori con questo IDE altro non è che una classica applicazione Android che alla partenza inizializza e lancia un substrato applicativo scritto in C# che poi è la vera applicazione Unity. Dopo aver impostato gli SDK l'ultimo passaggio è quello di importare il Package fornito da Rift negli Asset di Unity e da lì cominciare lo sviluppo.

La scelta di utilizzare questo Framework e l'IDE di Unity ai fini del progetto di tesi è dovuto al fatto che il supporto della comunità di program-

matori che c'è dietro allo sviluppo con questa piattaforma è molto vasto. La documentazione con la descrizione di ogni singola funzione è facilmente reperibile e la società ha messo a disposizione numerosi video tutorial che aiutano a muovere i primi passi con questa tecnologia. Altro grande punto a favore è legato anche al fatto che l'applicazione che ne viene fuori è, come anche scritto sopra, a tutti gli effetti una applicazione Android e per questo motivo è possibile attraverso il substrato in C# poter richiamare la MainActivity.java e qualsiasi funzione scritta che implementa l'Android SDK. Per farlo è sufficiente inserire all'interno della cartella *Asset/Plugin/Android* il .jar contenente il plugin Java che si intende utilizzare.

```
//Getting the system language with the safe approach
void Start () {
  using (AndroidJavaClass cls =
    new AndroidJavaClass("java.util.Locale")) {

      using(AndroidJavaObject locale =
        cls.CallStatic<AndroidJavaObject>("getDefault")){

          Debug.Log("current lang =
            " + locale.Call<string>("getDisplayLanguage"));
        }
    }
}
```

Esempio di funzione che richiama codice java e variabili, l'attributo using serve a dire al GC [21] di deallocare la memoria non appena la funzione sarà terminata.

In questo modo in linea di massima è possibile richiamare, attraverso Unity e una libreria particolare chiamata UnityEngine.AndroidJNI, la libreria nativa VrLib attraverso la MainClass.java ed utilizzarla liberamente. Questa cosa invece non è possibile farla in senso contrario, ovvero utilizzando da subito lo sviluppo nativo cercando di richiamare le funzionalità di Unity e dell'Oculus package. Naturalmente il punto a sfavore rispetto alla prima

scelta di SDK è che, avendo un dispositivo mobile e una sub-applicazione posta su una già pesante struttura Java, è necessario avere una maggiore accuratezza riguardo l'uso delle risorse e delle performance in modo da non avere lag o rallentamenti significativi.

# Capitolo 3

# Il servizio proposto

Nei capitoli precedenti è stato descritto il quadro generale in cui questo progetto di tesi intende inserirsi per portare innovazione nel campo informatico legato ai tour virtuali. Una volta descritto il contesto, si sono mostrate le interfacce utente, i servizi e i tool attualmente esistenti che offrono un approccio a questa tecnologia, i dispositivi VR attualmente sul mercato e i framework disponibili per la realizzazione di applicazioni su di essi.
In questo capitolo invece viene descritto in cosa consiste il progetto di tesi e le motivazioni che hanno spinto alla realizzazione di questa piattaforma e dei suoi client di visualizzazione.

## 3.1 Problemi dello stato dell'arte e obiettivi della soluzione proposta

Come mostrato nello stato dell'arte, nonostante siano molti i servizi che permettono di lavorare con panorami a 360 gradi o con tour virtuali veri e propri, la totalità di questi, ad oggi, manca di alcune caratteristiche fondamentali che sono state considerate importanti e che hanno spinto ad iniziare un lavoro di questo tipo.
Cominciando con l'analisi delle piattaforme sul web o dei tool esistenti si è notato come nessuno di quelli visionati ha un metodo davvero facile ed intuitivo per la realizzazione di tour virtuali. In alcuni di questi semplicemente non è possibile caricare in autonomia i panorami necessari mentre

altri non permettono di personalizzare poi il risultato finale come magari desiderato dall'utente. Oppure se invece le piattaforme, o i tool, permettono tutto questo, poi comunque sono a pagamento con un costo che supera le centinaia di euro l'anno per il mantenimento della loro licenza. La volontà di questo lavoro è quindi quella di supportare l'utenza comune che non intende spendere soldi per la realizzazione e la visualizzazione dei tour virtuali, fornendo tuttavia un servizio valido, completo e superiore alla media attualmente in circolazione. Per risolvere questa prima problematica si è pensato ad una piattaforma online gratuita, semplice da utilizzare, in cui caricare i propri contenuti multimediali in completa autonomia senza vincoli e dando la possibilità all'utente di creare, salvare ed editare come meglio crede il proprio tour virtuale.
Per l'editing il sistema proporrà due possibilità di hotspot differenti: testuale o di transizione tra le scene che potranno essere posizionati a piacimento dell'utente all'interno di ogni panorama. La piattaforma farà da hosting a tutti i tour virtuali generati dando così la possibilità agli utenti di non dover comprare uno spazio dedicato per lasciar visualizzare il proprio lavoro ma basterà semplicemente fornire l'indirizzo web del tour virtuale. Questa non è una vera e propria innovazione in quanto una buona parte dei servizi, tra cui alcuni di quelli citati sopra, già hostano i lavori costruiti. E' giusto tuttavia sottolinearla in quanto non è uno standard rispetto alla maggior parte dei servizi che invece preferiscono rilasciare come unica alternativa un codice da inserire altrove. Per concludere, un'altra differenza con i competitors visionati è che il servizio proposto è interamente RESTful, ciò significa che fornisce API ad eventuali sviluppatori esterni, in maniera gratuita, che permettono a chiunque ne abbia voglia di creare un proprio client per la costruzione di tour virtuali sfruttando la logica sul server di questa piattaforma proposta. Questa cosa attualmente non è supportata e promossa da nessun altro servizio o tool in rete.

Anche nell'utilizzo dei dispositivi di Realtà Virtuale per la visualizzazione dei tour, di cui ho parlato in un paragrafo dello stato dell'arte (Sezione 2.1.2), ho riscontrato diverse mancanze funzionali. Nonostante alcune piattaforme come 360cities abbiano inserito all'interno dei loro servizi online la possibilità di visualizzare da web le scene panoramiche con il Google Card-

board o altri VR set, tra quelli visionati, nessuno ha mostrato la funzionalità di poter esplorare l'intero tour virtuale. Questa carenza ha spinto a cercare di implementare questa funzione all'interno della applicazione per Gear VR così da offrire un'esperienza più completa per l'utente che potrà in questo modo esplorare un tour nella sua interezza senza dover cambiare scena ed inserire nuovamente il suo visore per la realtà virtuale.
Invece per quanto riguarda le applicazioni mobile scritte appositamente per dispositivi VR che ho visionato e da cui ho preso spunto, è venuto fuori che queste non interagiscono con un server online per il caricamento dinamico dei contenuti, ma necessitano tutte di avere i file multimediali già all'interno del dispositivo. Questo riduce notevolmente la scalabilità del numero dei tour virtuali all'interno dell'applicazione che deve essere aggiornata tramite gli store ufficiali per aumentare i propri contenuti, oppure che costringe l'utente a dover inserire manualmente i propri file all'interno del telefono per poi visualizzarli.
La soluzione prevista per questo lavoro di tesi invece punta a scaricare i dati presenti sulla piattaforma attraverso il servizio RESTful così da avere sempre contenuti aggiornati senza la necessità di modificare gli elementi interni dell'applicazione. Anche in questo caso, non sono state riscontrate piattaforme online che hanno sviluppato una propria applicazione nativa per un dispositivo di realtà virtuale che si sappia interfacciare dinamicamente con i loro contenuti in rete.

## 3.2 Requisiti

In questa sezione verranno analizzati i requisiti funzionali e no del sistema, nel quale le uniche persone o enti coinvolti sono gli utenti che sfruttano il servizio per creare tour virtuali o visualizzare altri tour presenti sulla piattaforma.

### 3.2.1 Requisiti funzionali e attori

Analizzando i requisiti sono state individuate tre tipologie di attori, ovvero di entità all'interno del sistema, a cui sono associate diverse funzionalita:

Utente non registrato, client web:

- Registrazione tramite compilazione di apposita form
- Registrazione attraverso un account Facebook e Google Plus esistente
- Ricerca per utenti, visualizzazione relativi profili
- Ricerca per coordinate geografiche
- Visualizzazione di tour virtuali esistenti
- Pubblicazione su Facebook di un tour virtuale

Utente registrato, client web:

- Login tramite credenziali
- Login tramite account Facebook o Google
- Ricerca utenti, visualizzazione dei relativi profili e following
- Creazione di nuovi tour e upload foto 360
- Modifica tour virtuali creati
- tutte le altre funzionalità che può fare un Utente non registrato

Utente non registrato, client Gear:

- Ricerca dei tour virtuali nelle vicinanze
- Visualizzazione di tour virtuali

Sostanzialmente, per avere accesso a qualunque funzionalità che non sia di sola lettura di informazioni, è necessario essere registrati. Un utente non registrato potrà accedere alla piattaforma web la prima volta utilizzando un account Facebook o Google: in questo caso verrà registrato automaticamente un nuovo utente utilizzando la mail usata per la registrazione a tale servizio. Gli account identificati dalla stessa mail saranno unificati: se ad esempio un utente si registra e in un secondo momento effettua il login tramite un account Facebook associato alla stessa mail, i due account verranno

unificati automaticamente. Ogni utente, una volta effettuato il login, verrà rimandato alla propria home page, dalla quale potrà navigare attraverso le varie pagine usufruendo delle funzionalità a sua disposizione, descritte nei punti appena sopra.

### 3.2.2 Sicurezza

Come già accennato in precedenza, esistono tre possibili modi per effettuare il login sulla piataforma web: account locale, account Facebook e account Google. In caso di login tramite account locale, il sistema richiederà la mail e password fornite dall'utente al momento della propria registrazione; verificherà quindi l'esistenza nel database di un utente che corrisponda alla mail inserita, e una volta trovato verificherà la corrispondenza della password. Per ragioni di sicurezza, tutte le password sono codificate nel database tramite un hash univoco. In caso di login effettuato correttamente, verrà generato ed inviato al client un token. In caso di login tramite Facebook o Google, invece, il sistema attenderà un token per il servizio corrispondente, ne verificherà la validità effettuando un controllo con il servizio, e in caso positivo salverà il token associandolo all'utente.

Ogni token è univoco ed è associato all'utente per l'intera durata della sessione corrente, ovvero fino a quando non viene ricevuta una richiesta di logout da parte dell'utente. Esso verrà richiesto dal sistema per effettuare tutte le funzionalità che solo l'utente registrato può compiere.

## 3.3 Descrizione

### 3.3.1 Entità e Relazioni

In fase di progettazione sono venute fuori le seguenti entità fondamentali per la piattaforma:

- Utenti
- Media (Info metadata)
- Hotspot
- Scene
- Tour virtuali

Ogni entità è in relazione con le altre come segue:

- Ad un Utente sono correlati da 0 a n Media caricati
- Ad un Utente sono correlati da 0 a n Tour Virtuali costruiti
- Ad una Scena sono correlati da 0 a n Hotspot
- Ad una Scena è correlato un solo Media
- Ad un Tour Virtuale sono correlate da 1 a n Scene

Quanto scritto è rappresentato nel seguente modello ER (Entità Relazione):

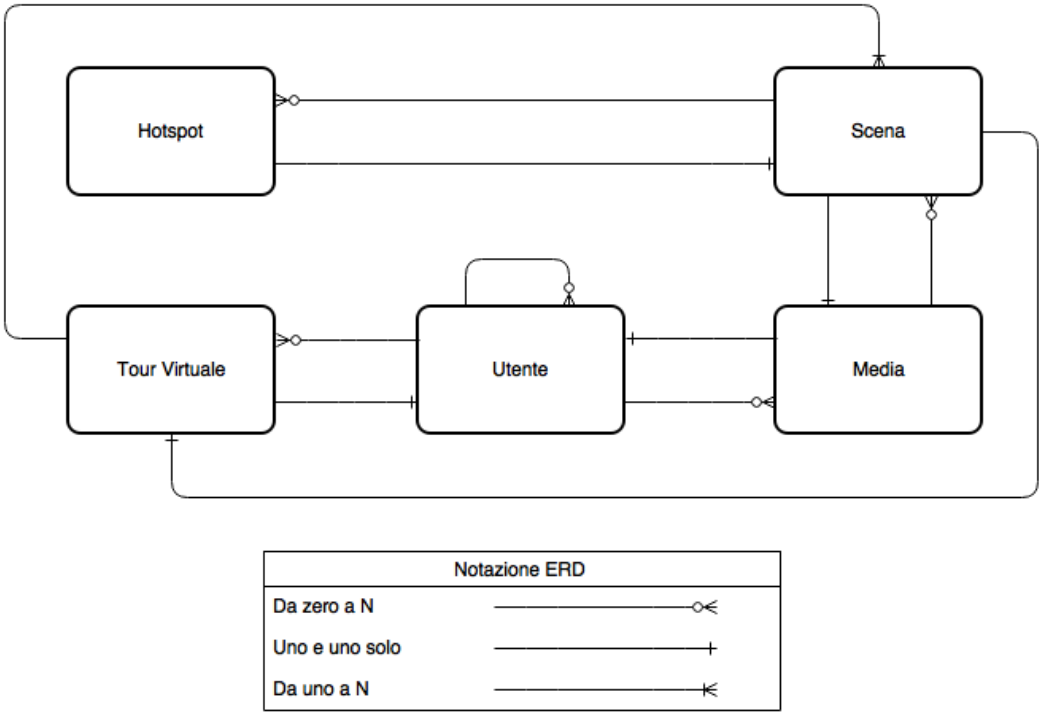


*Figura 3.1: Modello ER del sistema*

# Capitolo 4

# Architettura del sistema REST

## 4.1 Introduzione

La prima parte di questo progetto di tesi, come anticipato nei capitoli precedenti, si basa su un'applicazione web che permette di creare un tour virtuale con una interfaccia semplice ed intuitiva. Si presenta anche come un raccoglitore di questi tour in quanto fin da subito all'utente viene presentata a schermo una mappa con dei *segnaposto* che mostrano i tour postati da altre persone più vicini alla sua posizione. In questo modo un qualsiasi utente potrà visualizzarli e apprezzarli. Grazie ad una pagina personalizzata, è possibile anche gestire il proprio profilo e seguire gli utenti che postano i tour virtuali più interessanti.
Riguardo il costruttore di ambienti 360, la sua intuitività sta nel fatto che in automatico vengono proposti i propri file multimediali caricati da un'apposita pagina, e con una interfaccia semplice si ha la possibilità di creare un nuovo progetto tour virtuale e aggiungere hotspot di vario tipo in posizioni precise arricchendo le scene con testi e transizioni.
Questo servizio tuttavia non è solo un'applicazione per la creazione e visualizzazione social di tour virtuali, ma è anche un servizio RESTful [11] che espone le funzioni che lei stessa utilizza in modo da permettere a qualsiasi sviluppatore di costruire un client diverso basato su questa piattaforma e sulla sua logica.
In questo capitolo verrà spiegata la prima parte della tesi, ossia come è stata creata quest'applicazione web, quali scelte architetturali sono state adotta-

te ed infine vi sarà una descrizione sulla struttura del progetto ed il suo funzionamento.

## 4.2 Librerie utilizzate

Data la volontà di progettare un'applicazione disponibile in ogni momento, per ogni piattaforma e con la forte propensione alla condivisione dei contenuti, la decisione di sviluppare un applicativo web è sembrata la scelta più sensata.
Negli ultimi anni sono nati nuovi standard per il web come HTML5 e CSS3 che hanno colmato numerose problematiche legate alla scarsa qualità delle web-application, questi nuovi linguaggi infatti permettono funzionalità avanzate che erano solo prerogativa delle applicazioni native. Anche il linguaggio Javascript, supportato da ogni browser moderno, permette delle interazioni con l'utente che anni fa erano impensabili. Questo linguaggio è proprio alla base di numerosi framework nati con lo scopo di creare applicazioni avanzate. Alcuni di questi verranno illustrati in questo capitolo poiché sono stati impiegati per lo sviluppo della parte server e client side.

### 4.2.1 Node.js

Per lo sviluppo del prototipo è stato utilizzato *Node.js* [13], una piattaforma basata sul motore Javascript di Google Chrome, che permette di scrivere il codice sorgente del server. Il modello di networking su cui si basa Node.js non è quello dei processi concorrenti, ma è I/O event-driven. Ciò vuol dire che Node richiede al sistema operativo di ricevere notifiche al verificarsi di determinati eventi, e rimane quindi in sleep fino alla notifica stessa: solo in tale momento torna attivo per eseguire le istruzioni previste nella funzione di *callback*, così chiamata perchè da eseguire una volta ricevuta la notifica che il risultato dell'elaborazione del sistema operativo è disponibile. Tale modello di networking, è ritenuto più efficiente nelle situazioni critiche in cui si verifica un elevato traffico di rete, ma è anche molto utile se si vuole impostare un servizio RESTful basato sulla chiamata di API da parte di client esterni.

- Una caratteristica molto interessante di Node.js è l'orientamento verso lo sviluppo di codice asincrono. Ciò significa che l'esecuzione del codice non avviene in modo sequenziale, eseguendo un'istruzione dopo l'altra, come nella vecchia programmazione strutturata. In alcuni casi l'esecuzione del codice può saltare avanti ed indietro, soprattutto quando una delle istruzioni rimane in attesa di un risultato che impiega un certo tempo per essere disponibile. Ad esempio, quando un'istruzione richiede un valore ad un server remoto, come nel caso di una chiamata Ajax, l'esecuzione del codice non si ferma in attesa del risultato ma prosegue, 'saltando' l'istruzione che effettua la chiamata. Tale istruzione rimane in uno stato di intermedio, perchè è stata eseguita ma non completata, nel frattempo vengono eseguite le istruzioni successive. Quando la chiamata remota viene completata il controllo dell'esecuzione torna all'istruzione che era stata sospesa. Questo modo di lavorare permette di ottimizzare notevolmente le performance, perchè quando si lavora sul Web i tempi di attesa tra una richiesta HTTP e l'altra sono tempi molto lunghi rispetto ai tempi di lavoro del processore, per cui sarebbe uno spreco restare in attesa di ogni possibile risposta.

- Altro vantaggio di Node.js è la possibilità di realizzare applicazioni professionali scalabili. Ciò è dovuto all'architettura modulare che sta alla base di Node.js. Dopo averlo installato è possibile facilmente aggiungere altri moduli, scegliendo sia tra quelli ufficiali che vengono rilasciati con l'istallazione minima, sia quelli sviluppati da altri utenti. La cosa più importante, per quanto riguarda l'architettura modulare, è che ogni applicazione può essere scritta in questo modo, anzi, il paradigma di lavoro di Node.js spinge nella direzione di applicazioni prodotte con un approccio modulare.
  Lavorando con Node.js è molto facile organizzare il lavoro in librerie, importare i nostri moduli e poi riciclarli tra un progetto e l'altro. I moduli disponibili sin dalla prima installazione, cioè quelli nativi, si trovano all'interno della directory *npm* (che sta per Node.js Package Management), mentre i propri moduli personali possono essere collocati a piacimento. Dopo aver suddiviso un'applicazione in un certo nume-

ro di moduli diventa molto facile combinarli tra loro, un po' come se fossero oggetti nell'ambito del paradigma di programmazione ad oggetti (OOB). Un'applicazione Node potrebbe essere realizzata facendo lavorare in concerto un gran numero di moduli, eventualmente distribuiti su macchine diverse, facendoli comunicare tra loro. E' proprio per questo che è stato scelto il nome Node: ogni componente software si comporta come un piccolo nodo all'interno di una rete, la quale può essere facilmente scalata e riproporzionata in base alle esigenze.

- E' una tecnologia relativamente giovane e in continuo sviluppo, per questo motivo segue tutte le migliori prassi del momento. Questo fattore è di fondamentale importanza perchè questo prototipo fa largo uso di funzionalità moderne, in primis HTML5 e CSS3. Ogni giorno sempre più sviluppatori scelgono Node.js, e questo aumenta il numero di librerie, plugin, snippet e risorse che è possibile trovare sul Web. Sono molti infatti i framework di sviluppo nati e basati proprio su questa piattaforma, come ad esempio *ExpressJS*, che è anche quello che è stato utilizzato per questo progetto.

### 4.2.2 Express.js

Express.js [4] è un framework web che è basato su Node.js, sui moduli HTTP e serve a connettere i componenti che sono anche chiamati middleware. La sua filosofia si basa sul concetto di *configuration over convention*, ovvero gli sviluppatori sono liberi di prendere le librerie di cui hanno bisogno per un particolare progetto senza dove necessariamente ogni volta 'reinventare la ruota' ma riutilizzando moduli scritti in precedenza da altre persone per implementare task come:

- parsing di richieste HTTP
- parsing di cookies
- gestione delle sessioni
- routing delle richieste client

Express aiuta in questo aggregamento e fornisce una struttura MVC per la propria applicazione web che vuole essere RESTful. La separazione del Modello dati, della View dei risultati e dalla logica del Controller viene gestita grazie al *routing*: ovvero il framework capisce come l'applicazione sul server deve rispondere al client, che ha chiamato un particolare *endpoint* attraverso la View, e che azione deve compiere. La definizione del route deve avere la seguente struttura:

```
app.METHOD(PATH, HANDLER)
```

dove 'app' è una istanza di Express, METHOD è una richiesta HTTP e HANDLER corrisponde alla funzione da eseguire lato server. Il suo utilizzo è risultato fondamentale in quanto grazie ad esso tutta la gestione delle diverse API e delle inter-connessioni con il database noSQL di MongoDB è diventato automatico e facile da mantenere.

