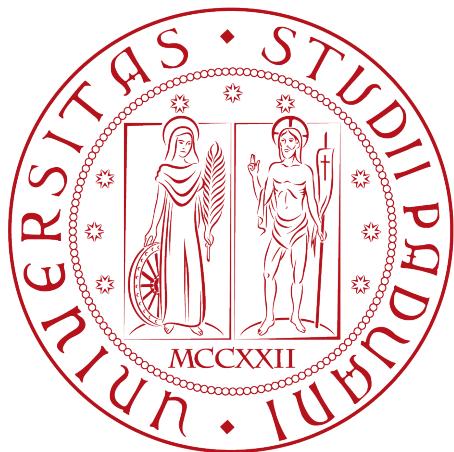


**Università degli Studi di Padova**

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA



**Realtà virtuale per colmare il divario tra  
e-commerce e negozio fisico**

*Tesi di laurea triennale*

*Relatore*

Prof. Tullio Vardanega

*Laureando*

Simone Magagna  
1009467

---

ANNO ACCADEMICO 2015-2016

Simone Magagna: *Realtà virtuale per colmare il divario tra e-commerce e negozio fisico*,  
Tesi di laurea triennale, © Ottobre 2016.

Dedica



# Sommario

Questo documento rappresenta la relazione finale di stage tenuta presso l'azienda The White Dog s.r.l. della durata complessiva pari a 308 ore.

Il primo capitolo tratta dell'azienda ospitante, dei prodotti e dei servizi che offre, in particolare all'azienda Diana Corp. della quale rappresenta il reparto ricerca e sviluppo. Il secondo capitolo descrive le strategie aziendali riguardanti l'attività di stage, di come il mio progetto si integri in esse, del piano di lavoro propostomi e degli obiettivi aziendali e personali.

Il terzo capitolo parla dell'effettivo sviluppo del progetto, descrivendo le fasi di pianificazione e ricerca effettuate con il mio tutor aziendale e il team di sviluppo, per arrivare a trattare dell'analisi dei requisiti, dei principi di progettazione perseguiti, dello sviluppo di alcune delle parti più peculiari e interessanti del progetto e della verifica e validazione.

Il quarto ed ultimo capitolo analizza i risultati ottenuti confrontandoli con gli obiettivi prefissati, valuta le conoscenze acquisite e discute quali tra queste debbano essere integrate nel corso di studi. Il capitolo si conclude con alcune considerazioni di carattere personale.



# **Ringraziamenti**

In questa sezione ci saranno i ringraziamenti.



# Indice

<b>1 The White Dog s.r.l.</b>	<b>1</b>
1.1 Chi è The White Dog s.r.l. . . . .	1
1.2 Prodotti e servizi . . . . .	1
1.2.1 Servizio di supporto tecnologico . . . . .	1
1.2.2 Il prodotto Live Story . . . . .	4
1.3 Processi interni . . . . .	5
1.4 Strumenti e tecnologie . . . . .	9
1.4.1 Ambienti di sviluppo . . . . .	9
1.4.2 Gestione dei progetti . . . . .	9
1.4.3 Versionamento . . . . .	10
1.4.4 Strumenti di automazione . . . . .	10
1.4.5 Tecnologie di sviluppo . . . . .	10
1.5 Ricerca e innovazione . . . . .	10
<b>2 Il quadro strategico</b>	<b>13</b>
2.1 Strategie aziendali di stage . . . . .	13
2.2 Il progetto di stage proposto . . . . .	13
2.2.1 Piano di lavoro proposto . . . . .	15
2.2.2 Obiettivi aziendali . . . . .	19
2.2.3 Obiettivi personali . . . . .	20
<b>3 Il progetto di e-commerce VR</b>	<b>21</b>
3.1 Analisi dei requisiti . . . . .	21
3.1.1 Caratteristiche degli utenti . . . . .	21
3.1.2 Casi d'uso . . . . .	22
3.1.3 Requisiti . . . . .	26
3.2 Progettazione . . . . .	28
3.2.1 Portabilità dell'applicazione . . . . .	28
3.2.2 Usabilità dell'applicazione . . . . .	30
3.2.3 Costruzione della scena 3D . . . . .	31
3.2.4 Interazione con gli oggetti all'interno della scena . . . . .	32
3.2.5 Progettazione e integrazione con AWS API Gateway . . . . .	33
3.3 Sviluppo . . . . .	35
3.3.1 Dati persistenti attraverso le scene . . . . .	35
3.3.2 Il pannello informativo . . . . .	37
3.3.3 Creazione e parsing di oggetti JSON in Unity . . . . .	39
3.3.4 Creazione a runtime di oggetti interattivi . . . . .	40