### 4.2.3 MongoDB

Per la gestione dei dati persistenti è stato utilizzato MongoDB [12], un DBMS non relazionale, orientato ai documenti. Classificato come un database di tipo NoSQL, MongoDB si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico (MongoDB chiama il formato BSON), rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce. La scelta di questa tipologia di database è avvenuta per i seguenti motivi:

- Necessaria flessibilità per future espansioni e future tipologie di dati. Il prototipo per i tour virtuali potrebbe avere in futuro espansioni a tipologie di dato che al momento non sono previste. Grazie alla mancanza di uno schema fisso è sempre possibile aggiungere informazioni differenti senza dover modificare gli elementi già inseriti nel database. Questo viene in aiuto anche per i dati che non sono totalmente completi, in quanto potrebbe capitare che alcuni tour virtuali siano più definiti di altri e senza uno schema fisso a tabelle non si vengono a creare elementi quasi vuoti con relativa perdita di spazio.

- La struttura delle entità a JSON rende perfetto il loro utilizzo per lo sviluppo di API.

- Qualunque campo in MongoDB può essere indicizzato. Sono disponibili anche indici secondari, indici unici, indici sparsi, indici geospaziali e indici full text. Funzione davvero utile in quanto ogni media presente nel tour virtuale è georeferenziato.

- MongoDB può essere usato anche come un file system, traendo vantaggio dalle caratteristiche di replicazione e di bilanciamento su più server per memorizzare file, anche di grandi dimensioni. Questa funzione, chiamata *GridFS*, è inclusa nei drivers di MongoDB e disponibile facilmente per tantissimi linguaggi di sviluppo, il database deve solo esporre delle funzioni per la manipolazione dei file. GridFS è stato molto usato all'interno del prototipo per il salvataggio dei file multimediali che potevano raggiungere anche grandi dimensioni in termini di MByte. Invece di memorizzare il file in un singolo documento, la GridFS divide il file in tante parti più piccole, chiamate chunks, e memorizza ognuno di questi chunck in un documento separato. Ai file possono essere associati dei metadati, su cui è possibile anche creare degli indici full-text.

### 4.2.4 Angular.js

E' un framework web open-source creato da Google e da una comunità di sviluppatori individuali. Ha avuto un ruolo principale nello sviluppo della parte front-end del prototipo in quanto ha una struttura particolarmente idonea al MVC e perchè possiede importanti caratteristiche che adesso verranno descritte.

- Il Data-binding è sicuramente la caratteristica più utile di AngularJS [2] che permette di risparmiare una notevole quantità di codice da scrivere. Una tipica applicazione web potrebbe contenere circa l'80% di codice base dedicato all'attraversamento, manipolamento o all'ascolto del DOM, grazie al framework questo codice scopare. Il data binding è il meccanismo di sincronizzazione automatica dei dati tra il

modello e la view. La maggior parte dei sistemi di template supporta il data binding in una sola direzione, tipicamente dal modello dei dati verso la view. Questo vuol dire che i dati del modello vengono combinati con il template HTML per generare la view visibile all'utente. Se però il modello viene modificato, le modifiche non si riflettono automaticamente sulla view. Analogamente, se l'utente modifica la view, queste modifiche non vengono automaticamente riportate sul modello dei dati. Per sincronizzare view e modello occorre in genere scrivere del codice che lo faccia. Invece questa sincronizzazione avviene senza la necessità di scrivere del codice particolare. E' sufficiente associare il modello allo scope (un oggetto standard del framework condiviso tra controller e view) all'interno del controller ed utilizzare la direttiva ng-model nella view.

- AngularJS incorpora i principi base dietro il pattern design MVC.

  *Il model* è semplicemente il dato nell'applicazione. Non c'è bisogno di ereditare dalle classi del framework, includerlo negli oggetti proxy o usare dei metodi speciali get/set per l'accesso.

  *Il viewmodel* è un oggetto che fornisce specifici dati e metodi per il mantenimento di specifici view. Il viewmodel è un oggetto $scope che vive con l'applicazione AngularJS. $scope è solo un semplice oggetto JavaScript con delle piccole API progettate per rilevare e trasmettere i cambiamenti al suo stato.

  *Il controller* è il responsabile dell'impostare lo stato iniziale e aumentare lo $scope con metodi che controllano il comportamento. Vale la pena notare che il controller non salva lo stato e non interagisce con i servizi remoti.

  *La view* è del semplice codice HTML che esiste dopo che AngularJS ha analizzato e compilato per includere markup e bind renderizzati.

  Questa divisione generale crea una base solida per architettare una applicazione. Lo $scope ha un riferimento ai dati, il controller definisce il comportamento, e il view gestisce il layout e consegna l'interazione al controller per rispondere di conseguenza.

### 4.2.5 Bootstrap

Rilasciato come open source da Twitter nel 2011, Bootstrap è diventato negli anni un punto di riferimento come front-end framework nelle applicazioni web [3]. Esso consiste in una collezione di strumenti utili per creare l'interfaccia grafica web. E' stato usato questo framework come punto di partenza per creare l'impaginazione dell' applicazione.

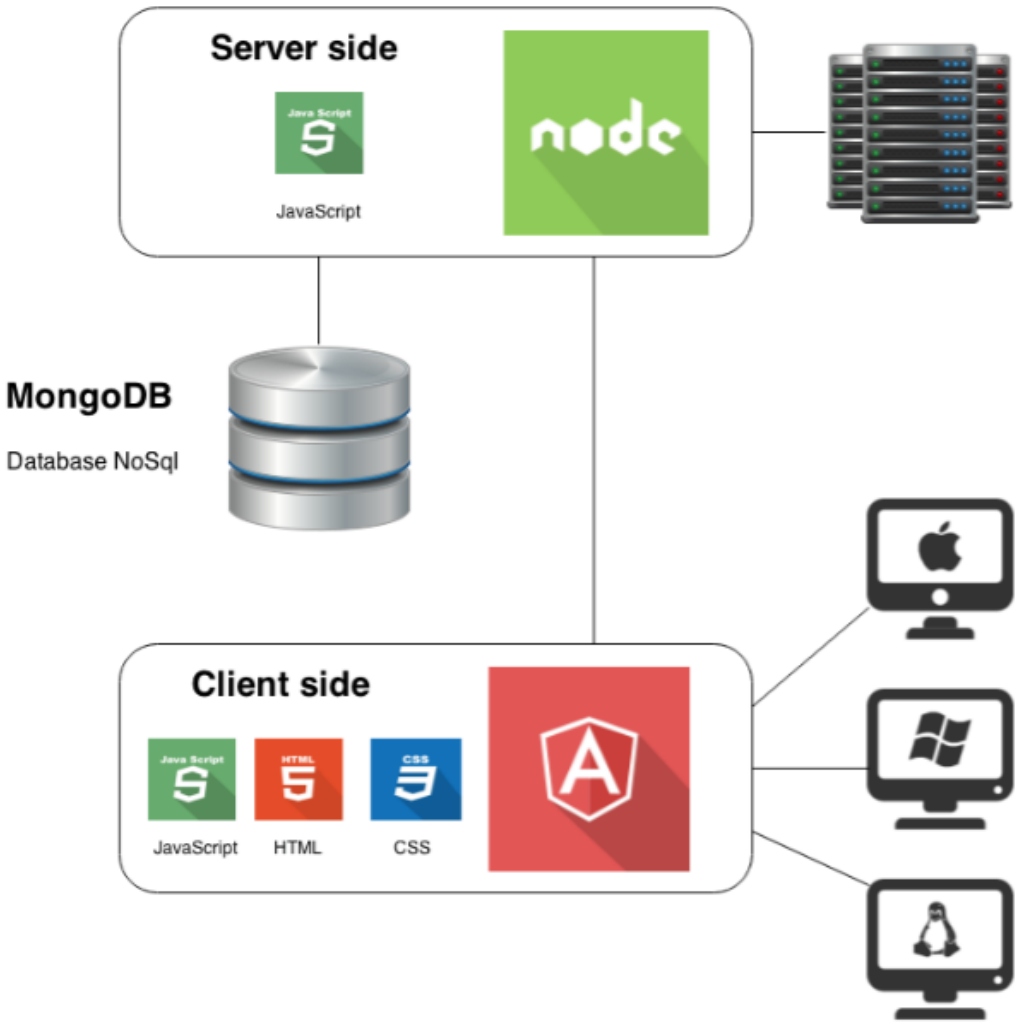


*Figura 4.1: Architettua del sistema, come i framework si interfacciano tra di loro formando il MEAN Stack*

## 4.3 Architettura

Ora che sono state fornite le conoscenze necessarie per poter comprendere il prototipo, verranno utilizzati i concetti appena esposti per spiegare il funzionamento della applicazione web: ovvero quella che permette all'utente di generare il proprio tour virtuale, di hostarlo sulla piattaforma e visualizzarlo.

### 4.3.1 Struttura del progetto

In questa sezione viene illustrato com'è stato strutturato il prototipo, in particolare viene descritta la gerarchia delle cartelle e le loro funzionalità ad alto livello. Più avanti invece verrà messo in evidenza cosa contengono le funzionalità dei singoli file.

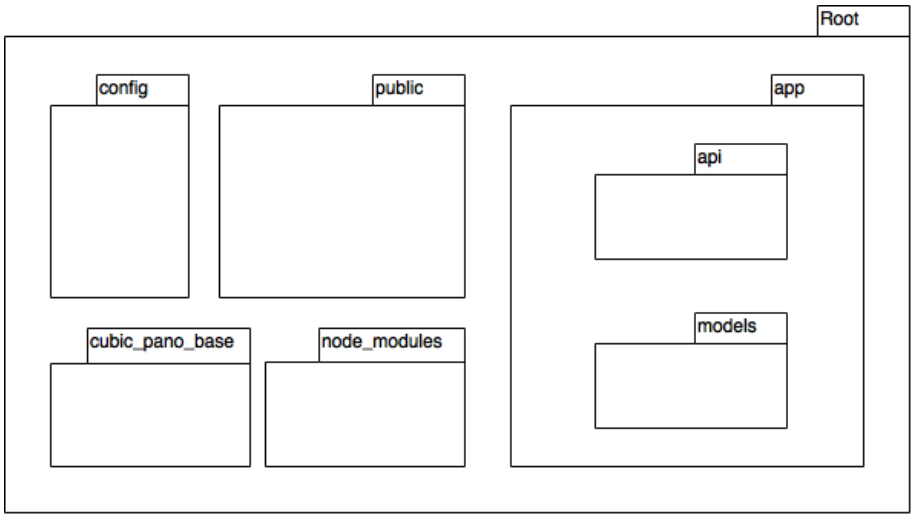


*Figura 4.2: Struttura cartelle prototipo*

Con questa modalità top-down sarà più facile capire la struttura generale e come i singoli componenti interagiscono tra di loro. Analizzando il codice sorgente si evince la seguente gerarchia:

- *app*: al suo interno vi è l'applicazione server vera e propria. Data la sua importanza verrà spiegata in modo approfondito anche nelle sezioni successive. Qui si trovano tutti i codici sorgenti che permettono alla web application e a tutte le API di funzionare.

- *config*: qui vi sono tutte le configurazioni del server e alcune funzionalità dette 'Utils', comode da richiamare in caso di necessità in tutto il codice.

- *node_modules*: al suo interno vi sono tutte le librerie necessarie per il corretto funzionamento degli applicativi che hanno supportato lo sviluppo e la distribuzione dell'applicazione.

- *cubic_pano_base*: cartella temporanea in cui vengono salvate momentaneamente le foto cubiche, dopo il processo di trasformazione da foto equirettangolari, prima di essere caricate sul database.

- *public*: al suo interno vi è l'applicazione client vera e propria. Data la sua importanza anche questa verrà spiegata in modo approfondito nelle sezioni successive. Qui si trovano tutti i codici sorgenti che permettono alla web application di mostrare i risultati e di costruire gli input da mandare al server.

Per concludere, oltre la presenza di queste cartelle vi è anche un file chiamato *server.js* che ha la funzione di inizializzare tutti i moduli di Node, la porta del server e tutte le connessioni con il database. Per farlo si avvale delle funzioni di Express.js di cui si è parlato nel paragrafo precedente (4.2.2).

### 4.3.2 Cartella app

Come introdotto in precedenza questa cartella contiene il cuore dell'applicativo server. E' qui dentro che vi sono tutte le implementazioni necessarie al funzionamento delle API e all'interfacciamento con il database. Proprio in base a questa logica la cartella *app* si divide in:

- *api*: al suo interno vi sono tutti i file in Javascript che compongono l'intero pacchetto di API. Ogni file gestisce una differente macroarea: *authorization*, *media*, *profile*, *virtualtour*.

- *models*: qui vi sono tutti i file Javascript che definiscono la struttura dei dati persistenti presenti nel prototipo e che permettono alle API

di interconnettersi con il database Mongo (4.2.3). Si parlerà meglio di questa cartella nei paragrafi successivi.

**Routes**

Il routes è un file nel progetto del prototipo che svolge un ruolo di primaria importanza. Il suo compito è quello di tradurre i differenti url di richieste indirizzando la chiamata alla corretta funzione sul server. In esso è possibile trovare l'intero schema di url disponibili, quindi API, che il prototipo è in grado di gestire. Il routes è il primo componente che entra in gioco quando arriva una richiesta sul server spinto da Node. Il file in questione si trova nella cartella *app/routes.js*.
Gli url in grado di gestire sono molteplici data la dinamicità dell'applicazione, molti di essi sono anche gerarchici in modo tale da soddisfare il paradigma CRUD [23] per le API, che impone l'utilizzo di *endpoint* comuni per lo stesso tipo di dato.
Per dare un'idea dello schema degli url utilizzati ecco qualche esempio. Attuando una richiesta di GET, PUT o DELETE sullo stesso endpoint */api/-media/:id* si dà il via a tre differenti funzioni sul server che rispettivamente forniscono info, aggiornano, o cancellano un Media con determinato :id dal database. Oppure vi sono altri endpoint singoli per funzionalità specifiche come */api/nearMedia*, questo nello specifico ritorna tutti i Media ad una certa distanza da una coordinata passata con la richiesta di GET.

### 4.3.3 Models

Per model si intendono le classi che possiedono uno stato persistente e che descrivono la struttura che poi le singole istanze di queste classi avranno sul database. I modelli creati nel prototipo non sono molti ma un buon design del model è stato di fondamentale importanza sia per semplificare la gestione dei dati sia per permettere all'applicazione di poter essere scalabile in futuro senza stravolgimenti di struttura. Tutti i modelli sono situati nella cartella *app/models*.
In questa sezione verranno passati in rassegna tutti i modelli presenti nel-

l'applicazione web spiegando in modo dettagliato tutti gli attributi che li compongono.

**Media**

Questo può essere sicuramente considerato il modello più importante in quanto è alla base della costruzione di una scena per un tour virtuale e al tempo stesso permette di collezionare tutte le informazioni dei file caricati su Mongo attraverso la GridFS [6]. Grazie a questo modello infatti è stato possibile tenere traccia non solo delle informazioni e descrizioni generali di un panorama, ma allo stesso tempo è stato utilizzato per salvare al suo interno tutti i riferimenti ai vari formati del panorama stesso in quanto al caricamento sulla piattaforma di un'immagine multimediale, questa viene tagliata in vari formati differenti per motivi che verranno spiegati a breve.

Per chiarezza è giusto ricordare che la GridFS è una astrazione di un File-System che Mongo utilizza per il salvataggio di file. Tutti i file multimediali del progetto sono stati inseriti su questa GridFS. Ogni singola istanza di questa Grid, chiamata *fs.files*, ha al suo interno una struttura standard che viene riportata di seguito:

```
{
  "_id" : <ObjectId>,
  "length" : <num>,
  "chunkSize" : <num>,
  "uploadDate" : <timestamp>,
  "md5" : <hash>,
  "filename" : <string>,
  "contentType" : <string>,
  "aliases" : <string array>,
  "metadata" : <dataObject>,
}
```

I campi su cui porre l'attenzione sono quelli dell'*_id* e del *metadata* poiché sono stati utilizzati in relazione con il modello Media per mantenere i riferimenti ai file sulla GridFS. Di seguito la specifica di ogni attributo del modello Media:

- *_id*: questo è l'identificativo con cui il Media viene salvato. E' posto in relazione con il file vero e proprio sulla GridFS in quanto questo _id è lo stesso che viene inserito all'interno del campo *metadata* del file. In questo modo ogni fs.file sulla Grid ha un suo modello Media collegato sul quale può salvare molte informazioni aggiuntive che vengono descritte di seguito.

- *namefile*: label identificativa che può essere data al media caricato.

- *country, city, zone*: tre attributi differenti che specificano rispettivamente la nazione, la città e la zona nel quale viene collocato spazialmente il media che si sta caricando.

- *date*: orario e data in cui viene collocato temporalmente il media che si sta caricando. Interessante per sviluppi futuri legati ad una ricerca basata sul periodo storico.

- *authorId*: identificativo dell'utente che sta caricando il media. Come detto in precedenza ogni Media è legato ad un solo utente del sistema.

- *loc*: [lng, lat] coordinate geografiche del media caricato. Di fondamentale importanza per dare al tour virtuale la possibilità di essere collocato spazialmente. Ogni Media è indicizzato anche in base a questo attributo, il che significa che è possibile tornare tutti i Media ad una distanza configurabile da una certa coordinata fornita.

- *mediaType*: [type, ext] formato del media in questione diviso per tipologia di file ed estensione.

- *sizeFileId*: questo è un campo importante in quanto al suo interno possiede gli _id che si riferiscono ai vari formati della stessa immagine. Questo attributo infatti non è altro che un oggetto con dei valori interni: *small, medium, big, full* che puntano agli indirizzi dei rispettivi file salvati sulla GridFS. Come annunciato poco prima infatti al caricamento di una nuova immagine sulla piattaforma, questa viene tagliata

in vari formati per questioni di performance, e tutti questi file differenti vengono salvati sul FileSystem. Grazie a questo attributo ognuno di questi formati è connesso, in maniera biunivoca, allo stesso modello Media dato che si tratta della stessa immagine ma ad un formato differente.

- *gearCube*: attributo che ha la stessa struttura e la stessa funzione di quello precedente, con la differenza che questo salva le 6 immagini cubiche derivanti da una trasformazione equirettangolare-cubica del panorama di base. Questo formato, che verrà spiegato in seguito è di fondamentale importanza per l'integrazione dei panorami con il Samsung Gear VR.

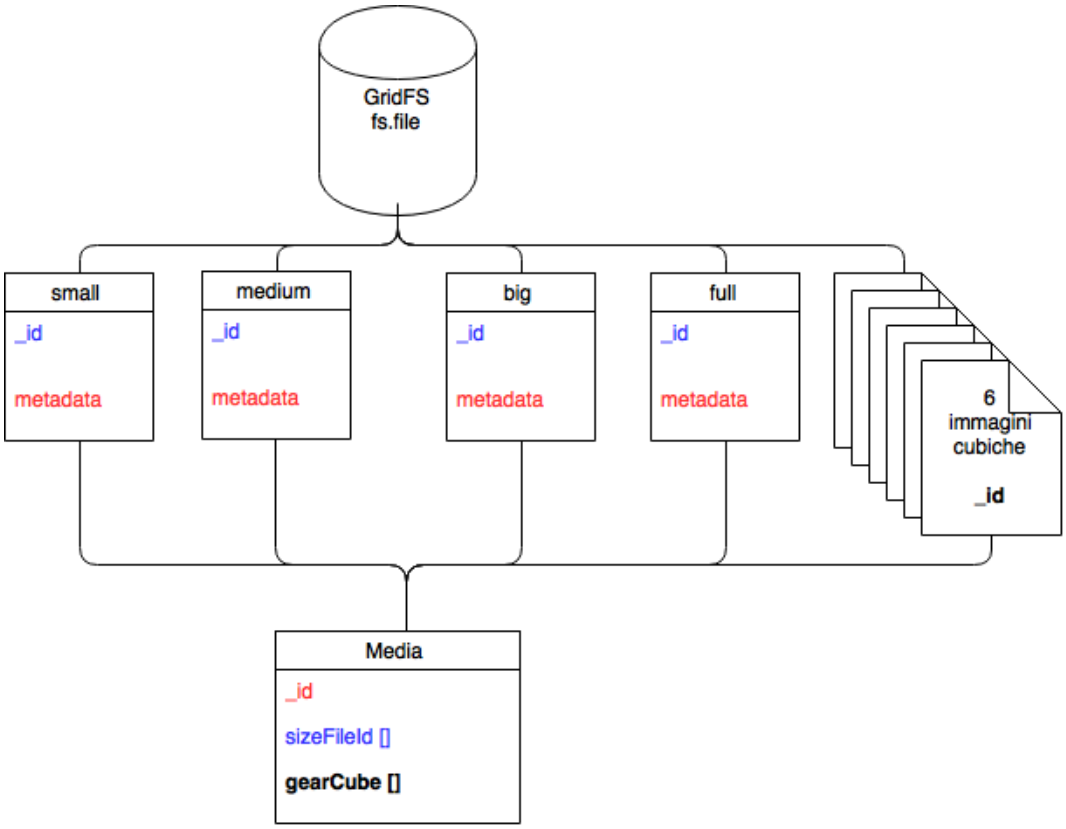


*Figura 4.3: Relazione tra i file sulla GridFS e il modello Media, i colori evidenziano le interconnessioni spiegate poco sopra*

**Scene**

Questo modello può essere considerato come il passo successivo al modello Media in quanto, come sfondo dell'ambiente circostante a 360 gradi, fa riferimento proprio ad un Media precedentemente caricato. Dall'interfaccia grafica per la costruzione di un tour virtuale, è possibile scegliere infatti un Media e a questo aggiungere una label e vari hotspot di riferimento per la creazione della scena. Una moltitudine di istanze di questo modello portano alla creazione di un Tour Virtuale completo. Di seguito i suoi attributi:

- *sceneId*: identificativo univoco di riferimento della scena.
- *title*: label che può essere data alla scena come etichetta di riconoscimento.
- *hotspots*: lista di oggetti di tipo Hotspot appartenenti e posizionati sulla scena.
- *media*: riferimento all'oggetto Media da cui vengono prese l'immagine panoramica e tutte le altre informazioni che il modello porta con se.

**Hotspot**

Modello alla base di una Scena, permette di effettuare le transizioni tra di esse all'interno di un tour virtuale e di mostrare informazioni testuali relative ad un punto preciso dell'ambiente circostante. Di seguito i suoi attributi:

- *pitch*: [-90, +90] angolo in gradi rispetto all'asse Y in cui l'oggetto è posizionato.
- *yaw*: [-180, +180] angolo in gradi rispetto all'asse X in cui l'oggetto è posizionato.
- *type*: [scene, info] tipologia di hotspot, rispettivamente di transizione scena o di informazione testuale.
- *text*: se l'hotspot è di tipo *info*, questo campo è valorizzato con un testo descrittivo utile a definire un punto dell'ambiente circostante.

- *sceneId*: se l'hotspot è di tipo *scene*, questo campo è valorizzato con l'indirizzo di riferimento della scena in cui ci si deve spostare.
- *URL*: se l'hotspot è di tipo *info*, questo campo è valorizzato con un indirizzo URL qualsiasi visualizzabile via web.

**VirtualTour**

Modello risultato della piattaforma, questo racchiude le informazioni di tutti gli altri modelli in quanto è formato proprio da essi. Essendo il prodotto finale è quello che l'utenza si trova davanti nella esplorazione. Come anche tutti gli altri modelli, questo è unico per qualsiasi client e non ha versioni diverse in base al tipo di visualizzazione richiesta, web o immersiva che sia.

- *firstScene*: identificativo di riferimento della prima Scena con cui il tour virtuale deve cominciare.
- *loc*: [lng, lat] coordinate geografiche del tour che vengono prese automaticamente da quelle presenti nel Media della *firstScene*. Di fondamentale importanza per dare al Tour Virtuale la possibilità di essere collocato spazialmente. Proprio come i Media, anche questo modello è indicizzato in base a questo attributo, il che significa che è possibile tornare tutti i tour virtuali ad una distanza configurabile da una certa coordinata fornita.
- *scene*: lista di oggetti di tipo Scene che fanno parte del tour virtuale, tra questi vi è anche la firstScene.
- *like*: lista di riferimenti ad Utenti che hanno apprezzato il tour virtuale.

**User**

Modello conclusivo e alla base della piattaforma, in quanto necessario se si è intenzionati ad utilizzare le funzionalità principali del prototipo. Al suo interno possiede varie informazioni che vengono viste di seguito:

- *local.email*, *local.password*, *local.name*, *local.age*, *local.city*, *local.picture*: informazioni di vario genere utili al riconoscimento e alla descrizione

della persona. Possono essere riempite attraverso una form sulla piattaforma oppure, se si usa un social network per accedere, questi campi vengono automaticamente riempiti dalle informazioni presenti sul social di riferimento. La password in quel caso non viene valorizzata.

- *local.token*: campo valorizzato con un UUID unico di riferimento generato dalla piattaforma e che rimane attivo fino ad un eventuale logout dell'utente. Deve essere fornito con le richieste web ogni qual volta si voglia impiegare una funzionalità riservata ai soli utenti registrati.
- *local.virtualTour*: lista di riferimenti ad oggetti di tipo VirtualTour appartenenti e creati dall'utente.
- *local.followers/local.following*: lista di riferimenti ad Utenti che che si vuole seguire perchè reputati interessanti, o utenti da cui si è seguiti a sua volta.
- *local.verified*: boolean che indica se l'utente ha verificato il suo account attraverso la sua email. Questo campo non viene valorizzato se si utilizza un Social Network per accedere alla piattaforma.
- *facebook.id/facebook.token*: campi che vengono valorizzati nel caso in cui si effettua l'accesso attraverso Facebook.
- *google.id/google.token*: campi che vengono valorizzati nel caso in cui si effettua l'accesso attraverso Google Plus.

### 4.3.4 API

Con API si indica ogni insieme di procedure disponibili allo sviluppatore, di solito raggruppate a formare un set di strumenti specifici per il raggiungimento di un determinato compito all'interno di un certo programma. Le API create nel prototipo sono alla base di tutte le funzionalità presenti sulla piattaforma, in quanto senza di esse non sarebbe possibile interfacciarsi con il database per la gestione dei dati persistenti. Tutte le API sono situate nella cartella *app/api*.
In quanto sistema RESTful sono state identificate delle risorse utili per la creazione degli *endpoints* (o macroaree), e secondo questa logica le funzioni sono state divise in file differenti. In questa sezione verranno passati in rassegna proprio questi file Javascript su cui si darà una breve descrizione delle caratteristiche principali.

#### Authorization

Gestisce tutte le funzionalità legate alle autorizzazioni degli Utenti, ovvero quelle che riguardano le fasi di Login, Logout, SignUp sulla piattaforma. Inoltre anche le registrazioni attraverso i social di Facebook e Google Plus passano da questo file che si interconnette con questi portali per confermare la validazione o per scaricare i dati sull'utente.

#### Profile

Gestisce tutte le funzionalità legate ai dati degli utenti e alle relazioni tra di essi. Tutto quello che riguarda la modifica dei dati, il follow/unfollow di utenti, la ricerca per nome, o le informazioni riguardanti i propri Media o Tour Virtuali si trovano in questo file.
Gli endpoint di riferimento sono */api/profile* per gli altri utenti e */api/me* per le funzinalità personali.

**Media**

Tutto quello che riguarda la gestione dei file multimediali in piattaforma avviene sotto il controllo delle funzioni contenute nel file Media. Su questo endpoint vi sono tutte le azioni CRUD disponibili ed altre funzionalità di ricerca per coordiante. Tuttavia la API più delicata è sicuramente quella legata all'upload di un file. E' proprio in questa fase che non si possono commettere errori in quanto il materiale che verrà caricato sarà poi alla base della costruzione di un tour virtuale.
Le informazioni da inserire non sono poche e vanno valorizzate correttamente, tuttavia la parte davvero complessa è quella riguardante la creazione di 10 formati di immagini partendo da quella principale che viene caricata dall'utente. La scelta di 10 formati è stata dettata da un bisogno di performance e di versatilità legata al fatto che:

- Non tutti i dispositivi sono uguali ed hanno bisogno degli stessi formati. Per questo 6 di questi tagli sono particolari e dedicati al Samsung Gear VR, o più in generale all'ambiente immersivo con Unity.

- Non tutte le parti del client web necessitano delle foto a risoluzione massima, basti pensare ad una semplice anteprima di un'immagine. Per questo motivo gli altri 4 formati sono basati su grandezze differenti utili a ridurre la quantità di Mbyte da scaricare.

Una volta che il file multimediale viene totalmente trasferito e salvato sulla GridFS con le sue informazioni correlate, parte in automatico il processo di resize nelle varie dimensioni. Per questioni di performance ogni formato viene creato in parallelo in maniera asincorona in modo tale da velocizzare il processo e fornire tutte le dimensioni nel minor tempo possibile.
Per una questione di spazio si era pensato anche ad una soluzione di *resizing a runtime* quando ve ne era bisogno invece di salvare in anticipo tutti formati. Sono state provate entrambe le soluzioni ma i tempi del 'resizing on the fly' erano nettamente insufficienti per soddisfare una richiesta in tempi ragionevoli.

Per i 6 formati relativi al Samsung Gear VR, si è dovuto ricorrere ad una libreria esterna Java chiamata *EquirectangularToCubic* [17] che permette

di trasformare una immagine panoramica equirettangolare in 6 differenti foto cubiche. Si discuterà di questo particolare formato e del suo utilizzo nel capitolo successivo relativo allo sviluppo con il Gear VR, ma la libreria citata è stata particolarmente utile in quanto è l'unica nel suo genere ad essere gratuita. La API lato server per questo tipo di resize, non fa altro che chiamare il file EquirectangularToCubic.jar ed eseguirlo passando come parametro la foto equirettangolare di partenza.

**VirtualTour**

Questo è l'ultimo dei file che compongono il pacchetto di API, al suo interno vi sono solo tre funzioni: una riguardante la ricerca di tour per distanza in base alle coordinate, un'altra legata al salvataggio di un tour virtuale (e di tutte le sue componenti) appena costruito ed in arrivo dal client ed una terza funzione relativa alla richiesta del dettaglio di un singolo tour virtuale dato il suo _id, ovvero la funzione madre per la visualizzazione di un tour, che sia questo sul client web o su uno di tipo immersivo.
Il punto di forza di questa terza API risiede nella sua versatilità in quanto riesce a fornire le informazioni compatibili con i client differenti in base ai parametri che le vengono passati. Se la richiesta è per un dispositivo immersivo questa ritornerà le scene con il dettaglio delle 6 immagini cubiche, altrimenti per il client web questa ritornerà direttamente l'immagine equirettangolare a grandezza *full*.
Di seguito un esempio di risposta JSON che definisce un tour virtuale:

```
{
  "default": {
    "firstScene": "5561b65c6bdfceb22c747f17"
  },
  "scenes": {
    "5562efcd5e94d2703602a62b": {
      "title": "pLeo_A_0011_20121127_01_VL_PA_SP_ex_0R_MO.jpg",
      "autoLoad": true,
      "panorama": "http://ec2-52-28-59-8.eu-central-1.compute.
                  amazonaws.com:8000/download?
                    id=5562efcd5e94d2703602a62b&size=medium",
      "hotSpots": []
    },
    "55657e3e4002899022ae82ba": {
      "title": "pLeo_A_0011_20110615_01_VL_PA_SP_ex_0R_MO.jpg",
      "autoLoad": true,
      "panorama": "http://ec2-52-28-59-8.eu-central-1.compute.
                  amazonaws.com:8000/download?
                    id=55657e3e4002899022ae82ba&size=medium",
      "hotSpots": [
        {
          "pitch": -5,
          "yaw": 92,
          "text": "",
          "type": "scene",
          "sceneId": "5562efcd5e94d2703602a62b"
        }
      ]
    },
    .
    .
    .
  }
}
```

## 4.4 Cartella public e FrontEnd

Completata l'analisi relativa alla struttura del progetto server e delle sue interconnessioni con il database, si è arrivati adesso allo studio di quello che è il primo dei due progetti client. La cartella Public di cui si è parlato nel paragrafo precedente (Sez. 4.3.1) contiene tutto il progetto in Angular.js che lo costituisce. Qui si trovano tutti i codici sorgenti che permettono alla web application di mostrare i risultati e di costruire gli input da mandare al server. In questa sezione verrà studiata per prima cosa la struttura interna delle cartelle di questo client e successivamente si entrerà nel dettaglio delle sue funzionalità.

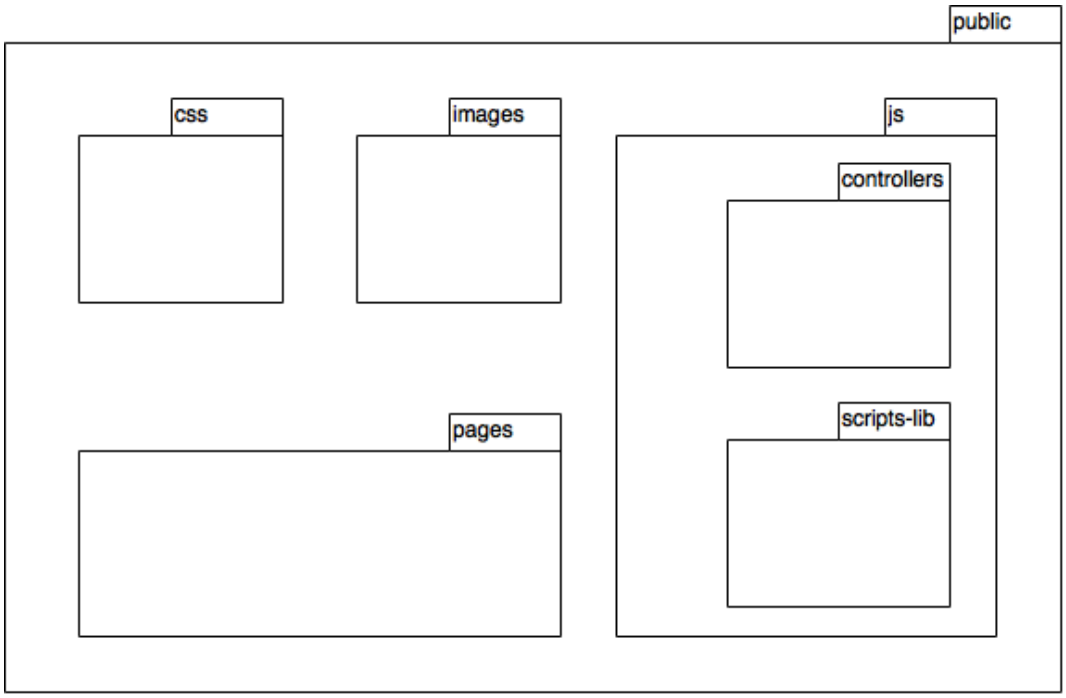


*Figura 4.4: Struttura cartelle prototipo, lato client*

Analizzando il codice sorgente si evince la seguente gerarchia:

- *js*: al suo interno vi è tutto il codice Javascript che regola la logica del client. Data la sua importanza verrà spiegata in modo approfondito nelle sezioni successive. Qui si trovano tutte le classi Controller di Angular.js e gli script delle librerie esterne utilizzate.

- *css*: in questa cartella sono stati inseriti tutti i fogli di stile CSS e i file di font che regolano l'aspetto del front-end.

- *images*: in questa cartella vi sono tutte le immagini che vengono utilizzate nel prototipo per la costruzione dei controlli dell'interfaccia grafica.

- *pages*: al suo interno vi sono tutti i template in codice HTML che compongono le pagine della piattaforma. Il codice di ogni pagina possiede dei *tag* di Angular.js che permettono ad ognuna di esse di interfacciarsi con il relativo Controller ed avere dati dinamici legati con il data-binding di cui si è parlato nel paragrafo 4.2.4. Si discuterà meglio della struttura di queste pagine nelle sezioni successive.

Infine nella cartella Public è situato il file *index.html* che fa da cornice a tutte le pagine web presenti nell'applicazione. Al suo interno troviamo i vari link che collegano i fogli di stile e i vari file Javascript alla pagina.
Conclusa la descrizione della struttura del progetto verranno analizzati i componenti chiave dell'applicazione client, con particolare attenzione nel descrivere gli elementi che costituiscono un'applicazione creata con il framework di Angular.js.

### 4.4.1 app.js

L'app.js è solamente un file all'interno del prototipo ma svolge un ruolo di primaria importanza. Il suo compito è quello di prendere l'url inserito nel browser e di tradurlo nell'effettiva pagina da visualizzare. In esso quindi possiamo trovare l'intero schema di url che il client è in grado di gestire. L'app.js è il primo componente che entra in gioco quando si accede all'applicazione web. Il file in questione si trova nella cartella *public/js/scripts-lib/app.js*.
Gli url che il prototipo è in grado di gestire sono molteplici data la dinamicità dell'applicazione e sono divisi per endpoint legati anch'essi alle risorse chiave dell'applicativo. Per ogni url presente questo file non fa altro che allegare un template.html che andrà visualizzato ed il suo rispettivo Controller che muoverà la logica dietro di esso attuando le richieste per la parte

server e fornendo i risultati da essa. Un esempio di sintassi di questo tipo interna ad app.js:

```
\$routeProvider.
when('/home', {
      templateUrl: 'home.html',
      controller: 'HomeController',
      access: { requiredLogin: false }
    }).
    .
    .
    .
    when('/admin', {
      templateUrl: 'admin.html',
      controller: 'AdminController',
      access: { requiredLogin: true }
    });
```

Naturalmente per l'utilizzo dell'applicazione non è necessario l'inserimento manuale dell'url, il tutto è gestito tramite l'interfaccia grafica.

### 4.4.2 Cartella JS

Come introdotto all'inizio di questo paragrafo, la cartella JS contiene il cuore della logica dell'applicativo client. E' qui dentro che vi sono tutte le implementazioni necessarie che permettono alla web application di mostrare i risultati e di costruire gli input da mandare al server.
Al suo interno questa cartella si divide in due macroaree:

- *controllers*: vi sono tutti file in Javascript che compongono la logica di ogni singola pagina HTML presente nel client. Per ogni template.html infatti esiste un suo corrispettivo file di Controller.js che assolve il ruolo di creare un collegamento tra il model (la fonte dei dati sul server) e la view (la presentazione di essi sul web).

- *scripts-lib*: qui vi sono tutti i file Javascript delle librerie esterne che sono state utilizzate all'interno della piattaforma. Alcune funzionalità, come la visualizzazione di marker e cluster sulla mappa, non so-

no state implementate direttamente ma si è fatto uso di librerie ben più avanzate in modo da presentare un lavoro migliore senza dover 'inventare nuovamente la ruota'.

**Controllers**

Per Controller si intendono delle classi Javascript che vengono considerate come veri e propri oggetti. Questi hanno il controllo dei dati in arrivo dal Model (sul server), che loro stessi richiedono tramite classiche chiamate HTTP. Una volta ricevuti i dati possono decorare la View alla quale sono legati attraverso il Data-Binding di Angular.
In questo prototipo sono stati creati Controller per ogni pagina web disponibile nella piattaforma ed in questa sezione verranno passati in rassegna i più importanti, accompagnati da una descrizione riguardante le loro caratteristiche.

- *MainCtrl*: questo Controller è il primo a cui si fa riferimento una volta arrivati sulla piattaforma in quanto gestisce la possibilità di dare inizio alla fase di login o di signup attraverso la form del prototipo o l'utilizzo dei social.

- *HomeCtrl*: una volta fatto il login, o una volta entrati nel portale come utenti non registrati si arriva in ogni caso nella Home della piattaforma che viene regolata da questo file Javascript. Si occupa della gestione della mappa, delle richieste dei tour virtuali più vicini alla coordinata corrente dell'utente e della ricerca di altri utenti.

- *IndexCtrl*: piccolo file che controlla ad ogni caricamento di una nuova pagina se l'utente che la sta visitando è registrato oppure no.

- *ProfileCtrl e MeCtrl*: controller dedicato alla gestione delle pagine profilo, personali o di altri utenti. In questa sezione vengono fatte le richieste per i dati anagrafici, dei loro tour virtuali e vi è la gestione della funzione di follow/unfollow.

- *NewMediaCtrl*: sicuramente uno dei più importanti, in quanto gestisce la chiamata API per il caricamento di un file sulla piattaforma. Oltre

alla gestione dell'upload, fa anche utilizzo di API legate alla ricerca di luoghi su Google Maps. In questo modo si ha uno standard a cui far riferimento per l'inserimento di dati come *Città, Nazione, Zona* che fanno parte dei dati di un Media sul database.

- *VirtualTourCtrl*: il controller in questione gestisce la visualizzazione di un tour virtuale all'interno della piattaforma. Grazie ad esso vengono decodificate le scene, gli hotspot e le immagini a 360 gradi. Gestisce inoltre le varie funzionalità di transizione tra le scene o di visualizzazione del testo informativo per gli Hotspot di tipo Info.

- *AdminCtrl*: è il file cardine nel suo genere ed è quello su cui si basa tutto il progetto di tesi. E' grazie a questo file che viene creata la struttura di un tour virtuale.
  Questo controller permette di mettere insieme le varie immagini, farle diventare scene, aggiungere gli hotspot nei punti desiderati e di selezionare la prima scena da cui partire. Dopo un check sulla validità del composto creato ha anche la funzionalità di inviare il tutto al server per un altro controllo ed il successivo salvataggio sul database.

In questa lista sono stati descritti i Controller reputati principali. Vi è un'altra buona quantità di file che non verrà citata ma che ha avuto un ruolo comunque importante all'interno della piattaforma e legato ad altre funzioni di contorno quali la cancellazione di un tour virtuale o la sua modifica.

**Codifica posizione Hotspot**

Come è stato detto, il Controller ha il compito di legare l'interfaccia grafica con i modelli salvati sul server, quindi è di sua competenza effettuare anche il processo di validazione dei dati inseriti dall'utente. Tutti i Controller della piattaforma danno una prima validazione al contenuto degli input prima che questi vengano mandati al server, in questo modo si velocizzano alcune dinamiche di errore. Allo stesso tempo però un secondo controllo viene fatto comunque lato server in modo tale da evitare che facendo direttamente una richiesta bypassando il client, si possano salvare dei dati non consoni.