<b>4 Analisi retrospettiva</b>	<b>43</b>
4.1 Bilancio dei risultati rispetto agli obiettivi prefissati . . . . .	43
4.2 Bilancio formativo . . . . .	43
4.3 Analisi critica del rapporto formativo tra stage e corso di laurea . . . . .	43
4.4 Valutazioni personali . . . . .	43
<b>Glossario</b>	<b>45</b>
<b>Bibliografia</b>	<b>49</b>

# Elenco delle figure

1.1	Logo dell'azienda The White Dog s.r.l. . . . .	1
1.2	Processo d'acquisto tramite <i>e-commerce</i> . Immagine tratta dal sito dell'azienda LEVERPLAN: <a href="http://www.leverplan.com/it/agenzia_web_agency/ecommerce.aspx">http://www.leverplan.com/it/agenzia_web_agency/ecommerce.aspx</a> . . . . .	2
1.3	Ambiti aziendali gestiti dal sistema ERP. Immagine tratta dal sito della compagnia di sviluppo software KNOWART: <a href="http://www.knowarth.com/enterprise-resource-planning/">http://www.knowarth.com/enterprise-resource-planning/</a> . . . . .	4
1.4	<i>Wall mood</i> creato tramite <i>Live Story</i> per l' <i>e-commerce</i> di una nota marca di occhiali: <a href="http://www.ray-ban.com/italy">http://www.ray-ban.com/italy</a> . . . . .	5
1.5	Ciclo di sviluppo della Metodologia Agile. Immagine tratta dal sito informativo Quora: <a href="https://www.quora.com/">https://www.quora.com/</a> . . . . .	6
1.6	Competenze necessarie alla metodologia di sviluppo DevOps. Immagine tratta dal sito dell'organizzazione RTTS: <a href="http://www.rttsw.com/services/strategy/software-process-assessment">http://www.rttsw.com/services/strategy/software-process-assessment</a> . . . . .	7
1.7	Ciclo di sviluppo software Extreme Programming. Immagine tratta dalla pagina Wikipedia dedicata all' <i>extreme programming</i> : <a href="https://en.wikipedia.org/wiki/Extreme_programming">https://en.wikipedia.org/wiki/Extreme_programming</a> . . . . .	8
1.8	Esempio di <i>scrum board</i> all'interno del servizio Jira . . . . .	9
1.9	Visore per la realtà virtuale <i>Google Cardboard</i> . . . . .	11
2.1	Schema rappresentativo della differenze tra <i>single-channel</i> , <i>multi-channel</i> , <i>cross-channel</i> e <i>omni-channel</i> . Immagine tratta dal sito dell'azienda ON THE MARK: <a href="http://on-the-mark.com/omnichannel-organization-designs-biggest-test/">http://on-the-mark.com/omnichannel-organization-designs-biggest-test/</a>	14
2.2	Samsung Gear VR . . . . .	17
2.3	Stack tecnologico di sviluppo per Samsung Gear VR. Immagine tratta dal sito SAMSUNG GEAR VR DEVELOPERS US: <a href="http://www.samsung.com/us/samsungdeveloperconnection/developer-resources/gear-vr.html">http://www.samsung.com/us/samsungdeveloperconnection/developer-resources/gear-vr.html</a> . . . . .	19
3.1	Gerarchia degli utenti. L'utente autenticato acquirente può effettuare tutte le operazioni dell'utente registrato visitatore ma in più ha la possibilità di acquistare gli oggetti presenti nel carrello . . . . .	22
3.2	UC1: Autenticazione . . . . .	23
3.3	UC2: Interazione con l'ambiente virtuale . . . . .	24
3.4	UC2.1: Interazione con un oggetto . . . . .	25
3.5	UC2.2: Interazione con il carrello . . . . .	26
3.6	Camera VR posizionata all'interno dell'ambiente tridimensionale creato in Unity . . . . .	29

3.7 Schermata 2D di login . . . . .	30
3.8 Sfera inversa a cui è stata applicata la foto 360 della stanza come texture	32
3.9 Pannello tridimensionale posto davanti ad un oggetto presente nella stanza per renderlo interattivo . . . . .	33
3.10 Flusso della chiamata GET all'interno della risorsa <i>/products</i> in AWS API Gateway . . . . .	34
3.11 Pannello informativo attivato dopo la selezione di un particolare oggetto all'interno della scena . . . . .	38
3.12 Pannello informativo attivo all'esterno della sfera . . . . .	39
3.13 Carrello riempito dei prodotti presenti nella stanza . . . . .	42

# Elenco delle tabelle

3.1 Tabella dei requisiti funzionali che l'applicazione deve soddisfare . . . **27**







# Capitolo 1

## The White Dog s.r.l.

### 1.1 Chi è The White Dog s.r.l.

The White Dog s.r.l. è una realtà aziendale nata nel 2008 con sede a Torreglia, in provincia di Padova. Essa è stata fondata dal signor Stefano Mocellini, fondatore e CEO di Diana Corp.<sup>1</sup>, con la volontà di creare un *team* di lavoro focalizzato sulla ricerca e sviluppo per quest'ultima.

The White Dog s.r.l. coordina e gestisce società tutte affini al settore *e-commerce*, come Diana Corp. e LiveStory<sup>2</sup>. L'azienda possiede un reparto di ricerca e sviluppo denominato R&D, il quale esplora nuove tecnologie da applicare poi alle società figlie nel caso di esito positivo o facendo nascere nuovi progetti separati.



figura 1.1: Logo dell'azienda The White Dog s.r.l.

### 1.2 Prodotti e servizi

L'azienda svolge principalmente due attività: la prima legata al servizio di supporto tecnologico offerto a Diana Corp., la seconda legata alla gestione e sviluppo del prodotto Live Story.

#### 1.2.1 Servizio di supporto tecnologico

Il principale servizio che l'azienda offre a Diana Corp. è la ricerca e lo sviluppo di nuove tecnologie da applicare nell'ambito del *fashion e-commerce*.

---

<sup>1</sup><http://www.dianacorp.com/>

<sup>2</sup><http://www.livestory.nyc/>

*E-commerce* è l'acronimo di *Electronic Commerce* e consiste nella presentazione, vendita e gestione di prodotti utilizzando strumenti elettronici, in particolare siti internet dedicati. Avere un sito *e-commerce*, o implementare il pagamento degli acquisti sul proprio sito web, offre la possibilità ad una azienda di estendere a livello globale i propri potenziali clienti, espandendo così il proprio *business*. La sicurezza delle operazioni di acquisto viene garantita tramite l'utilizzo di server sicuri, caratterizzati dall'indirizzo HTTPS, un apposito protocollo che crittografa i dati sensibili dei clienti contenuti nell'ordine di acquisto allo scopo di tutelare il consumatore.