Tra questi controlli, nel file di AdminCtrl vi è anche un processo di decodifica della posizione degli hotspot sulla scena che è importante segnalare. Come spiegato nella sezione 4.3.3, gli hotspot possono essere inseriti in ogni punto della scena equirettangolare in una posizione misurata in gradi che va da [-90, +90] per l'asse Y e [-180, +180] per l'asse X, così da coprire una intera sfera a 360 gradi. Per dare all'utente la possibilità di posizionare uno di questi hotspot in maniera precisa sulla scena tuttavia non è sembrato *user friendly* fornire la sola possibilità di inserire delle coordinate valide manualmente. La soluzione è arrivata offrendo invece la possibilità all'utente di cliccare un punto dell'immagine equirettangolare che corrispondesse esattamente alla posizione dell'hotspot. *Ma come mappare un punto di una foto panoramica con questa convenzione utilizzata per la posizione degli hotspot?*

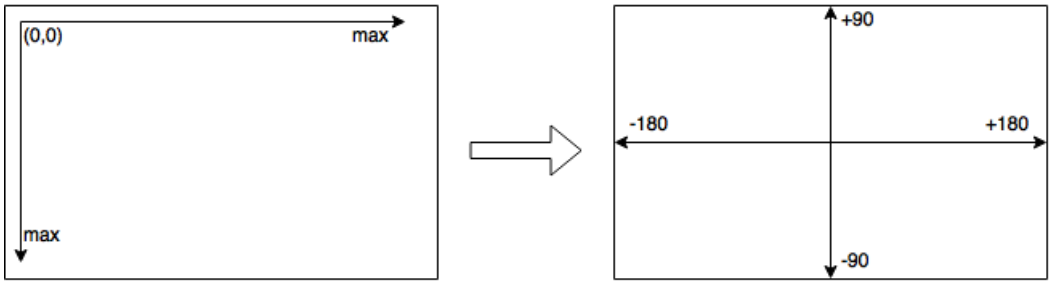


*Figura 4.5: Trasformazione tra i punti di una foto e la convenzione utilizzata per la posizione degli hotspot*

Per risolvere si è dovuto pensare ad una proporzione tra i punti di una foto e questa convenzione che si voleva utilizzare. I punti dall'immagine sono stati presi attraverso i pixel che la compongono e che vanno da [0,0] nell'angolo in alto a sinistra fino a crescere nel loro massimo nell'angolo in basso a destra. Trasformando questi punti con una proporzione si è arrivato a poter mappare ogni pixel della foto panoramica in coordinate angolari che vanno da [-90, +90] per l'asse Y e [-180, +180] per l'asse X come necessario.

**Scripts-lib**

All'interno di questa cartella, come anticipato, vi sono le librerie che sono state utilizzate in simbiosi con il codice sviluppato in modo da aggiungere funzionalità al prototipo. La scelta di usare queste librerie invece di sviluppare del codice proprio è derivata dal fatto che in questo modo si potevano implementare funzioni in maniera rapida, senza dover reinventare qualcosa che già esisteva e sopratutto avendo un supporto agli errori maggiore, dato dalla maturità che una libreria già rodata da tempo acquisisce. Così per questo motivo si è fatto uso di tre librerie per altrettante funzioni all'interno della piattaforma web.

- La prima di queste si chiama *Leaflet* [1], ed è una libreria Open-Source, Mobile Friendly per la creazione di mappe interattive. E' grazie a questa che la Home della piattaforma ha una mappa iniziale con dei *segnaposto* che rappresentano i punti in cui sono geo-posizionati i tour virtuali degli utenti. Il suo utilizzo è servito alla visualizzazione della mappa e alla creazione di questi segnaposto in cluster quando questi sono molto vicini tra loro.

- La seconda si chiama *angular-social.js* [5], e più che una vera e propria libreria è un modulo per Angular che aiuta nel primo interfacciamento con i social per la fase di login. Il suo utilizzo è servito nella fase di contatto tra la piattaforma e i social per la validazione dei permessi e per la restituzione del primo ID di verifica che successivamente andava nuovamente validato dalla piattaforma sviluppata per controllare che l'utente avesse davvero un profilo Facebook o Google Plus. Nulla di particolarmente complesso ma l'impiego della libreria ha fatto risparmiare tempo nell'implementazione.

- La terza, ma la più importante in assoluto, si chiama *Pannellum* [16]. Questa libreria è quella che è alla base della visualizzazione dei panorami a 360 gradi in quanto una volta che l'immagine equirettangolare arriva al client è lei che la decodifica e la rende esplorabile attraverso l'uso del mouse o del touch.
  La libreria è stata tuttavia modificata per renderla compatibile con i

formati JSON provenienti dal servizio server del prototipo. In principio infatti questa libreria poteva solo leggere formati JSON salvati su un documento all'interno di se stessa che andava sostituito di volta in volta per la visualizzazione di scene differenti.

### 4.4.3 Cartella pages ed Interfaccia grafica

Finito con la descrizione dei Controller e della logica generale del client web, in questa sezione verrà spiegato come è stata organizzata l'interfaccia utente, dalla divisione delle funzionalità all'interno delle varie pagine alle scelte grafiche di ognuna di esse.
L'interfaccia grafica è composta principalmente dal codice HTML presente nella cartella *public/pages*, dai fogli di stile CSS presenti in *public/css* e dalle immagini presenti in *public/images*. Tralasciando le ultime due di cui non vi è nulla di rilevante ai fini di questo lavoro, il cuore della sezione si concentrerà invece nella esamina dei template.html presenti nel progetto client.

I template definiscono l'interfaccia grafica presente nel prototipo ed in questi file si può trovare il codice HTML che compone tutte le pagine.
Ad essere più precisi i template presenti non contengono solo codice HTML, ma essendo in relazione con i Controller di Angular.js portano al loro interno i riferimenti a questi file Javascript. In aggiunta al puro codice HTML i template quindi possono contenere:

- *espressioni*: come {{title}} che permettono di ottenere informazioni da una variabile nel Controller legato al template. Le espressioni sono fondamentali per mantenere aggiornata la View.

- *outlets*: sono simboli che vengono utilizzati per richiamare altri template. Gli outlet permettono una migliore organizzazione e una maggiore riusabilità del codice.

- *componenti*: sono degli elementi HTML che si possono usare per creare dei controlli ad hoc che hanno delle funzioni specifiche.

Se si visionasse la cartella *public/pages*, si noterebbe che i template sono più di 15. Come spiegato nella sezione 4.4.1, l'app.js ha il compito di mostrare i template corretti in base all'url presente nel browser. Grazie a questo forte connubio tra l'app.js e i template è possibile creare una interfaccia grafica gerarchizzata dove ogni pagina ha la propria logica di esecuzione. In questo modo è possibile creare un'interfaccia complessa in modo scalabile. Adesso verranno passate in rassegna le pagine HTML principali del progetto, presentando le funzionalità più interessanti che portano con loro.

### Home

Questa è la prima pagina che si raggiunge una volta arrivati sulla piattaforma web. Come anticipato nella descrizione del suo Controller, questa interfaccia permette di accedere a tutti i tour virtuali presenti nelle vicinanze in quanto all'accesso viene richiesta la posizione del dispositivo dal quale si sta visualizzando l'applicazione (in caso di risposta negativa verranno caricate le coordinate di Milano). Altre possibilità invece sono quella di cercare un utente per nome, inserire una coordinata geografica, oppure se si è registrati, di entrare nella propria area personale.

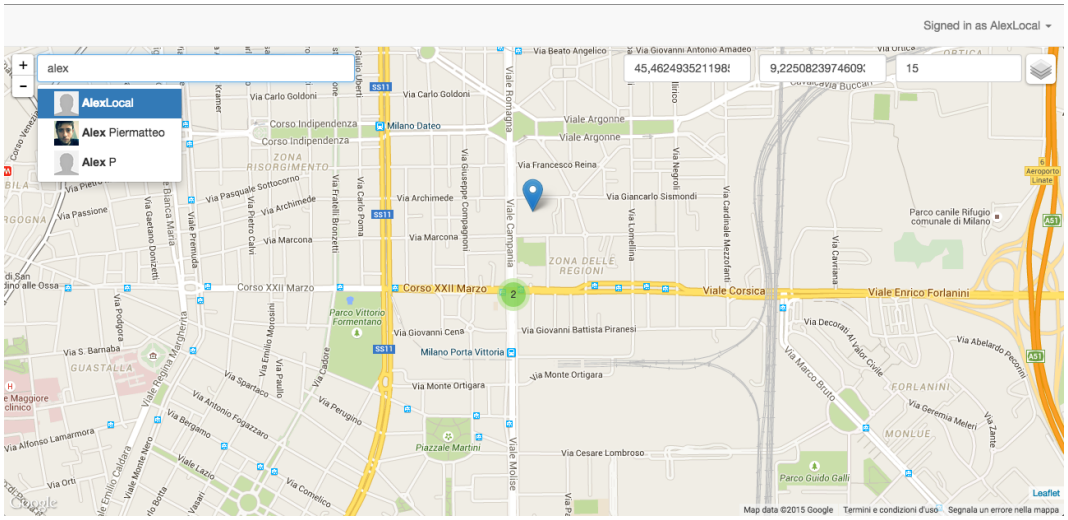


*Figura 4.6: Schermata home, piattaforma web*

### New Media

Grazie a questa interfaccia grafica è possibile caricare un file multimediale di vario tipo, anche se al momento la piattaforma web per le funzionalità di visualizzazione dei tour virtuali supporta solo le immagini di tipo JPEG o PNG a 360 gradi.
Per l'upload del file ogni campo risulta obbligatorio, una volta premuto il pulsante partirà una barra di avanzamento che mostrerà il tempo mancante alla conclusione del caricamento.

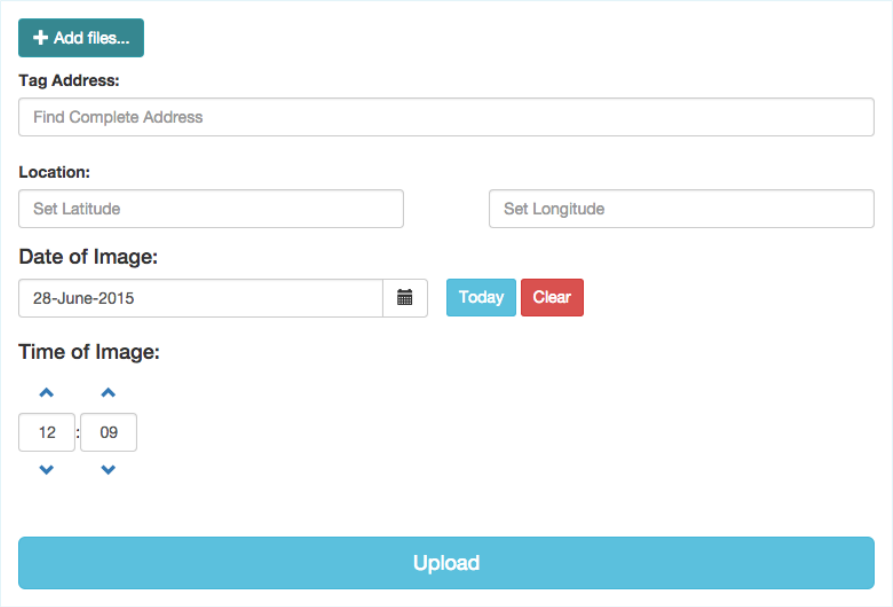


*Figura 4.7: Modal del file uploader, piattaforma web*

### Tour Virtuale

Il visualizzatore di un tour virtuale passa da questa pagina che presenta al suo interno la prima scena a tutto schermo. Da lì grazie alla presenza dei vari hotspot è possibile transitare tra le varie scene esplorabili o di far uscire informazioni testuali in primo piano riguardanti punti precisi all'interno dell'ambiente a 360 gradi. Le immagini ambientali sono panorami equirettangolari interi e nonostante alcune di esse possano raggiungere i 20MByte i caricamenti sono molto reattivi.

**Admin, generatore di Tour Virtuali**

Questa pagina, insieme a quella relativa al caricamento dei file multimediali è per certo quella più importante. Grazie ad essa è possibile assemblare in poche e facili mosse un tour virtuale, in totale autonomia. L'interfaccia si presenta con la lista sulla destra dei file caricati dall'utente, quando se ne seleziona uno questo si apre al centro portando in primo piano dei nuovi comandi. Tra questi vi sono:

- la possibilità di selezionare la foto corrente come una da aggiungere al tour virtuale.
- la possibilità di marcare la foto corrente come foto principale;
- la possibilità di aggiungere un hotspot di transizione o di informazione. Una volta aggiunto, è possibile scegliere una posizione in cui inserirlo cliccando direttamente su un punto della foto. Questo permette di avere l'hotspot esattamente dove lo si vuole in maniera davvero precisa.

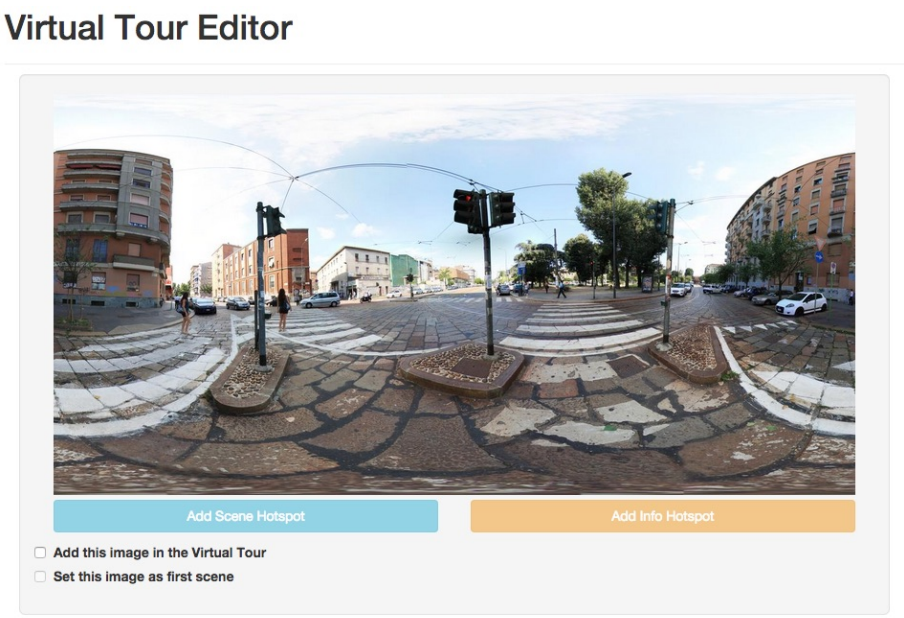


*Figura 4.8: Dettaglio centrale dell'interfaccia per la generazione di un tour virtuale*

Durante la costruzione di un tour virtuale vi è un controllo sempre attivo che si assicura che tutto sia coerente, come ad esempio il fatto che non è possibile inserire due scene come principali.

## 4.5 Conclusioni

In questo capitolo ho spiegato nel dettaglio il funzionamento del prototipo web, l'applicazione che permette l'upload di file multimediali e la creazione di un tour virtuale attraverso un'interfaccia semplice e intuitiva. In questo modo gli utenti grazie a questa piattaforma gratuita potranno caricare i propri contenuti senza vincoli e creare, salvare ed editare come meglio credono il proprio tour virtuale.
La piattaforma farà da hosting a tutti i tour generati dando così la possibilità agli utilizzatori di non dover comprare uno spazio dedicato per lasciar visualizzare il proprio lavoro. Per concludere, come dalle intenzioni iniziali, il servizio proposto è stato architettato in maniera interamente RESTful, e fornisce API ad eventuali sviluppatori esterni, in maniera gratuita, che permettono a chiunque ne abbia voglia di creare un proprio client per la costruzione di tour virtuali sfruttando la logica sul server di questa piattaforma proposta.
Nonostante tutto l'applicativo web è solo la prima parte del progetto: la seconda parte riguarda lo sviluppo di un altro applicativo client che si possa interfacciare con lo stesso backend, per la visualizzazione in maniera totalmente immersiva dei tour virtuali creati . Questo è il compito dell'applicazione dedicata al Samsung Gear VR, il progetto che verrà spiegato in dettaglio nel capitolo successivo.

# Capitolo 5

# GearVR nel progetto

## 5.1 Introduzione

La seconda parte di questo progetto di tesi riguarda lo sviluppo di un secondo client per la visualizzazione di tour virtuali. Questo a differenza del primo permetterà la visione dei tour attraverso un dispositivo di Realtà Virtuale cercando di rendere l'esperienza totalmente immersiva.
L'idea che c'è dietro è quella di creare un'applicazione per il Samsung Gear VR che permetta di interfacciarsi con il server, implementato e descritto nel capitolo precedente, utilizzando le API RESTful che vengono sfruttate dall'altro client web. Questo permetterebbe di rendere l'applicazione particolarmente scalabile senza la problematica di aggiungere al suo interno già tutti i dati dei tour da visualizzare, come invece accade con le applicazioni studiate e discusse nella sezione 2.1.2. Inoltre così non sarà necessario aggiornare l'applicativo tramite gli store ufficiali per aumentarne i contenuti oppure dover costringere l'utente ad inserire manualmente i propri file all'interno del dispositivo per poi visualizzarli. Questo scambio di informazioni con il server della piattaforma tuttavia dovrà essere abbastanza veloce e performante da rendere l'esperienza utente piacevole, la sfida di questo secondo sviluppo risiede tutta in questa condizione fondamentale.
Nel capitolo verrà fatta una introduzione su Unity, si descriverà com'è stata creata quest'applicazione per Gear VR, quali scelte architetturali sono state utilizzate ed infine vi sarà una spiegazione su alcune ottimizzazioni apportate che hanno permesso di velocizzare parti dell'applicativo che

risultavano particolarmente lente.

## 5.2 Sviluppo con Unity

Unity è un Game Engine cross-platform sviluppato da Unity Technologies e usato principalmente per lo sviluppo di videogames per PC, console oppure dispositivi mobile. Il suo Game Engine è costruito su Mono, una implementazione open-source del .NET Framework che utilizza linguaggi di programmazione come il Javascript, C# o il Boo (linguaggio ispirato al Python).

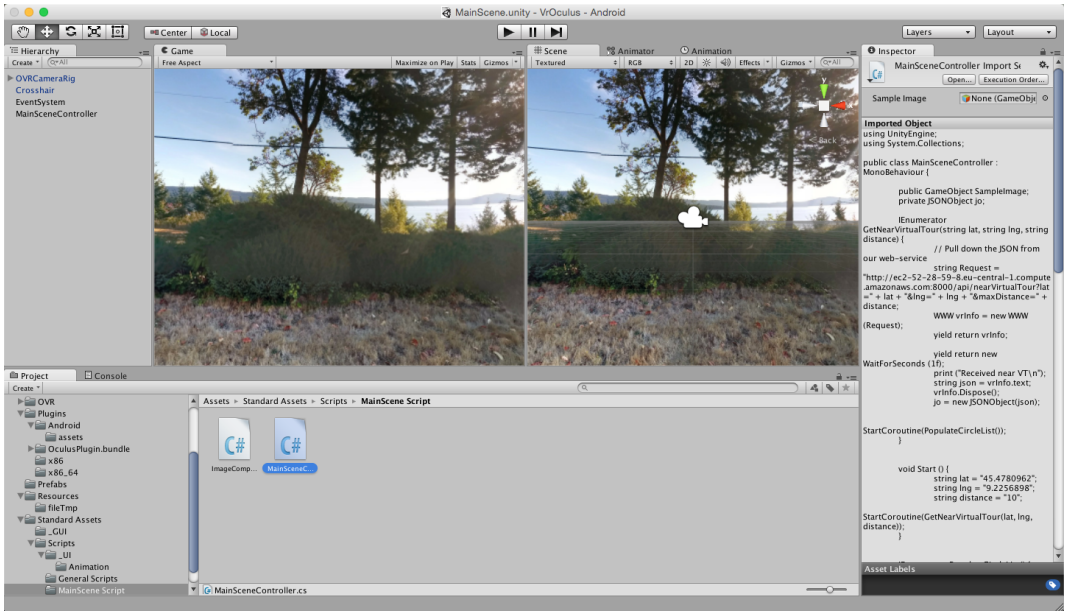


*Figura 5.1: IDE di Unity per lo sviluppo dell'applicativo su GearVR*

Su questo IDE (Integrated development environment) di sviluppo Rift, la società che fornisce l'SDK per programmare su Samsung Gear VR, ha costruito un *Integration Package* di applicazioni di esempio, classi di supporto e oggetti prefabbricati che possono essere inseriti già perfettamente operativi all'interno del progetto e rendere lo sviluppo più semplice.
Il pacchetto intero di funzionalità pesa 5MByte e presenta una struttura interna di questo tipo:

- *Editor*: contiene scripts che arricchiscono l'editor di Unity con nuove funzionalità per lo sviluppatore. Non aggiungono nulla al programma creato ma sono utili in fase di sviluppo.

- *Materials*: contengono materiale grafico utile per determinati sviluppi sulla GUI.

- *Moonlight*: vi sono delle classi specifiche per il Gear VR e lo sviluppo strettamente mobile.

- *Prefabs*: la vera classe di supporto all'interno del pacchetto, in quanto fornisce i due oggetti OVRCameraRig e OVRPlayerController. Questi due prefabs permettono di visualizzare una scena in modalità stereoscopica senza dover fare altro.

- *Resources*: contengono altri prefabs che vengono richiamati da script presenti nel pacchetto.

- *Scripts*: contengono scripts in C# che vengono usati insieme ad alcuni prefabs per fornire, sulla carta, funzionalità altrimenti difficili da scrivere. In verità con lo sviluppo di questo prototipo è stato più semplice alcune volte implementare da zero alcune di queste funzioni invece di capire come integrare questi scripts troppo carichi di impostazioni.

- *Scenes*: contengono alcune scene di esempio da cui prendere spunto.