**figura 1.2:** Processo d'acquisto tramite *e-commerce*. Immagine tratta dal sito dell'azienda LEVERPLAN: [http://www.leverplan.com/it/agenzia\\_web\\_agency/ecommerce.aspx](http://www.leverplan.com/it/agenzia_web_agency/ecommerce.aspx)

Diana Corp., principale cliente di The White Dog s.r.l. e sua azienda d'origine, propone ai suoi clienti un portale *e-commerce* che comprende la gestione dei prodotti, del design, dell'infrastruttura e della manutenzione. Inoltre, il pacchetto può essere arricchito sia con soluzioni di marketing, come *newsletter* o campagne, sia con la gestione delle spedizioni, ospitando i prodotti nel magazzino dell'azienda. Alcune delle

principali piattaforme sviluppate da Diana Corp. sono ad esempio The Blonde Salad<sup>3</sup> e Pryma<sup>4</sup>.

The White Dog s.r.l. svolge un'attività di *testing*<sub>G</sub> dei nuovi strumenti presenti nel mercato da applicare agli *e-commerce* di Diana Corp., li valuta attentamente in termini di prestazioni e costi, per poi renderli disponibili agli sviluppatori di quest'ultima attraverso consulenze.

The White Dog s.r.l., inoltre, interviene direttamente nel codice dei servizi Diana Corp. se a quest'ultima vengono commissionate nuove *features* o applicazioni che, per tempistiche o competenze, non è in grado da sola di sviluppare.

Infine, offre consulenza a Diana Corp. per l'utilizzo delle seguenti piattaforme di *management*:

- \* **SAP:** sistema informativo per la gestione di tutti i processi aziendali. SAP supporta il sistema di gestione ERP, acronimo di *Enterprise Resource Planning*, il quale, come è visibile nella [figura 1.3](#), integra e gestisce tutti i processi di *business* rilevanti di un'azienda: vendite, acquisti, gestione magazzino, contabilità, produzione e salvataggio dei dati. ERP fornisce una visione dei processi di business, spesso in tempo reale, utilizzando *database* comunemente gestiti da un *database management system*. Esso traccia risorse come contanti, materie prime, capacità di produzione e lo stato degli impegni lavorativi come ordini, ordini d'acquisto e libro paga. Facilita il flusso di informazioni tra tutte le funzioni aziendali e gestisce le connessioni agli *stakeholder*<sub>G</sub> esterni.

Nel caso dell'azienda Diana Corp. il sistema ERP contiene le informazioni riguardanti i prodotti e lo stato del magazzino. Alcune delle informazioni, come la descrizione e la disponibilità del prodotto, vengono sincronizzate con la piattaforma di vendita.

Al termine di ogni ordine, il portale *e-commerce* notifica il sistema ERP le seguenti operazioni:

- Contabilizzazione del pagamento;
- Aggiornamento dello stato del magazzino, decrementando la quantità disponibile della merce appena ordinata;
- Generazione della fattura.

- \* **Magento:** è una piattaforma *e-commerce open source* che offre la possibilità di avviare una attività online di commercio elettronico con ampia flessibilità sia nella grafica, sia nelle funzionalità che nei contenuti.

L'interfaccia di amministrazione fornisce strumenti di marketing, SEO e gestione del catalogo, in modo da offrire ai commercianti la possibilità di creare siti a misura delle proprie esigenze di business. Progettato per essere completamente scalabile e per essere ripristinato facilmente.

- \* **WordPress:** è una piattaforma software di *personal publishing* e *content management system*<sub>G</sub>, ovvero un programma che consente la creazione e distribuzione di un sito internet formato da contenuti testuali o multimediali, facilmente gestibili ed aggiornabili in maniera dinamica.

---

<sup>3</sup><http://www.theblondesalad.com/it/>

<sup>4</sup>[http://www.pryma.com/it\\_it/](http://www.pryma.com/it_it/)



**figura 1.3:** Ambiti aziendali gestiti dal sistema ERP. Immagine tratta dal sito della compagnia di sviluppo software KNOWART: <http://www.knowarth.com/enterprise-resource-planning/>