### 5.2.1 Cartella Assets e struttura del progetto

Il pacchetto fornito da Rift, una volta scaricato, per funzionare ha bisogno di essere inserito nella cartella madre del nuovo progetto Unity, la cartella *Assets*. In questo spazio si possono inserire tutti i pacchetti che si reputano necessari, sul web infatti esiste uno store proprietario di Unity dove è possibile scaricarne di nuovi, alcuni gratuiti ed altri a pagamento, che forniscono funzionalità aggiuntive senza scrivere codice. In questo progetto non è stato necessario implementare alcun pacchetto esterno ad eccezione di quello fornito da Rift, tuttavia la struttura del codice implementato parte ugualmente da questa cartella.

Analizzando il codice sorgente si può facilmente identificare la seguente struttura interna.

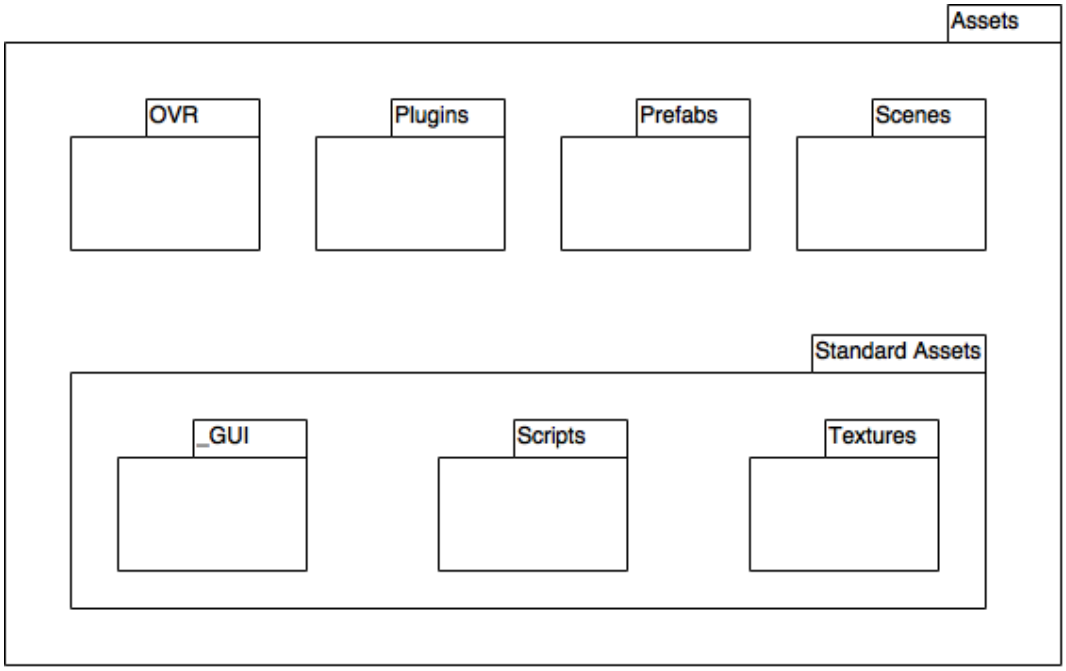


*Figura 5.2: Struttura del progetto GearVR*

**OVR**

Contiene l'intero Integration Package fornito da Rift che è stato di supporto allo sviluppo, sopratutto per l'utilizzo della sua OVRCameraRig per la visuale stereoscopica.

**Plugins**

In questa cartella vi è tutta quella parte che fa da collegamento al sottostrato applicativo Java di Android. Qui andrebbero inseriti i plugin .jar che possono essere richiamati da Unity per attivare funzionalità della VrLib come descritto nel capitolo 2.5.

**Prefabs**

In questa cartella si possono inserire degli oggetti chiamati Prefabs. Questi oggetti non sono altro che elementi costruiti dallo sviluppatore, attraverso

l'IDE, che poi vengono presi come modello di riferimento. Ogni Prefabs potrà quindi poi essere caricato dagli script già pronto per l'utilizzo.

#### Scenes

Contiene le scene costruite con l'IDE. Le scene sono un formato particolare proprietario di Unity che, come una build di un classico programma, vengono generate prima di far partire l'esecuzione.

#### Standard Assets

Questa cartella è la più importante dell'intero progetto in quanto contiene il nucleo centrale dell'applicativo. E' divisa ulteriormente in 3 sottocartelle: *_GUI, Scripts, Textures*. La prima e la terza cartella racchiudono dei materiali necessari alla costruzione di alcuni elementi grafici all'interno dell'applicazione, la seconda invece è di fondamentale importanza poichè contiene tutta la logica del client, è a sua volta suddivisa in cartelle e verrà approfondita di seguito.

- *_UI*: in questa cartella vi è tutta la logica riguardante alcune parti grafiche come la visualizzazione del testo sugli hotspot di informazione all'interno della scena o quella del crosshair (mirino) al centro della visuale stereoscopica.

- *General Scripts*: qui vi sono tutti gli script che regolano la logica dietro la creazione e la transizione delle scene, dei loro hotspot e delle loro immagini a 360 gradi di contorno.

- *Main Scene Scripts*: in quest'ultima cartella invece vi sono gli script che regolano la visualizzazione delle anteprime dei tour virtuali disponibili da selezionare. Esattamente come accade sul client web, qui è possibile scegliere un tour per poi arrivare sulla sua prima scena facendo partire tutta l'esplorazione.

### 5.2.2 Fasi di sviluppo

In questa sezione verranno esposte le fasi dello sviluppo di questa applicazione ideata per un dispositivo molto particolare come il Samsung Gear VR. Si parlerà delle necessita che sono nate mano a mano che lo sviluppo andava avanti e di come si sono superati problemi incontrati durante la programmazione. L'obiettivo di questo lavoro è stato quello di creare un client capace di interfacciarsi in maniera rapida con il servizio server precedentemente realizzato. Andava fatto però senza scaricare una mole di dati troppo impegnativa in quanto il dispositivo all'interno del Gear VR è pur sempre un telefono, quindi con una memoria limitata, e soprattutto perchè questa applicazione doveva poter essere usata con una connessione dati mobile senza consumare però tutto il traffico di un utente.
Per quanto concerne le funzionalità, l'idea di base è stata quella di sviluppare un visualizzatore capace di fare le stesse cose che proponeva anche quello del client web, ovvero una esplorazione dei panorami a 360 gradi, la presenza di hotspot sospesi con cui poter interagire, apprendere da informazioni nell'ambiente circostante o di spostarsi tra le scene del tour. Non ultimo in importanza infine, la possibilità di scegliere tra dei tour virtuali nelle vicinanze.

#### Creazione dell'ambiente a 360 gradi

Una volta installato tutto il necessario per cominciare lo sviluppo, la prima parte del progetto si è focalizzata sulla creazione di una singola scena. Questo perchè la Scena è l'elemento base di un tour virtuale ed è in essa che sono contenute tutte le informazioni di un ambiente immersivo. Un tour virtuale in fondo non è altro che un raccoglitore di scene, quindi una logica di sviluppo bottom up che parte da una di esse per arrivare ad un tour è sembrata la scelta più logica.

Il primo passo è stato quello di costruire un 'Punto di Vista' per l'utente che doveva essere capace di osservare la scena immersiva senza però potervisi muovere all'interno. Questo risultato, come anticipato nei capitoli precedenti, si è immediatamente ottenuto grazie all'utilizzo del Prefabs *OVRCameraRig* fornito con il pacchetto per Unity della Rift. E' stato solo ne-

cessario inserire l'oggetto all'interno della scena di sviluppo, a quel punto tutto ha funzionato dal primo momento.

Creato il Punto di Vista, la maggior parte degli sforzi si sono incentrati sulla visualizzazione della foto panoramica che doveva avvolgere l'utente a 360 gradi nella scena. Questa foto doveva arrivare attraverso una richiesta al servizio RESTful del server, dato che l'obiettivo prefissato era quello di non salvare nulla precedentemente sul dispositivo.
Una volta scaricata l'immagine, per la visualizzazione si è utilizzato l'oggetto Skybox in quanto l'unico del framework Unity capace di renderizzare correttamente un panorama equirettangolare.
Uno Skybox è un contenitore attorno all'intera scena di sviluppo e, di solito, nella creazione di videogames viene utilizzato per la renderizzazione degli sfondi e dei paesaggi al di là del campo di gioco.

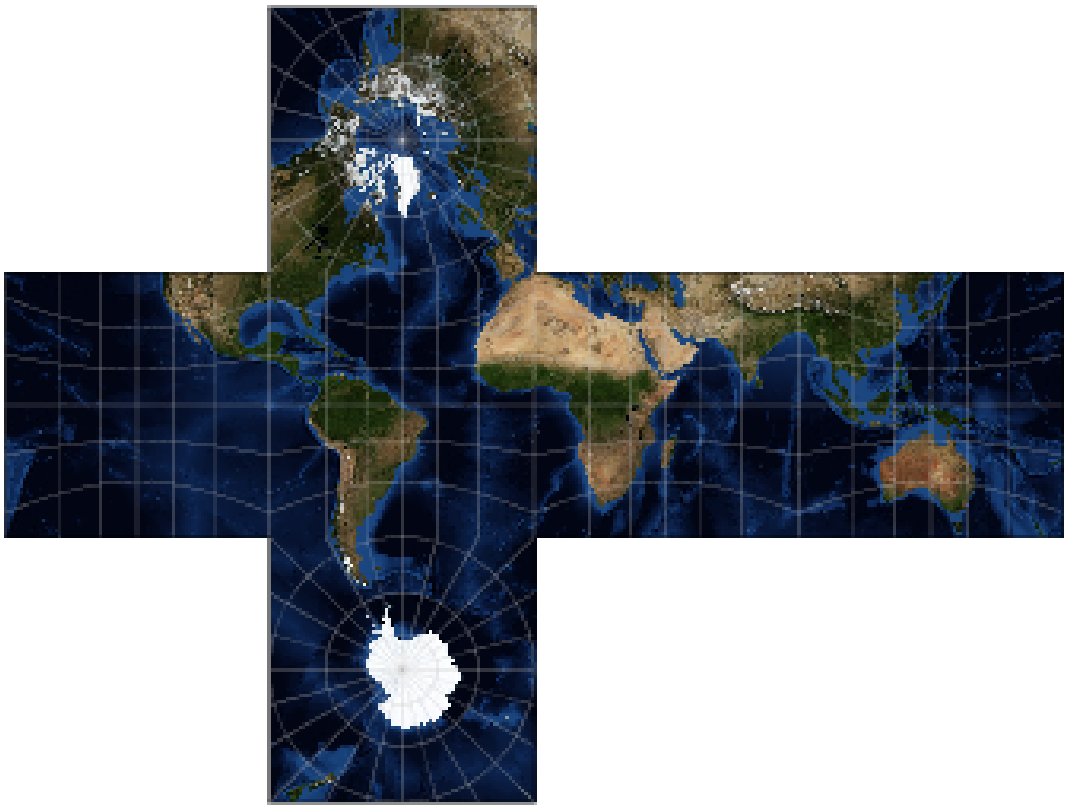

*Figura 5.3: Esempio di immagine cubica costruita su una immagine della terra equirettangolare*

Purtroppo per la sua visualizzazione non è stato possibile utilizzare una foto equirettangolare intera poiché lo Skybox per funzionare ha necessità di 6 foto cubiche derivanti da essa. Per questo motivo sul server, come descritto nel capitolo precedente (sezione 4.3.4), sono state salvate 6 immagini cubiche della foto panoramica originale attraverso una trasformazione particolare.
Ottenute sul dispositivo le 6 immagini che compongono l'ambiente circostante, attraverso lo script in C# ImportSkyboxImages presente in *Assets/-Standard Assets/General Scripts*, queste sono state dinamicamente fuse insieme in un Material di nome *CubeMap* inserito all'interno dello Skybox. Lo skybox infine, una volta attivato ha riempito l'orizzonte dell'ambiente con l'immagine panoramica già renderizzata. Questa delicata fase di caricamento verrà spiegata nel dettaglio nella sezione 5.4.1.

**Inserimento di un Crosshair per il puntamento degli oggetti**

Ottenuta un scena con all'interno un'immagine a 360 gradi totalmente esplorabile, è subito nato il bisogno di creare un mirino al 'centro dello sguardo' in modo da poter puntare degli oggetti, nello specifico i futuri hotspot, a distanza nell'ambiente virtuale e poter così scatenare delle Azioni.
La gestione di questa parte è stata affidata allo script *Reticle.cs* presente in *Assets/Standard Assets/General Scripts*, il quale non fa altro che visualizzare sulla scena un Prefabs a forma di mirino costruito ad hoc. Il codice dinamicamente ad ogni frame prende questo Prefabs e ne regola la distanza dalla OVRCameraRig, la grandezza e la posizione al centro dell'angolo di visione dell'utente. In questo modo il mirino risulta sempre nel punto centrale della visuale e con la grandezza e la distanza regolate a runtime in base alla presenza di oggetti sullo schermo. Questo perchè il mirino non finisce mai dietro un oggetto ma gli si pone sempre davanti, e per una questione di proporzioni se l'oggetto è molto vicino alla Camera anche il mirino diventa più grande.
Lo scaling del mirino viene calcolato moltiplicando la grandezza originale del Crosshair per un coefficiente (*distance*) che cambia in base alla distanza

in cui il mirino stesso è posizionato, secondo la seguente formula:

$$distance = distance * (1 + e^{-distance})$$

Attraverso questo coefficiente logaritmico, dopo una distanza superiore ai 10 metri la grandezza del mirino non si riduce ulteriormente e rimane costante.
Oltre a gestire l'aspetto grafico lo script invia un raggio invisibile di nome *Raycast* che parte dalla OVRCameraRig e che interseca il centro del mirino puntando all'infinito. In questo modo nel prossimo paragrafo verrà spiegato come gli hotspot colpiti da questo raggio capiscano di essere stati puntati e rispondano con azioni ben definite.

**Importazione degli Hotspot e le loro tipologie**

La creazione degli Hotspot all'interno della scena è gestita dallo script *ImportSpheres.cs* presente in *Assets/Standard Assets/General Scripts*. Una volta richiesto il JSON di un intero tour virtuale questo file decodifica gli hotspot della scena corrente e crea delle sfere di grandezza e colore diverso in base se questi siano Hotspot di tipo 'Scene' che servono per le transizioni o di tipo 'Info' che invece mostrano del testo informativo.
Il primo grande problema da risolvere è stato quello di capire come mappare la posizione degli Hotspot in due dimensioni X e Y provenienti dal server con una posizione a 3 dimensioni necessaria in un ambiente immersivo. Per risolvere questo problema è stato necessario applicare qualche formula di geometria dei solidi e delle coordinate sferiche:

Come accennato nella sezione 2.2, sappiamo che un'immagine equirettangolare altro non è che una rappresentazione piana di una immagine sferica e quindi è possibile considerarla con tale forma. Sappiamo inoltre che i punti degli Hotspot mappati attraverso il client web e che arrivano attraverso il JSON del tour virtuale, corrispondono ai gradi sull'asse X e sull'asse Y rispetto al punto (0,0).
Da queste condizioni iniziali, dato uno spazio cartesiano XYZ dove Y rappresenta l'altezza e Z la profondità, si è potuta immaginare la posizione di un Hotspot come un punto sulla superficie di una sfera con raggio dal

centro definito a piacere, mentre la posizione della OVRCameraRig come il punto in (0,0,0), nonché centro della sfera. A quel punto è bastato proiettare tali punti sulla superficie della sfera ortogonalmente con il piano XZ e con l'asse Y per avere il punto trasformato in coordinate tridimensionali.

Le formule utilizzate per la trasformazione sono le seguenti:

$$x = \rho \sin(90 - \beta) * \sin(\alpha)$$
$$y = \rho \cos(90 - \beta)$$
$$z = \rho \sin(90 - \beta) * \cos(\alpha)$$

Dove il punto $(\alpha, \beta, \rho)$ indica la posizione dell'hotspot sulla superficie della sfera con angolo $\alpha$ sull'asse X, con angolo $\beta$ sull'asse Y e $\rho$ la distanza, tutto rispetto al centro della sfera posizionata nell'origine.
Il punto (x,y,z) invece indica la nuova posizione nello spazio tridimensionale costruita sulle proiezioni del punto $(\alpha, \beta, \rho)$ con il piano XZ e l'asse Y.

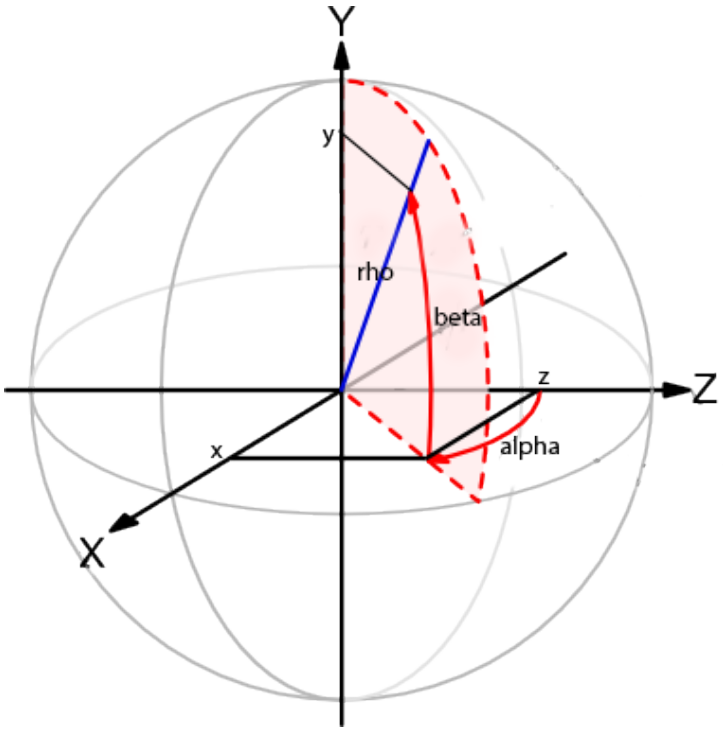


*Figura 5.4: Proiezioni utilizzate per trasformare un punto da due dimensioni a tre dimensioni*

Ottenuti i nuovi punti nello spazio tridimensionale,gli hotspot sono stati posizionati e, come accennato prima, sono stati costruiti dinamicamente con

colore e grandezze diverse in base alla loro tipologia. Tuttavia la differenza non è esclusivamente nell'aspetto ma anche negli script che questi hotspot differenti contengono. In Unity infatti è possibile allegare ad ogni oggetto più di uno script che poi può regolare il comportamento di tale oggetto. Agli Hotspot di tipo Scene è stato allegato il file *HotspotScene.cs* mentre a quelli di tipo Info il file *HotspotInfo.cs* che ora verranno descritti più nel dettaglio:

- *HotspotScene.cs*: lo script controlla lo stato dell'oggetto a cui è collegato verificando che questo sia o meno osservato dal mirino. Una volta scatenata l'azione di 'transizione' dopo l'osservazione dell'oggetto, lo script dell'Hotspot corrente invia un messaggio al *MainSceneManager.cs*, che regola tutta la scena. Nel messaggio viene passato come riferimento l'indirizzo _id della nuova scena da caricare che l'hotspot possiede dentro di sé e a cui è collegato, in modo da far partire tutto il processo di cambio scena e la nuova chiamata agli script ImportSkyboxImages.cs e ImportSpheres.cs che ricaricheranno rispettivamente un nuovo panorama a 360 gradi e le nuove sfere riferite alla nuova scena.

- *HotspotInfo.cs*: anche questo script controlla lo stato dell'oggetto a cui è collegato verificando che questo sia o meno osservato dal mirino. Una volta scatenata l'azione di 'apertura testo' dopo l'osservazione dell'oggetto, lo script dell'Hotspot corrente invia un messaggio ad un oggetto presente sulla scena e reso nascosto di nome *Canvas*. Il Canvas è un elemento grafico che permette di creare delle GUI all'interno dello spazio tridimensionale e per il progetto è stato reso simile ad un pannello contenente del testo. Nel messaggio infatti viene passata la descrizione che si vuole far visualizzare all'utente e che l'hotspot possiede dentro di sé. Ricevuto il messaggio il Canvas in automatico grazie allo script *Typer.cs* a lui collegato, inserisce con una animazione il testo al suo interno e si posiziona di fianco all'hotspot che ha scatenato l'evento.

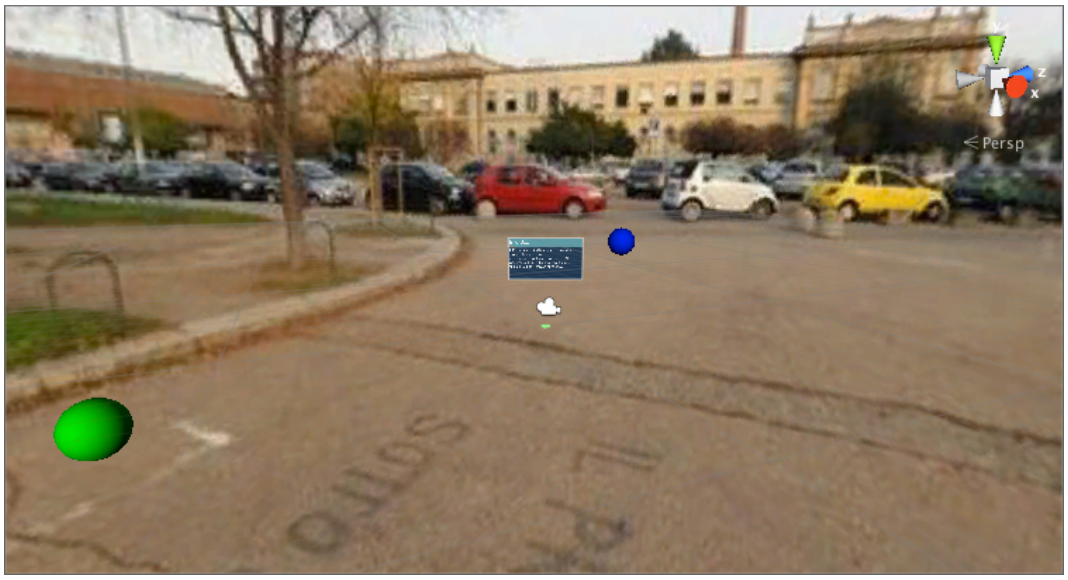


*Figura 5.5: Costruzione del Canvas per la visualizzazione delle descrizioni per gli hotspot di tipo Info*

Per individuare l'evento di puntamento di un oggetto è stato utilizzato il raggio di nome *Raycast* spiegato poco più su nella sezione sul Crosshair. Gli script sugli Hotspot controllano ad ogni frame che il Raycast non li intersechi, ma in caso affermativo fanno partire un timer. Se il timer raggiunge il tempo stabilito di 3 secondi, scatta l'azione corrispondente.
Si è pensato ad un utilizzo di questo tipo invece che all'uso di un Tap sul trackpad laterale del Gear VR, poichè in questo modo si evita all'utente di fargli portare le mani al volto per l'esecuzione di ogni singola funzionalità.