### 1.2.2 Il prodotto Live Story

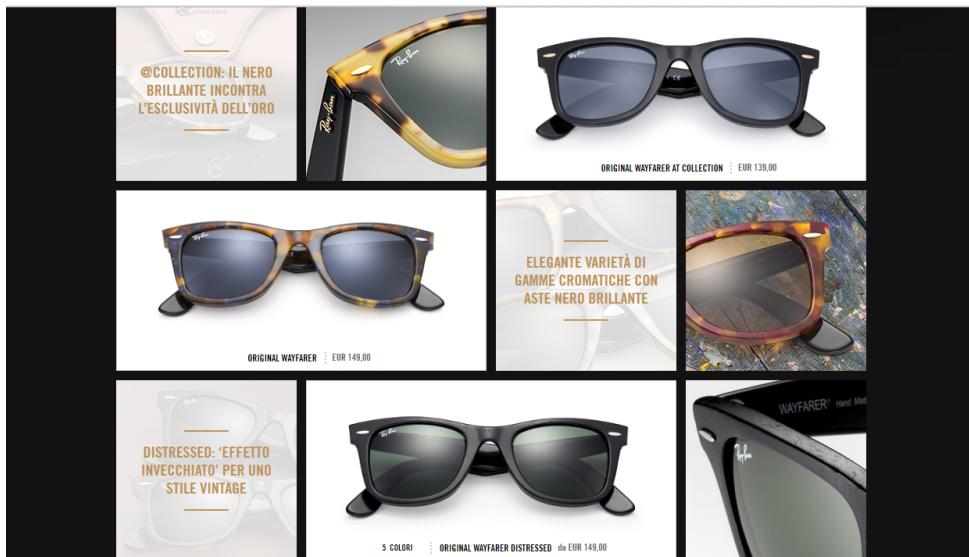
Live Story è un *social management system* che gestisce contenuti *social* e li rende acquistabili. Il *concept* di Live Story è nato nel reparto R&D di The White Dog s.r.l., *concept* che è diventato poi azienda nel 2015 con sede a New York.

Live Story è una piattaforma, resa disponibile alle aziende di moda, che colleziona foto degli utenti dei *social network*, come *Facebook*, *Instagram* e *Twitter*, marcate da particolari *hashtag*. Tali *hashtag* rappresentano i vari prodotti dell'azienda che utilizza il servizio Live Story, la quale invita i propri clienti ad utilizzarli all'interno dei *social media*. Il sistema accoppia la foto marcata ad un particolare prodotto presente nel catalogo aziendale e genera automaticamente le richieste di permesso di sfruttamento della foto, inviandola all'utente interessato. Se l'utente approva e il moderatore aziendale ritiene conforme la foto, l'azienda può utilizzare il contenuto nel proprio sito o *e-commerce*.

Una volta approvati, i contenuti possono comporre un *wall*, ovvero un pannello dei prodotti visibile in una pagina web. Vi sono due tipi di *wall* che Live Story permette di comporre:

- \* **Mood:** *wall* creati con contenuti e foto scelti dall'azienda. In questo caso non vi è nessuna interazione con gli utenti dei *social network*, ma solo una creazione più rapida dei contenuti visualizzabili nell'*e-commerce*;
- \* **Feed:** *wall* creati automaticamente tramite i contenuti marcati dagli *hashtag* aziendali presenti nei *social network*.

Ogni *wall* è personalizzabile e ogni *brand* può definire il proprio stile tramite fogli CSS.



**figura 1.4:** *Wall mood* creato tramite *Live Story* per l'e-commerce di una nota marca di occhiali: <http://www.ray-ban.com/italy>

### 1.3 Processi interni

Lo sviluppo del software a The White Dog s.r.l. segue una metodologia tipicamente Agile<sup>5</sup>.