**Tour Virtuale e scelta di quelli vicini**

Una volta costruita l'intera scena immersiva, completa di panorama a 360 gradi, di un mirino per puntare gli oggeti presenti, la presenza di vari hotspot capaci di fornire informazioni o di sostituire la scena corrente con un'altra collegata ad essa, il tour virtuale è risultato quasi totalmente pronto. Il passo seguente infatti è stato quello di fornire una scena principale in cui l'utente potesse accedere per visualizzare tutte le anteprime dei tour virtuali disponibili. Per questo motivo è stata sviluppata una nuova *MainScene* in cui l'utente viene inserito all'avvio dell'applicazione.
Esattamente come funziona per il client web, in questa 'Stanza Virtuale' ven-

gono mostrati in cerchio tutti i Tour presenti sul server e che si trovano a non più di una certa distanza dal punto in cui si trova fisicamente l'utente.

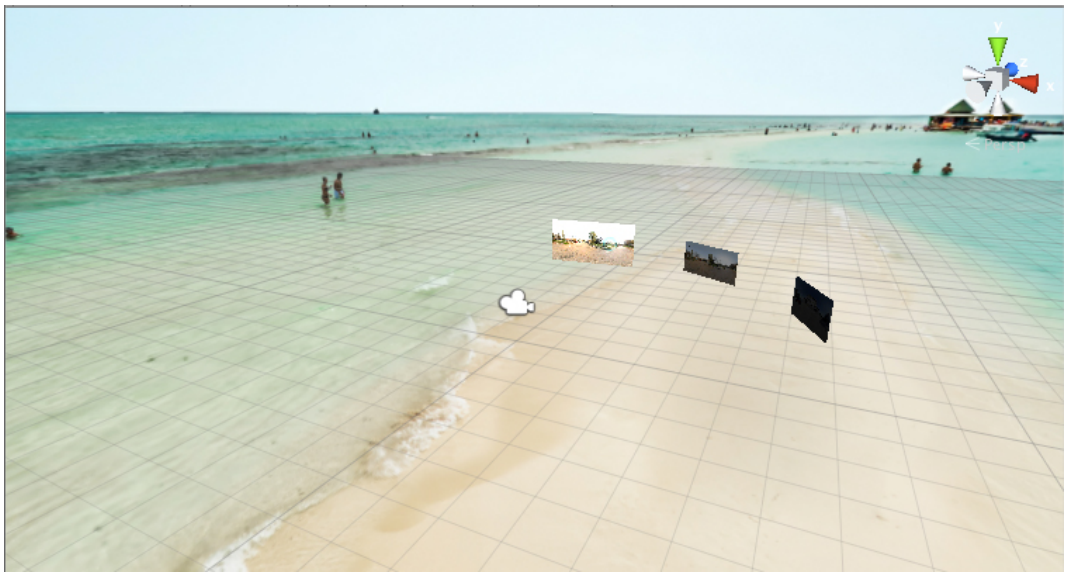

*Figura 5.6: Costruzione della MainScene, la Camera al centro è il punto di vista dell'utente posto davanti alle anteprime dei tour virtuali*

La gestione di questa MainScene è sotto il controllo di due soli script in C#:

- *MainSceneController.cs*: questo file è affidato alla OVRCameraRig, l'unico elemento dello *scenario* (discusso nella sezione 5.2.1) che non viene mai distrutto quando si cambia Tour Virtuale o uno scena al suo interno. Quando avviene la selezione di un Tour infatti, lo scenario cambia e ne viene caricato un altro in cui parte tutto il caricamento del tour nuovo selezionato. In questa fase gli elementi della scena precedente vengono tutti distrutti per una questione di performance ad eccezione della MainCamera che resta sempre attiva.
  Per questo motivo lo script è sempre operativo durante tutta l'esecuzione dell'applicazione. E' un fattore molto importante in quanto essendo l'unico che non viene mai distrutto in un cambio di scenario ha la gestione dei compiti di più alto livello in assoluto.
  E' questo file infatti che gestisce la prima chiamata al server per il JSON di risultati dei tour virtuali vicini, che dispone le anteprime dei tour in cerchio o che invoca la chiamata per il caricamento del tour da

visualizzare. Allo stesso tempo restando attivo anche all'ingresso in un altro scenario, questo script può essere richiamato per ricaricare la MainScene principale attraverso il tasto *Back* fisico del Gear VR.

- *ImageComponent.cs*: questo script controlla lo stato dell'oggetto anteprima a cui è collegato. Infatti ogni anteprima di un tour virtuale possiede questo file come allegato che, come gli script legati agli hotspot, ha il compito di capire se il mirino posto al centro dello schermo sta osservano o meno l'anteprima. Questo perchè attraverso un Tap sul trackpad è possibile scegliere l'elemento che si sta guardando facendo partire il tour desiderato.
  In questo caso, per la selezione, si è deciso di utilizzare il trackpad e non solo lo sguardo poiché oltre la funzione di selezione, l'hardware del dispositivo viene in aiuto anche per lo scrolling delle anteprime. Con una operazione di swipe sul trackpad infatti è possibile scorrere le anteprime sospese senza dover necessariamente girarsi con lo sguardo.

## 5.3 Debug con il Gear VR

Durante la fase di sviluppo, come è necessario fare per ogni genere di applicazione da produrre, è stato fondamentale entrare in modalità debug per scovare l'origine di alcuni problemi che si presentavano durante l'esecuzione e che difficilmente era possibile trovare in maniera differente.
Solitamente nello sviluppo di un applicativo mobile, il modo più semplice per accedere ai LOG è quello di mantenere il dispositivo connesso attraverso la porta USB al computer da dove si fa partire l'applicazione. In questo modo, esattamente come per la fase di bulid, tutti i dati passano per il cavo favorendo un debug semplice da utilizzare.
Il Samsung Gear VR tuttavia non può funzionare con questa modalità di debugging in quanto la porta USB viene già occupata dal Gear che utilizza quella periferica per interfacciarsi e scambiare dati con il dispositivo.

### 5.3.1 Utilizzo di ADB in WiFi

Non potendo utilizzare un cavo per il debug l'unica metodologia possibile è stata quella di utilizzare ADB [7], ovvero l'*Android Debug Bridge* di Google. L'ADB è incluso nell'Android SDK, che è un requisito fondamentale per lo sviluppo del progetto, ed è il tool a linea di comando principale per comunicare con un dispositivo Android per il debug.
Usando adb attraverso la shell del sistema operativo, è possibile connettersi e comunicare con un dispositivo Android anche via TCP/IP attraverso una connessione WiFi. A patto che sulla macchina, oltre all'SDK come accennato in precedenza, sia installata una versione dell'Android Software Development Kit.
Per connettere il dispositivo via TCP/IP, prima di tutto è necessario determinare l'indirizzo IP del telefono ed essere sicuri che il dispositivo sia già connesso attraverso la porta USB. Una volta trovato l'indirizzo è semplicemente necessario digitare i seguenti comandi sul terminale:

```
adb tcpip <port>
adb connect <ipaddress>:<port>
```

A quel punto è possibile scollegare il dispositivo dalla porta USB per continuare ad utilizzarlo attraverso la rete. Per fermare l'esecuzione invece è necessario utilizzare il comando:

```
adb disconnect
```

### 5.3.2 Logcat

Una volta che il dispositivo è capace di scambiare informazioni con il computer il secondo passo diventa quello di registrare i LOG durante l'esecuzione. Per farlo l'Android SDK viene in aiuto fornendo l'utility di logging chiamata *Logcat*, che è essenziale per determinare cosa sta facendo un applicativo in esecuzione.
Per utilizzarlo basta digitare sul terminale:

```
adb logcat
```

Con il dispositivo mobile connesso e riconosciuto, i log cominciano immediatamente a stamparsi sul terminale. Il problema di questo approccio è che

la quantità di informazioni stampate è molto alta, per questo logcat dà la possibilità di inserire dei filtri, attraverso dei <tag>, che regolano gli output da stampare, come nel caso di questo progetto di tesi:

```
adb logcat -s Unity
```

**Logcat su file per determinare crash**

Logcat non deve per forza essere attivo mentre un'applicazione che si sta testando crasha. Fortunatamente, è capace di tenere in memoria un buffer di output recenti, e in molti casi attraverso un comando è possibile poi chiedere al tool di farsi restituire il log attraverso:

```
adb logcat > crash.log
```

In questo modo, una volta avuto il file basta cercare il momento del crash all'interno per tentare di risalire al problema. In alternativa è anche possibile dire a logcat di scrivere fin da subito l'intero output su un file per ogni esecuzione del programma. Il comando utilizzato più spesso per questo applicativo infatti è stato:

```
adb logcat -s Unity:D > E:/AndroidLog/android_log.log
```

Che è l'unione di tutto ciò che è stato spiegato in questo paragrafo.

## 5.4 Ottimizzazione della memoria e dei tempi di caricamento

Al completamento delle funzionalità presenti nel prototipo, il lavoro mostrava numerosi problemi legati alla velocità di caricamento delle scene e alla presenza di numerosi lag che rendevano l'applicazione poco appetibile. Questi problemi erano dovuti a due cause principali, la prima legata al fatto che le immagini da scaricare per la visualizzazione di ogni singolo panorama erano 6 e tutte alquanto pesanti, il che portava a tempi lunghi di download prima della effettiva visualizzazione. La seconda problematica invece era dovuta al fatto che nonostante il Galaxy Note 4 sia un dispositivo ad alte prestazioni, il codice scritto con Unity presenta un'architettura *single thread*

che non permette l'uso contemporaneo di funzioni. Questo portava a dei forti lag nel momento in cui si cercava di cambiare lo Skybox sostituendo le 6 immagini cubiche una per volta.
In questo paragrafo verrà quindi illustrato come si è cercato di risolvere questi problemi analizzando prima le possibili soluzioni e descrivendo infine quella utilizzata nel prototipo. Prima di procedere tuttavia è importante descrivere alcuni strumenti utilizzati all'interno del codice e che il framework di Unity fornisce:

**Coroutine**

Quando viene chiamata una funzione, questa viene totalmente eseguita prima di ritornare. Ciò significa che ogni azione presente al suo interno avviene in un singolo frame e quindi non può essere usata per contenere una animazione procedurale o una sequenza di eventi spalmata nel tempo. Una Coroutine invece è simile ad una funzione che però ha la capacità di mettere in pausa la sua esecuzione dando di nuovo controllo al Main thread di Unity, per poi riprendere l'esecuzione in un frame successivo da dove si era interrotta. Viene dichiarata in questo modo:

```
IEnumerator WaitAndPrint()
{
    //suspend execution for 5 seconds
    yield return new WaitForSeconds(5);
    print("WaitAndPrint " + Time.time);
}
```

Dove *IEnumerator* è una interfaccia obbligatoria per la dichiarazione di una Coroutine, mentre *yield* è il comando chiave che permette di fermare l'esecuzione della Coroutine per poi riprenderla esattamente da quel punto.
Questa struttura è stata molto di aiuto durante lo sviluppo perchè ha permesso di recuperare più volte l'esecuzione di una funzione dal punto in cui era necessario. Un esempio è legato alla ripresa della sostituzione delle immagini per la Skybox una volta che queste avessero completato il download.

**WWW**

Classe per l'accesso alle pagine web che permette di ritornare il contenuto di un URL, è l'unico componente che può lavorare in maniera asincrona all'interno di Unity. Con l'aiuto del comando *yield* precedentemente spiegato, è possibile attendere il completamento di un download e ripartire da esso senza bloccare il resto dell'applicazione.

### 5.4.1 Flusso logico del caricamento di una scena

Al fine di descrivere le ottimizzazioni apportate è importante illustrare come fossero stati scritti in precedenza i flussi di chiamata per il caricamento di una scena. In questo modo sarà poi più facile capire le differenze con le modifiche aggiunte successivamente.
Una volta arrivati su un tour virtuale le fasi per la preparazione della scena, e di tutte le altre a lei collegate, erano le seguenti:

- Lo *SceneManager.cs* cominciava la sua fase di Start() avviando una Coroutine per il ritorno delle informazioni JSON legate al singolo tour virtuale contenente tutte le scene. Ricevuti i dati venivano presi i link alle 6 immagini cubiche e la definizione dei vari Hotspot presenti nella *firstScene*.

- Da quel momento partiva inizialmente la *ImportSkyboxImage.cs* per la costruzione delle immagini. Al suo interno veniva avviata un'altra Coroutine dove in sequenza venivano scaricate le 6 immagini e sostituite nello skybox. Questo era un punto critico in quanto l'utente che osserva la scena mentre tutto questo accadeva, notava grossi lag derivanti dalla funzione di Unity *LoadImageIntoTexture()* che è davvero pesante e lenta nella sua esecuzione e nell'utilizzo della memoria RAM. Inoltre avviando il download delle immagini proprio nel momento di cambiare ambiente, si avevano dei tempi molto lunghi per la sostituzione della scena che potevano superare i 12 secondi.

- Ultima fase quella legata al posizionamento degli hotspot gestita dalla *ImportSpheres.cs*. In questo script, come descritto in precedenza vengo-

no decodificate tutte le posizioni degli hotspot ed inseriti con caratteristiche diverse in base alla loro tipologia. Il tutto è racchiuso all'interno di una Coroutine anche i questo caso.

- In caso di attivazione di un Hotspot di tipo Scene per il cambio di un ambiente, il file *HotspotScene.cs* passava allo *SceneManager.cs* l'_id della scena da cui prendere i 6 link alle immagini cubiche e le definizioni dei nuovi hotspot. A quel punto si riprende normalmente dal punto due.

### 5.4.2 Perfezionamenti

**Lag nella scena**

Per risolvere il problema legato ai lag durante il cambiamento di una scena, si sono prese in considerazione varie soluzioni. Attraverso test di prestazioni e ricerca online, si sono identificati due *bottleneck* critici: uno nella funzione LoadImageIntoTexture() che risultava davvero lenta, compromettendo l'esecuzione fluida delle altre operazioni da eseguire essendo Unity una architettura single thread, e l'altro legato al massiccio utilizzo di oggetti Texture, elementi davvero pesanti per la memoria RAM e che non venivano automaticamente eliminati dal Garbage Collector.

Un primo approccio è stato quello di capire se si potesse superare il vincolo delle API non asincrone di Unity utilizzandone altre, ma purtroppo non esiste ancora nulla a riguardo. Vi sono infatti numerosi framework che rendono parallelizzabile l'utilizzo di conti e semplici routine all'interno dell'IDE, tuttavia con il loro utilizzo non è comunque possibile adoperare le funzioni sugli oggetti dell'SDK perchè non thread safe. Per questo motivo non essendoci una alternativa esterna alla funzione LoadImageIntoTexture(), che è fondamentale ai fini del progetto, non è stato possibile implementare alcun tipo di parallelizzazione utile.
Dovendo utilizzare quindi la funzione LoadImageIntoTexture() in maniera sincrona, l'unica possibilità risiedeva nel cercare di farla durare il meno possibile. Per fare questo era necessario ristrutturare la Coroutine inclusa nel file ImportSkyboxImage.cs separando la fase dei download delle immagini dall'utilizzo della LoadImageIntoTexture() in cui queste venivano effettiva-

mente sostituite nella Skybox. Con le due fasi separate, una volta scaricate le immagini, la transizione di scena con l'utilizzo della funzione critica, durava solo 2 secondi. In questo modo i lag in quel tempo non erano più particolarmente percepibili data la loro poca durata.

Per il secondo problema legato al troppo utilizzo di oggetti Texture invece è bastato gestire la memoria manualmente eliminando tutti gli oggetti di quella tipologia non appena non ve ne fosse stato più bisogno. Questa ha eliminato completamente i lag relativi alle scene esplorate successivamente. Succedeva infatti che lasciando in memoria le texture delle scene passate, a mano a mano che si procedeva nel tour iniziavano a verificarsi sempre più lag anche durante l'esplorazione e non solo nel momento delle transizioni. Persino aprire un hotspot di tipo Info diventava difficoltoso.

#### Riduzione tempi di esplorazione

Nonostante la riduzione dei lag durante l'esplorazione, restava il fatto che passare da una scena ad un'altra attraverso gli hotspot risultasse davvero lento a causa dell'alto tempo di download di 6 diverse immagini. Il tempo medio di transizione infatti era di circa 13 secondi di cui 10 secondi legati al download delle immagini cubiche necessarie e 3 relative alla sostituzione della Skybox con la funzione LoadImageIntoTexture().
Anche i questo caso non era possibile trovare una soluzione valida relativa alla diminuzione dei tempi di download, ma allo stesso tempo si potevano apportare alcuni accorgimenti per una migliore visualizzazione:

Una volta selezionato un tour virtuale, invece di mostrare la *firstScene* appena finito il suo caricamento, si è pensato di continuare ad attendere fino al completo download delle immagini subito successive e collegate ad essa. In questo modo un utente avrebbe potuto subito attuare una transizione di scena abbattendo i tempi di download e cambiando ambiente in soli 3 secondi, ovvero quelli necessari ad un cambio scena con la LoadImageIntoTexture(). Il caricamento iniziale della prima *firstScene* è diventato così di circa 10 secondi più lungo per ogni scena collegata direttamente a lei costringendo l'utente ad aspettare all'inizio. In questo modo tuttavia si offre una migliore navigazione una volta avviata l'esperienza esplorativa che non

sarà più lenta come in precedenza.
Oltre al download delle scene subito connesse alla *firstScene* infatti, è stata implementata anche una coda di priorità nella quale una volta che l'utente arriva in una scena, in background vengono scaricate tutte le immagini cubiche degli ambienti vicini e una volta finito, si comincia con le scene ancora più in profondità. In questo modo come un utente arriva in un ambiente subito l'applicativo si porta avanti scaricando quello che c'è dopo abbattendo i tempi di transizione. Naturalmente le immagini già scaricate non vengono eliminate ma vengono salvate in una hashmap formata da chiave uguale all'_id della scena e per valore la lista delle 6 immagini.
Per evitare di occupare troppa memoria, in realtà non vengono salvate le immagini vere e proprie ma direttamente gli oggetti WWW di Unity che al loro interno contengono anche le immagini in una versione compressa e poco pesante. Questa è sembrata una soluzione migliore rispetto a quella di salvare le immagini vere e proprie all'interno del dispositivo appesantendo così le varie operazioni (in quanto il salvataggio risulta lento) ed il dispositivo stesso.

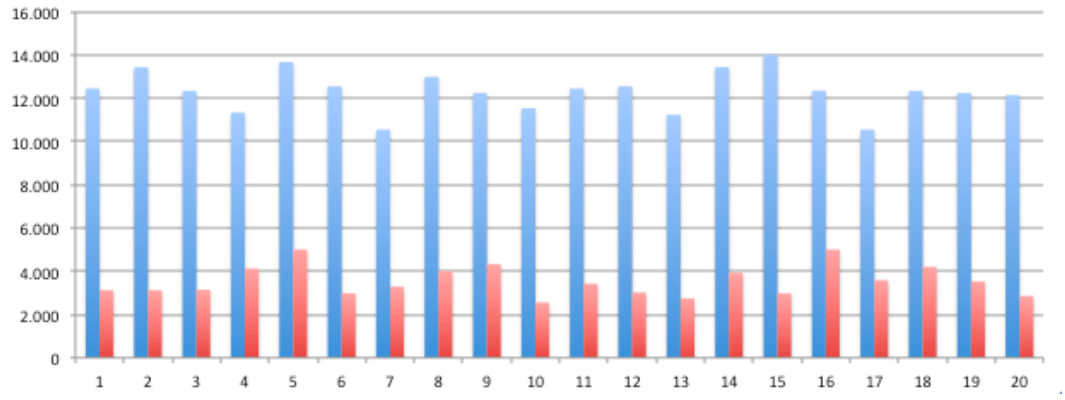


*Figura 5.7: Riduzione tempi di transizione tra una scena ed un'altra dopo le ottimizzazioni sul codice*

## 5.5 Conclusioni

In questo capitolo è stato illustrato nel dettaglio il funzionamento del client immersivo sviluppato per Samsung Gear VR, inoltre si sono mostrate le varie fasi dello sviluppo e la risoluzione dei problemi relativi alle prestazioni e all'incapacità di poter debbuggare semplicemente. Il risultato dello sviluppo ha portato alla creazione di un applicativo capace di visualizzare i tour presenti nella piattaforma attraverso un dispositivo molto innovativo e differente dal solito. Attraverso il sistema RESTful costruito in precedenza questa applicazione è capace di interfacciarsi con la piattaforma e ottenere le informazioni necessarie all'intera esplorazione del tour virtuale. Questi sono stati due punti fondamentali di questo lavoro in quanto al momento un applicativo per VR di questo genere capace di collegarsi ad una piattaforma esterna, e di interagire per il download di dati invece di prenderli direttamente dalla sua memoria, è qualcosa di non ancora presente ad oggi nei competitors visionati.
Nel prossimo capitolo verranno proposti alcuni casi d'uso riassuntivi che mostrano le potenzialità e le funzioni realizzate con questo lavoro di tesi.