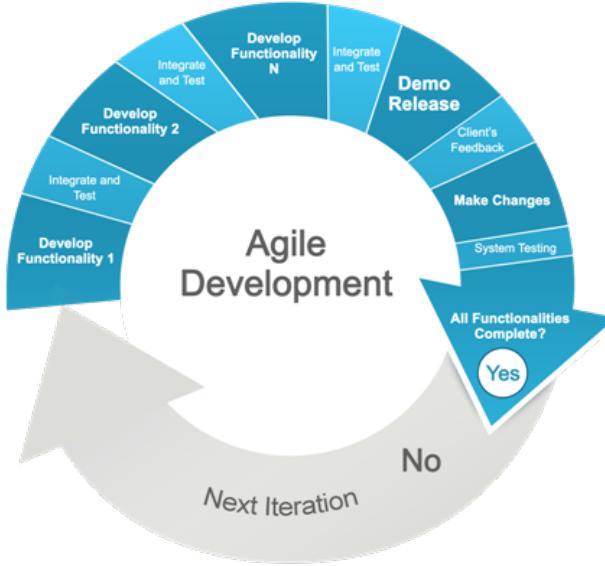
La Metodologia Agile si riferisce ad un insieme di metodi di sviluppo software fondati su principi comuni, direttamente o indirettamente derivati dai principi del *Manifesto per lo Sviluppo Agile di Software*:

- \* **Gli individui e le interazioni** più che i processi e gli strumenti;
- \* **Il software funzionante** più che la documentazione esaustiva;
- \* **La collaborazione col cliente** più che la negoziazione dei contratti;
- \* **Rispondere al cambiamento** più che seguire un piano.

I metodi agili si contrappongono al modello a cascata e ad altri processi software tradizionali, proponendo un approccio meno strutturato e focalizzato sull'obiettivo di consegnare al cliente, in tempi brevi e frequentemente, software funzionante e di qualità. Fra le pratiche promosse dai metodi agili ci sono la formazione di team di sviluppo piccoli, cross-funzionali e auto-organizzati, lo sviluppo iterativo e incrementale, la pianificazione adattiva e il coinvolgimento diretto e continuo del cliente nel processo di sviluppo.

---

<sup>5</sup><http://agilemanifesto.org/>



**figura 1.5:** Ciclo di sviluppo della Metodologia Agile. Immagine tratta dal sito informativo Quora: <https://www.quora.com/>

L’azienda The White Dog s.r.l. è composta da un *team* di sviluppo formato da sole tre persone, le quali possiedono competenze sia software, sia decisionali che di controllo della qualità. Questo permette all’azienda di essere molto agile sia nello sviluppo che nel rilascio del software.

Essa dà molta importanza agli individui presenti e all’interazione tra di loro, eliminando completamente la gerarchia lavorativa classica e impegnandosi molto per mantenere un ambiente lavorativo di reciproco rispetto paritario.

La collaborazione col cliente avviene in maniera assidua e giornaliera tramite riunioni nell’ufficio R&D o con trasferte. Se entrambe le soluzioni non sono attuabili, vengono optate sessioni di video conferenze. Questa pratica è agevolata dal fatto che per volontà del fondatore, The White Dog s.r.l. ha sede all’interno dello stesso stabilimento di Diana Corp., suo principale cliente.

Infine la natura dell’azienda la obbliga a rispondere al cambiamento in maniera molto veloce e repentina, dovendo pianificare così molti cicli di *refactoring* per lo sviluppo software.

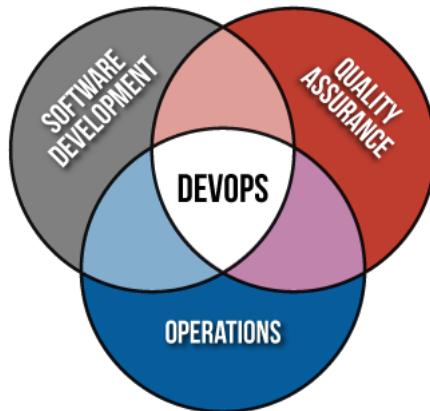
Le tre principali metodologie di sviluppo, derivanti da quella Agile, che l’azienda adotta sono:

### DevOps

Metodologia di sviluppo software che punta alla comunicazione, collaborazione e integrazione tra gli sviluppatori e addetti alle *operations* dell’*information technology*. DevOps vuole rispondere all’interdipendenza tra sviluppo software e IT *operations*,

puntando ad aiutare un'organizzazione a sviluppare in modo più rapido ed efficiente prodotti e servizi.