# Capitolo 6

# Utilizzo delle interfacce client

Nei capitoli precedenti sono stati descritti tutti gli aspetti tecnici che riguardavano lo sviluppo della piattaforma client-server e dell'applicazione nativa per il Samsung Gear VR. Sono stati inoltre spiegati i modi in cui queste applicazioni tanto diverse riuscissero ad interfacciarsi tra loro e le soluzioni ai problemi riscontrati durante lo sviluppo.
Per dare adesso una visione generale di quanto prodotto con questo lavoro di tesi, nel capitolo verranno mostrati alcuni esempi significativi di utilizzo delle interfacce sviluppate, accompagnate da descrizioni esplicative.

## 6.1 Costruzione di un tour virtuale

Il primo esempio d'uso riguarda la costruzione di un tour virtuale per un utente registrato sulla piattaforma e che ha già effettuato l'operazione di login.

La prima pagina che viene presentata dopo l'accesso è la Home della piattaforma. Questa è formata da una mappa a tutto schermo centrata sulla posizione geografica del dispositivo da cui si sta visualizzando l'applicazione web. Viene richiesta infatti l'autorizzazione alle informazioni di localizzazione prima del caricamento della pagina. In caso di rifiuto verranno caricate la coordinate del centro di Milano, città in cui è nata questa piattaforma.
Dalla Home è possibile visualizzare i *segnaposti* relativi alla posizione dei

tour virtuali vicini all'utente che possono essere esplorati. Per la creazione di un nuovo tour tuttavia è necessario andare in alto a destra e cliccare sul proprio nome utente. In questo modo verrà visualizzato un menù a tendina e, cliccando su *Admin Panel*, si potrà accedere al costruttore di tour virtuali.

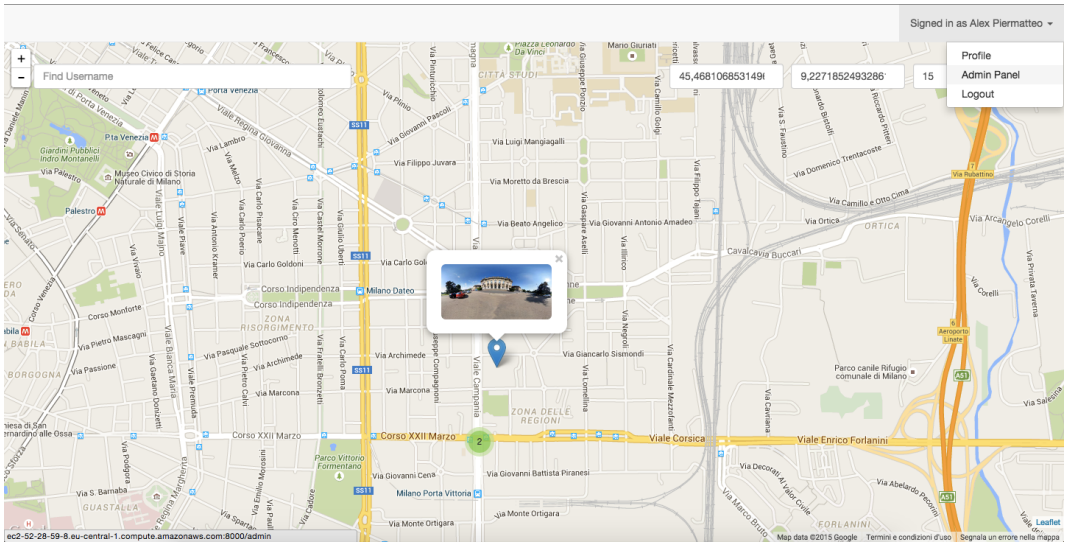


*Figura 6.1: Home della piattaforma web*

Arrivati sulla pagina Admin Panel vi si trova di fronte al vero e proprio costruttore di tour virtuali. In questa interfaccia sono presenti sulla destra tutte le immagini caricate dall'utente e che possono essere utilizzate per la creazione delle scene. Selezionata una delle foto equirettangolari disponibili, si visualizza al centro il dettaglio dell'immagine con la presenza di alcune funzionalità necessarie alla costruzione di una scena.
Per aggiungere la foto selezionata ad un tour virtuale rendendola effettivamente una scena, è necessario spuntare la checkbox *Add this image in the Virtual Tour*. A quel punto si sbloccheranno le funzionalità per l'aggiunta di hotspot per il cambio scena o per quelli di informazione.

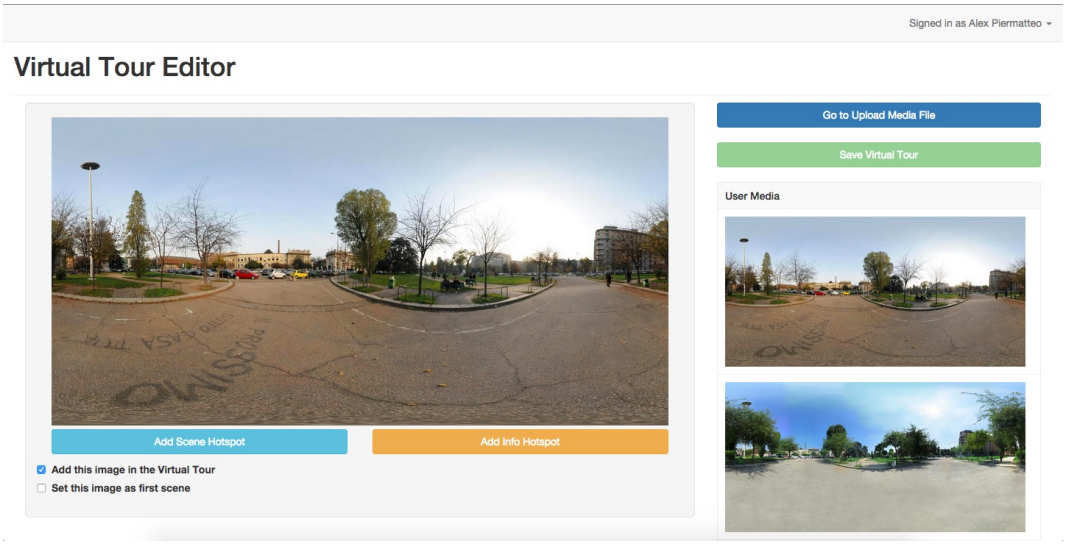


*Figura 6.2: Generatore Tour Virtuali*

Gli hotspot vengono aggiunti nella posizione desiderata premendo prima sul pulsante *Add Scene/Info Hotspot* e successivamente cliccando il punto desiderato sulla immagine posta poco più in alto. In quel modo verranno aggiunte in automatico le coordinate relative al punto selezionato sulla foto.

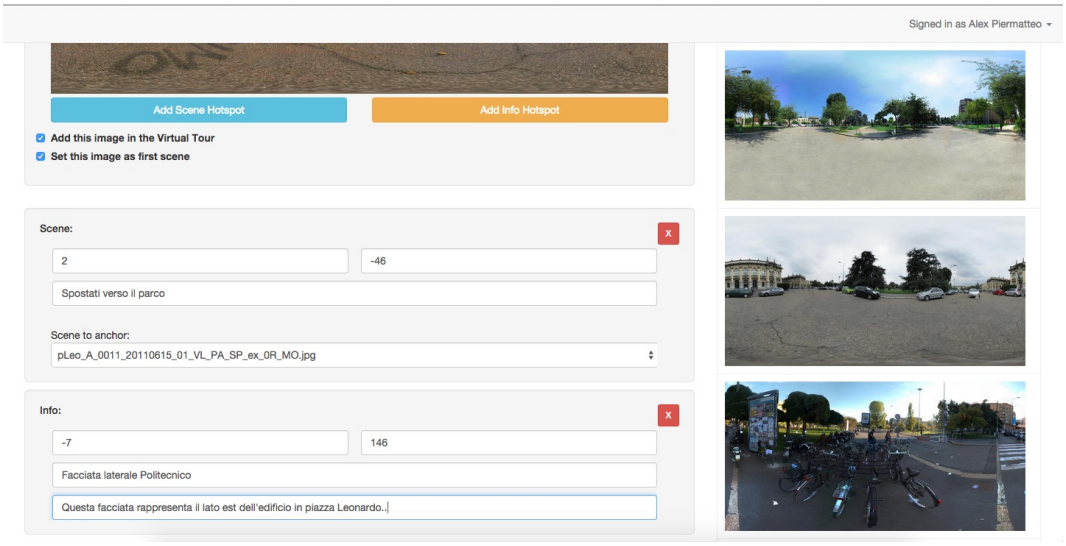


*Figura 6.3: Focus sugli hotspot aggiunti nel generatore di Tour Virtuali*

Oltre le informazioni sulla posizione, per entrambe le tipologie di Hotspot è possibile inserire un testo come Label e, nel caso di un hotspot 'Info' anche

una descrizione. Nel caso dell'hotspot di tipo 'Scene' invece sarà possibile scegliere da un menù a tendina a quale altra immagine disponibile collegare la scena corrente. Naturalmente nella lista vi sono solo i collegamenti alle immagini che sono state scelte per far parte del tour virtuale.
Una volta cliccato sul bottone *Save Virtual Tour*, a completamento della procedura, verrà visualizzato un messaggio di corretta operazione e sarà possibile costruire subito un altro tour virtuale.

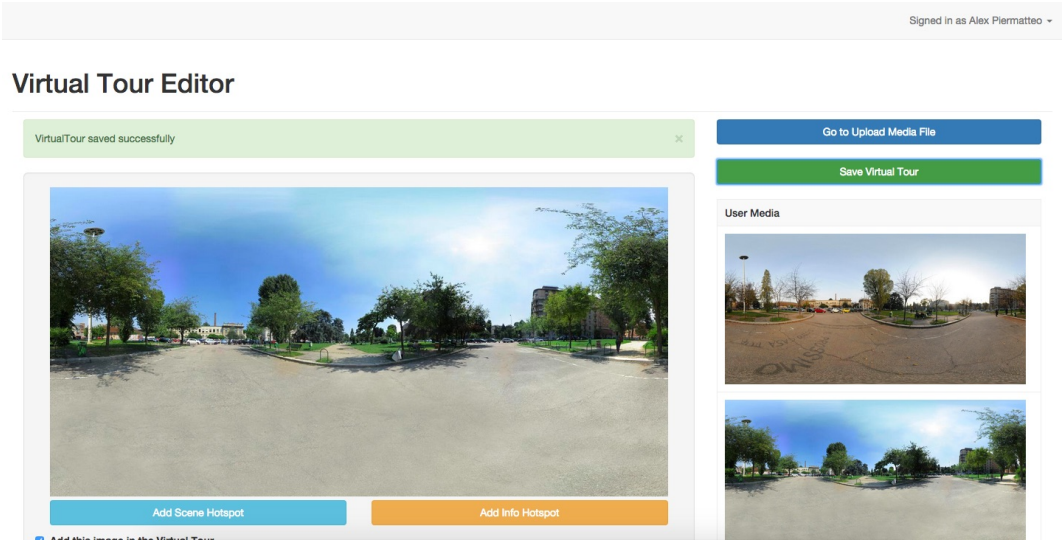


*Figura 6.4: Successo nella operazione di salvataggio di un tour virtuale*

Nel caso non siano presenti immagini sulla lista di destra per la costruzione di un tour, dal pulsante *Go to upload media file* è possibile raggiungere la pagina per il caricamento di una nuova foto equirettangolare. In questa interfaccia di file uploading è possibile caricare file di tipo JPEG o PNG di immagini panoramiche. Ogni campo risulta obbligatorio ed una volta inserita la Nazione, la Città, la Zona, le coordinate geografiche, e la data di riferimento del file multimediale, basterà premere il tasto *Upload* per far partire una barra di avanzamento che misura lo stato di completamento del processo.

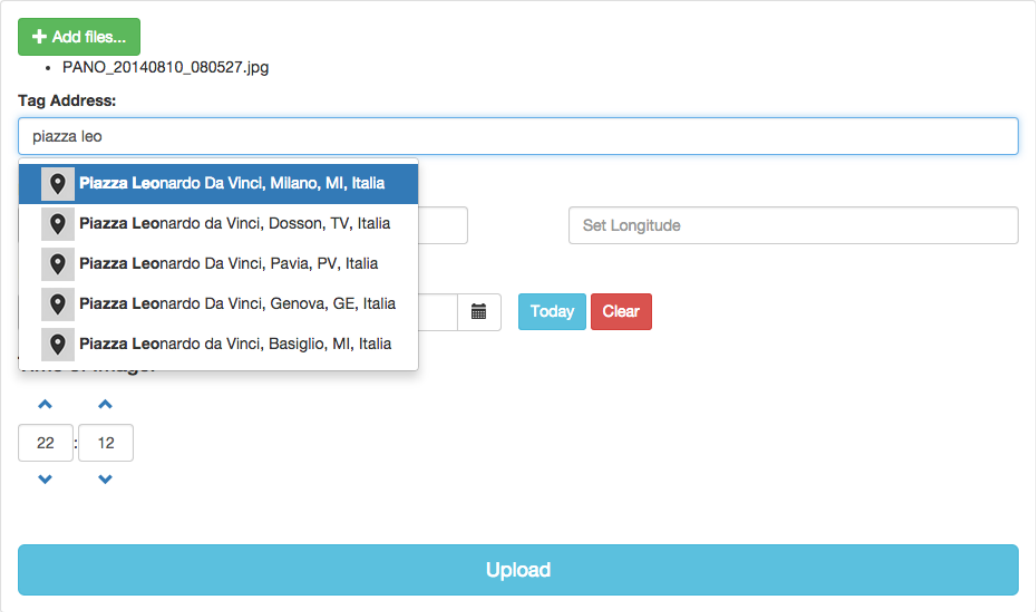


*Figura 6.5: File Uploader per il caricamento di una foto equirettangolare*

## 6.2 Visualizzazione di un tour, client web

Mostrato il caso d'uso relativo alla costruzione di un tour virtuale, è logico passare subito alla visualizzazione di uno di questi, rimanendo tuttavia sempre sulla piattaforma web.
Partendo dalla mappa nella *Home* descritta nel paragrafo precedente, è possibile notare come sia comparso un *segnaposto* nella zona centrale di Piazza Leonardo da Vinci, Milano. Questo è il tour appena costruito attraverso la descrizione del caso d'uso precedente.

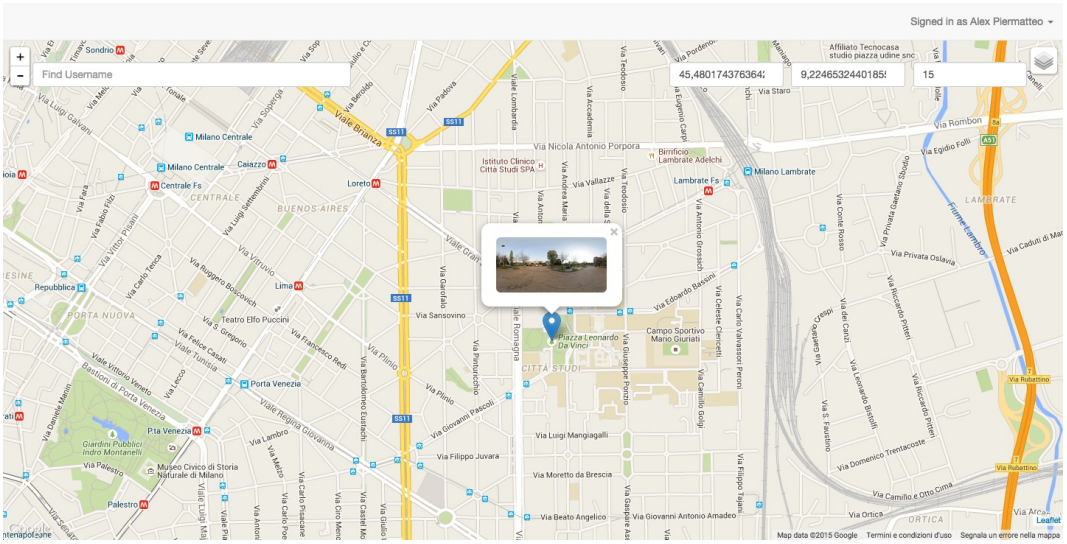


*Figura 6.6: Presenza di un nuovo segnaposto in Home dopo la creazione di un nuovo tour virtuale*

Selezionando l'immagine sopra il *segnaposto*, si viene indirizzati verso la pagina di visualizzazione del tour virtuale. In questa interfaccia sono presenti in alto a destra le informazioni relative ad ogni scena come ad esempio: nome utente di chi ha caricato l'immagine, data, ora e luogo di riferimento della scena ed infine una mini-mappa che mostra il punto esatto della geolocalizzazione della foto panoramica.
Sul lato opposto vi è invece uno strumento per lo zoom avanti e indietro del panorama.

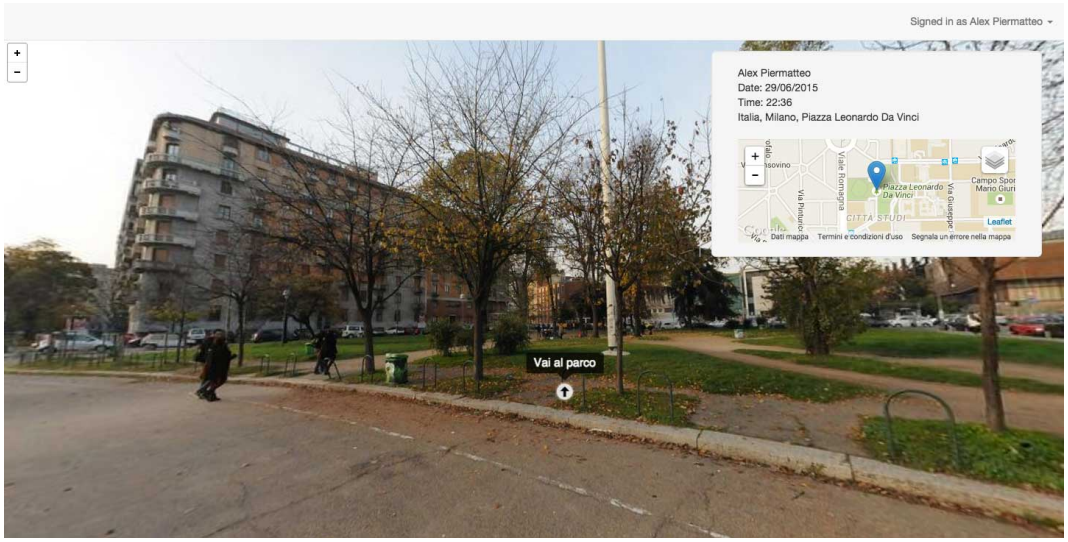


*Figura 6.7: FirstScene del tour virtuale con hotspot di tipo 'Scene'*

E' possibile notare come vi sia, all'interno dell'ambiente, la presenza di un hotspot di tipo *scene* che permette la transizione sull'altra scena del tour virtuale precedentemente creata. Se vi si clicca sopra, l'intero ambiente viene modificato, cambiano tutte le informazioni e vi si ritrova sulla seconda scena.


*Figura 6.8: Seconda scena del tour virtuale*

Alternativamente alla transizione nella seconda scena, si può continuare l'esplorazione della prima ambientazione e notare la presenza di un altro

hotspot di tipo *info*. Cliccando su di esso compare sulla sinistra dello schermo il testo descrittivo relativo a quell'elemento, che è possibile far sparire selezionando nuovamente l'hotspot in questione.

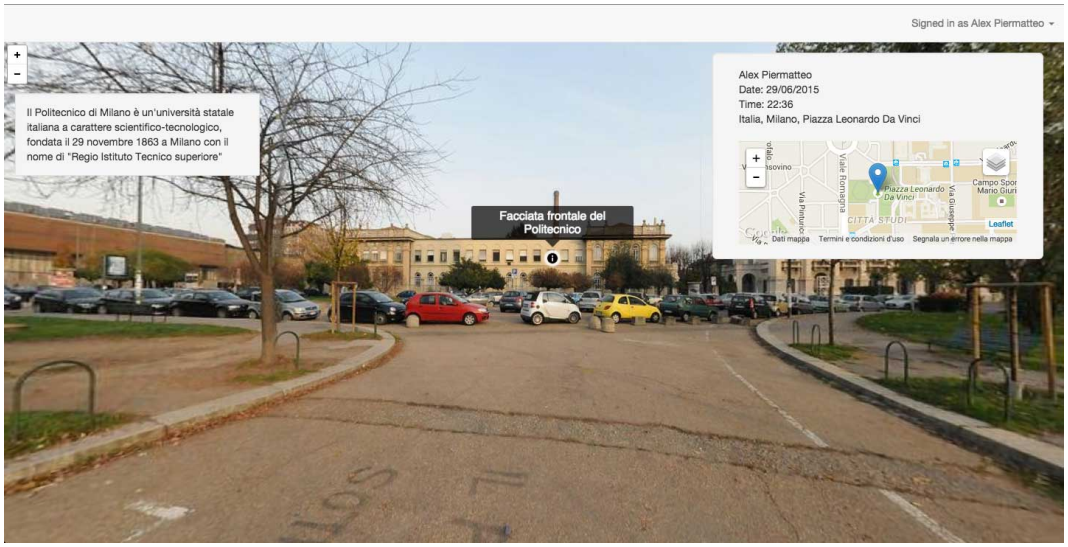


*Figura 6.9: FirstScene del tour virtuale con hotspot di tipo 'Info'*

## 6.3 Visualizzazione di un tour, client Gear VR

Concluso con i casi d'uso sulla piattaforma web, viene da sé mostrare la visualizzazione che si può ottenere attraverso l'applicativo sviluppato per il Samsung Gear VR. Verrà preso in esame lo stesso tour virtuale mostrato nei paragrafi precedenti in modo tale da avere un confronto il più paritario possibile.
All'avvio dell'applicazione, come succede per il client web, viene attuata una geo-localizzazione del dispositivo in modo tale da restituire i soli tour virtuali ad una distanza ben definita (10 km) dalla posizione dell'utente.
La scena principale di selezione si compone di uno sfondo a 360 gradi predefinito caricato internamente dall'applicativo e poi da tutte le anteprime dei tour virtuali poste circolarmente intorno alla posizione dell'utente che si ritrova al centro. Da questa posizione si possono scorrere tutti i tour disponibili e selezionare attraverso il puntatore quello che si preferisce visualizzare.

Le anteprime, inizialmente in ombra, al passaggio del mirino subiscono un ingrandimento ed una maggiore luminosità.

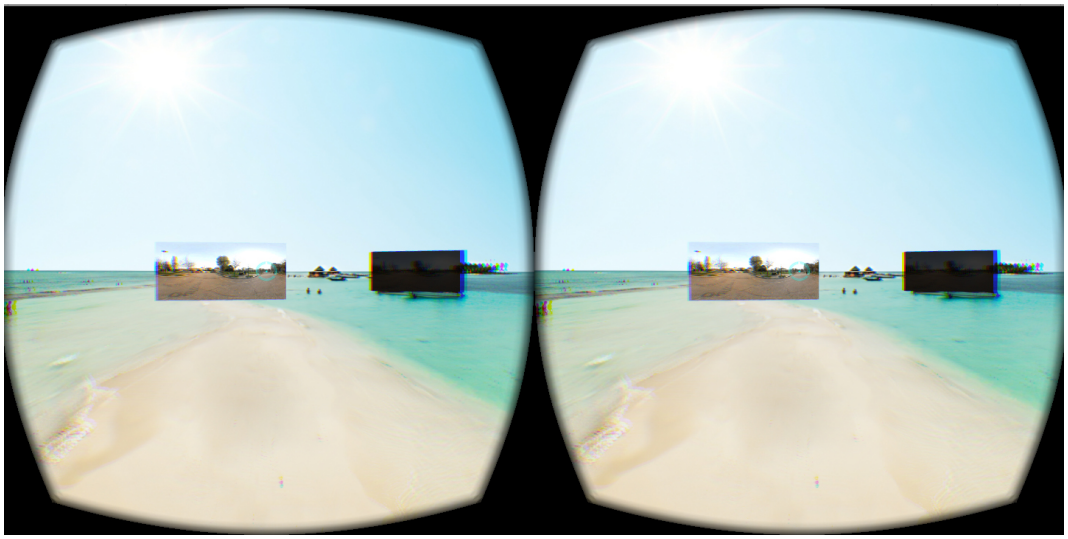

*Figura 6.10: Scena principale dell'applicativo per Gear VR, da qui vengono selezionati i tour virtuali da visualizzare*

Una volta selezionata una anteprima da visualizzare e cliccato sul touchpad laterale del dispositivo si viene catapultati in una schermata di caricamento per un tempo di circa 20 secondi. Questa tempistica è indicativa e può variare in base alla complessità del tour che si vuole esplorare.
Completato il caricamento l'utente si ritrova nella *FirstScene* del tour virtuale in Piazza Leonardo da Vinci, Milano. All'interno vi sono gli stessi Hotspot visti nel paragrafo precedente e si può notare come anche la loro posizione nell'ambiente sia la stessa grazie alla trasformazione illustrata nella sezione 5.2.2.

Nell'applicativo per il Samsung Gear VR gli hotspot di tipo *scene* sono colorati di verde quando non sono puntati dal mirino e rossi quando invece lo sono. Se si decide di puntarne uno per 3 secondi consecutivi si viene trasportati nella seconda scena di riferimento.

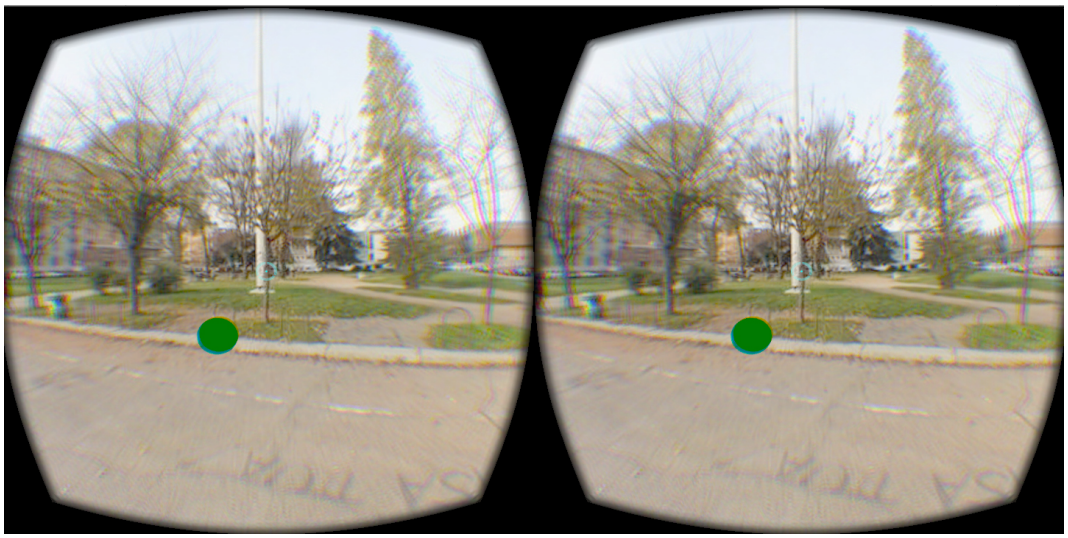

*Figura 6.11: FirstScene del tour virtuale per Gear VR, con hotspot di tipo 'Scene'*

Oltre agli hotspot di tipo *scene,* anche in questo caso c'è la possibilità di esplorare la scena e visualizzare invece gli hotspot di tipo *info*. I colori di quest'altra tipologia sono blu quando non vengono puntati dal mirino e viola quando invece lo sono.

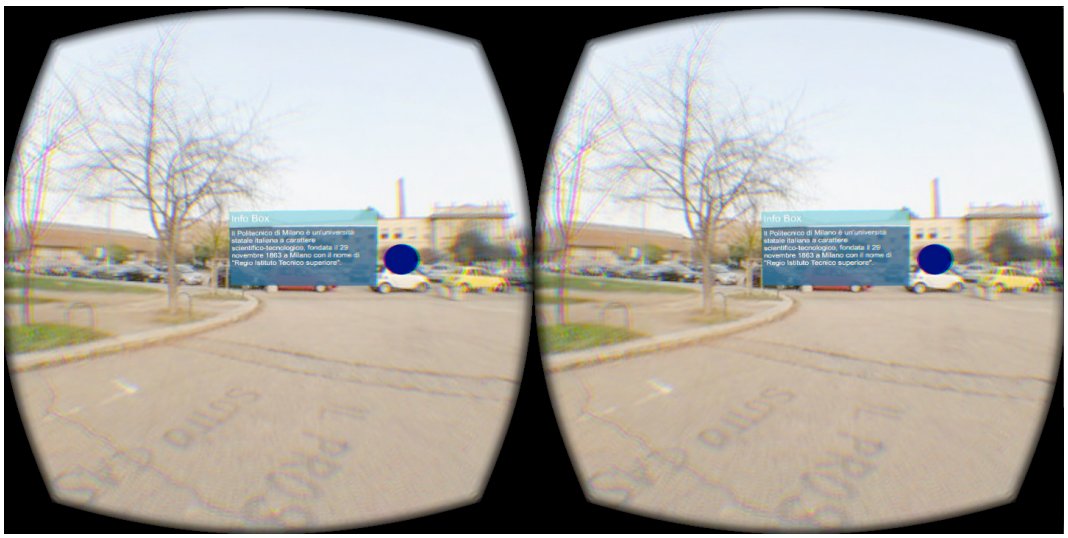


*Figura 6.12: FirstScene del tour virtuale per Gear VR, con hotspot di tipo 'Info'*

La descrizione si apre di fianco all'elemento non appena si punta l'oggetto per 3 secondi esattamente come avviene per la transizione di una scena. Il pannello che contiene il testo descrittivo si chiama Canvas ed è di colore

azzurro. E' possibile chiuderlo e farlo sparire puntando nuovamente per 3 secondi l'hotspot di tipo *info*.

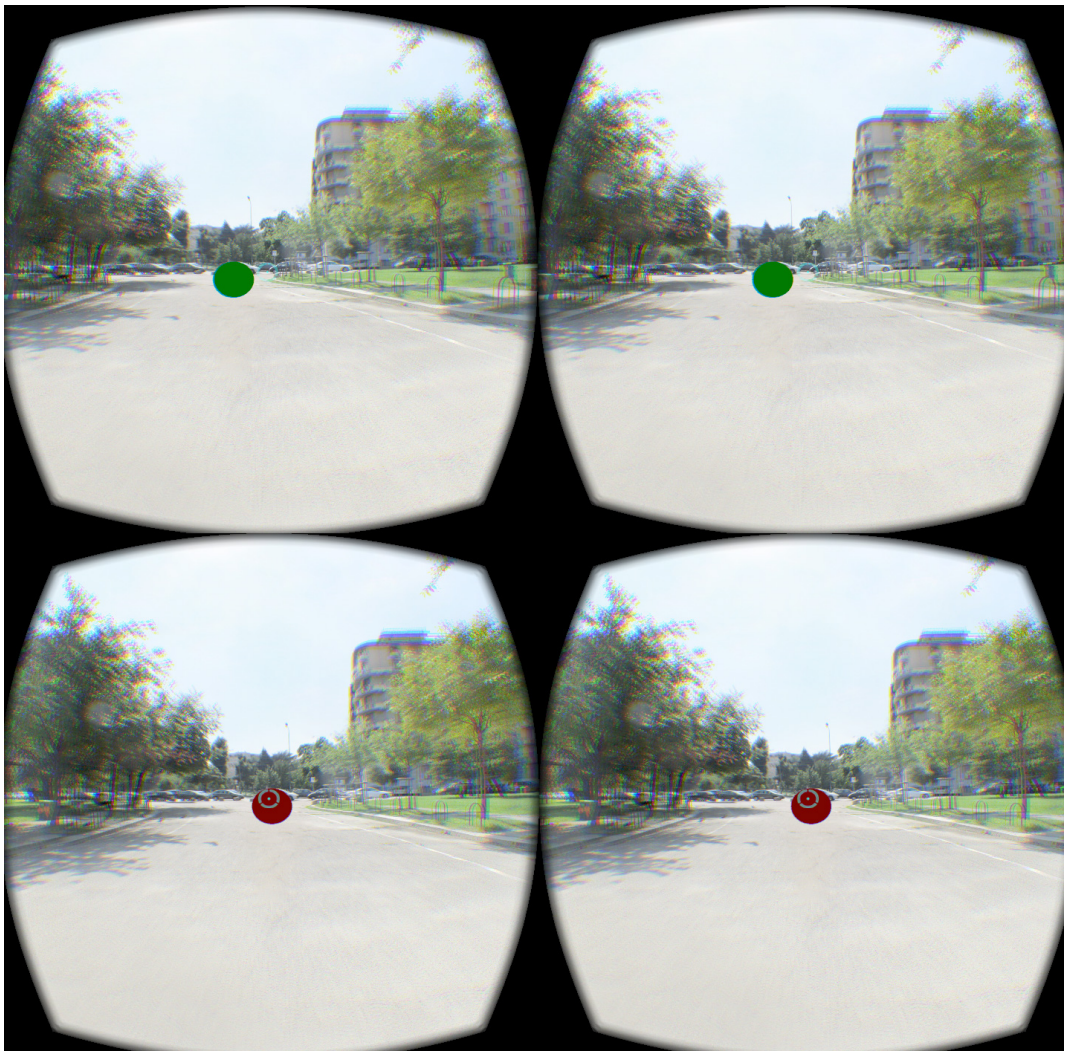

*Figura 6.13: Differenza tra hotspot inizialmente non puntato dal mirino e successivamente selezionato*

Si conclude così questo capitolo relativo ai casi d'uso della piattaforma e dei due client sviluppati per questo lavoro di tesi. Vi sarebbero altre parti minori, come la cancellazione o la modifica di un tour virtuale, che potrebbero essere mostrate ma che sono state considerate secondarie rispetto alle grosse funzionalità descritte.

# Capitolo 7

# Conclusioni e sviluppi futuri

## 7.1 Valutazione qualitativa e conclusioni

Arrivati alle conclusioni di questo documento è d'obbligo un piccolo confronto tra lo stato dell'arte, di cui si era parlato nel Capitolo 2, con ciò che si è effettivamente svolto con questo lavoro di tesi.
In linea generale questo progetto ha riguardato lo sviluppo di uno strumento di supporto per quelle persone comuni che sono interessate alla creazione e alla scoperta di tour virtuali attraverso una piattaforma che renda semplice tutto questo. Indirettamente inoltre, grazie alle API RESTful utilizzate per la sua realizzazione, può anche offrire ad eventuali sviluppatori esterni, in maniera gratuita, la possibilità di creare un proprio client per la costruzione di tour virtuali sfruttando la logica sul server di questa piattaforma proposta.
La parte di fondamentale importanza riguarda tuttavia l'interazione di questa piattaforma con un dispositivo totalmente immersivo come il Samsung Gear VR che usufruisce delle informazioni presenti in essa attraverso la chiamata delle API sviluppate e non attraverso l'utilizzo di dati già interni al dispositivo come accade nelle applicazioni immersive concorrenti.

La tesi come si è visto dai capitoli precedenti è stata divisa in due parti: la prima legata allo sviluppo dell'applicazione web formata da una parte server ed una client totalmente separate, e che presenta diverse funzionalità in più, rispetto al parco delle piattaforme concorrenti, tra le quali:

- In primis la possibilità di costruire tour molto velocemente senza dover aspettare l'approvazione dei panorami, e in totale autonomia attraverso il tool online. Tool intuitivo, rapido e fruibile senza il bisogno della minima conoscenza di linguaggi di formattazione (vedi XML per krPano).

- La possibilità di vedere hostato il proprio lavoro direttamente nella piattaforma invece che ricevere del solo codice da inserire nel proprio sito web.

- La presenza di un taglio più social legato alla possibilità di condivisione o di follow/unfollow di altri utenti interni alla piattaforma.

- La possibilità per gli sviluppatori di utilizzare le API fornite per la realizzazione di altri client di costruzione di tour virtuali.

La seconda parte di questo lavoro invece , è stata legata allo sviluppo tramite Unity di una applicazione compatibile per il dispositivo immersivo Gear VR. Questo secondo sviluppo non si pone solo come semplice costruzione di un client alternativo al primo, ma tenta di portare innovazione e di fornire uno studio su quanto sia possibile uno scambio di dati, anche pesanti, con un server in relazione al mantenimento di una buona usabilità dell'applicazione. In questo senso si sono ottenuti buoni risultati di velocità di download e di visualizzazione dell'intero tour, seppur con la limitazione di dover scaricare una buona parte di informazioni prima di cominciare l'esperienza immersiva. Questa volontà di lavorare con un servizo esterno pone l'applicativo più avanti rispetto alla concorrenza che utilizza sempre dispositivi immersivi per le proprie scene a 360 gradi.

In definitiva, questi due progetti, hanno costituito il lavoro di tesi che si è posto la sfida di generare una piattaforma web con funzionalità non presenti nel parco delle applicazioni del suo genere e che soprattutto si è posto l'obiettivo di integrare client molto differenti come quello web e quello immersivo utilizzando lo stesso modello di API e dati all'interno del servizio RESTful, riuscendoci.

## 7.2 Sviluppi futuri

Il progetto proposto tuttavia è davvero molto giovane: attualmente il lavoro si è concentrato soprattutto nella creazione di un prototipo funzionante sia per quanto riguarda la parte server e sia per quanto concerne i due client differenti (web e immersivo). Particolare attenzione è stata posta sulla buona struttura dell'architettura generale, sulla bontà delle API fornite e sull'efficace interazione tra il Gear VR ed il servizio. Questo ultimo argomento è molto delicato in quanto la scalabilità delle informazioni attraverso l'utilizzo di chiamate server è la strada giusta da percorrere per sfruttare a pieno le capacità di questi dispositivi immersivi che al momento invece preferiscono sfruttare elementi interni all'applicazione per una maggiore reattività. Per queste ragioni, gli sforzi si sono concentrati soprattutto nel riuscire a fornire una interazione che permettesse una navigazione di buon livello senza però subire pesanti rallentamenti derivanti dallo sfruttamento di servizi esterni. Oltre questo però sono state implementate funzionalità utili come la creazione di tour direttamente in piattaforma che rendono il prototipo ad un livello leggermente superiore agli altri, anche solo per il fatto che chi fornisce servizi simili non lo fa in maniera gratuita. Vi sono tuttavia differenti aspetti migliorabili e che possono riguardare sviluppi futuri in questo senso:

- Poichè la struttura del modello Media creato sul database supporta varie tipologie di multimedia oltre alle immagini, sarebbe interessante estendere la compatibilità della piattaforma e dei client con altri tipi di file come ad esempio audio. In questo modo sarebbe possibile aggiungere ad ogni scena dei suoni ambientali a 360 gradi coinvolgenti, utili soprattutto per l'esperienza immersiva con il Gear VR che arriverebbe a diventare ancora più completa.

- Allo stesso modo il modello dei Media supporta il salvataggio delle date temporali in cui le immagini sono riferite. Sarebbe interessante poter costruire un servizio di timeline in cui mostrare per ogni scena il suo passato, il suo presente o il suo futuro così da fornire all'utente forme di visualizzazioni alternative.

- Per entrambe le piattaforme un'altra aggiunta funzionale potrebbe essere quella di fornire hotspot di tipo differente oltre a quelli di transizione o informazione. Ad esempio un hotspot che fornisca la vista di immagini all'interno dell'ambiente immersivo.

- In ultima la possibilità di estendere l'utilizzo della piattaforma a dispositivi immersivi differenti oltre il Gear VR, magari prendendo in considerazione visori di fascia più bassa come i Google Cardboard. Questo sarebbe possibile in quanto considerata l'attuale applicazione in Unity, con alcune modifiche potrebbe diventare facilmente compatibile.

# Bibliografia


[1] Vladimir Agafonkin. Leafletjs api. http://leafletjs.com/reference.html.

[2] AngularJS. Angularjs api. https://docs.angularjs.org/api.

[3] Bootstrap. Bootstrap ui. http://angular-ui.github.io/bootstrap/.

[4] ExpressJS. Expressjs api. http://expressjs.com/4x/api.html.

[5] Ciul Github repo. An angularjs module to take approach of social javascript sdk. https://github.com/Ciul/angular-social.

[6] GridFS. Gridfs manual. http://docs.mongodb.org/manual/core/gridfs/.

[7] Google Inc. Adb download. http://developer.android.com/tools/help/adb.html.

[8] Google Inc. Android sdk download. https://developer.android.com/sdk/index.html.

[9] Google Inc. It is your turn to make it. https://www.google.com/get/cardboard/manufacturers/.

[10] krPano. krpano doc. http://krpano.com/docu/.

[11] Sam Ruby Leonard Richardson. *RESTful Web Services*. O'Reilly, 2007.

[12] MongoDB. Mongodb documentation. http://docs.mongodb.org/manual/.

[13] NodeJS. Nodejs documentation. https://nodejs.org/documentation/.

[14] Rift Oculus VR. *Mobile Development Documentation*. Oculus VR, Rift, 2015.

[15] Komal Deepak Ajmera Khushbu Mehta Parth Rajesh Desai, Pooja Nikhil Desai. A review paper on oculus rift-a virtual reality headset. In *International Journal of Engineering Trends and Technology, Volume 13 Number 4*, pages 175–179. 2014.

[16] Matthew Petroff. Pannellum documentation. https://github.com/mpetroff/pannellum/blob/2.1.1/doc/json-config-parameters.md.

[17] Zephyr Renner. *EquirectangulartoCubic Library*, 2012.

[18] Stackoverflow. C++ performance vs. java/c#. http://stackoverflow.com/questions/145110/c-performance-vs-java-c.

[19] Dasault Systéms. Expo milano 2015 tour virtuale. http://www.virtual.expo2015.org/.

[20] Unity. Unity documentation. http://docs.unity3d.com/Manual/index.html.

[21] Unity. *Understanding Automatic Memory Management*. Unity, 2015.

[22] Wikipedia. Comparison of html5 and flash. https://en.wikipedia.org/wiki/Comparison_of_HTML5_and_Flash.

[23] Wikipedia. Create, read, update and delete. https://en.wikipedia.org/wiki/Create,_read,_update_and_delete.

[24] YouVisit. 3 strategies to recruit more out-of-state students. http://www.youvisit.com/virtual-tours/blog/3-strategies-to-recruit-more-out-of-state-students/.