L'integrazione *DevOps* ha come obiettivo il rilascio del prodotto, il collaudo del software, l'evoluzione e il mantenimento in modo tale da aumentare affidabilità e sicurezza e rendere più veloci i cicli di sviluppo e rilascio.



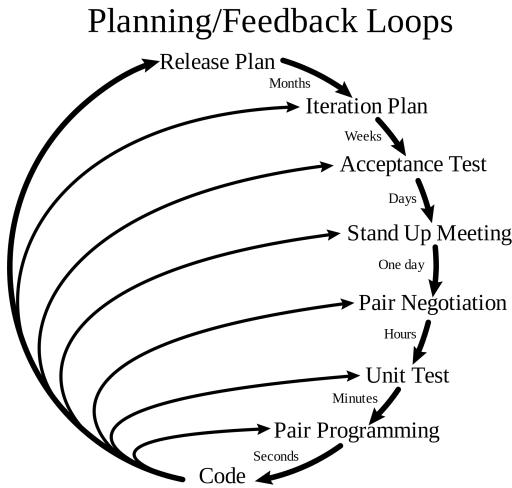
**figura 1.6:** Competenze necessarie alla metodologia di sviluppo DevOps. Immagine tratta dal sito dell'organizzazione RTTS: <http://www.rttsw.com/services/strategy/software-process-assessment>

In The White Dog s.r.l. questo principio è concretizzato dal fatto che ogni membro possiede sia le competenze di sviluppo, sia amministrative che di controllo della qualità, migliorando così di molto l'efficienza e l'agilità nello sviluppo del software.

Questa metodologia aiuta e migliora anche i continui rilasci che l'azienda giornalmente è costretta a fare per migliorare i servizi di Diana Corp. o per aumentarne le potenzialità con nuove tecnologie.

### Extreme Programming

Metodologia di sviluppo software che enfatizza la scrittura di codice di qualità e la rapidità di risposta ai cambiamenti di requisiti. Prescrive lo sviluppo iterativo e incrementale, soprattutto in brevi cicli di sviluppo. Suggerisce inoltre l'uso sistematico di *unit testing* e *refactoring*, vietando ai programmatore di sviluppare codice non strettamente necessario. Sostiene la chiarezza e la semplicità del codice, preferisce strutture gestionali non gerarchiche e dà molta importanza alla comunicazione diretta e frequente fra sviluppatori e cliente e fra gli sviluppatori stessi.



**figura 1.7:** Ciclo di sviluppo software Extreme Programming. Immagine tratta dalla pagina Wikipedia dedicata all'*extreme programming*: [https://en.wikipedia.org/wiki/Extreme\\_programming](https://en.wikipedia.org/wiki/Extreme_programming)

Il *team* di sviluppo di The White Dog s.r.l. fa ampio utilizzo di questa metodologia, spingendo molto sulla semplicità del codice prodotto, che andrà a migliorare e ampliare i servizi *e-commerce* di Diana Corp. e Live Story e che quindi dovrà poi essere compreso dai loro sviluppatori.

Prevede continui cicli di *refactoring* per adattarsi alla volubilità del cliente finale e di *unit testing* per garantirgli codice funzionante e di qualità.

### Scrum

*Framework* agile di sviluppo software, iterativo ed incrementale, concepito per gestire progetti e prodotti software. Esso enfatizza tutti gli aspetti di gestione di progetto legati a contesti in cui è difficile pianificare in anticipo. Vengono utilizzati meccanismi propri di un processo di controllo empirico, in cui i cicli di *feedback*, che ne costituiscono le tecniche di *management* fondamentali, risultano in opposizione alla gestione basata sul concetto tradizionale di *command-and-control*. Il suo approccio alla pianificazione e gestione dei progetti è quello di portare l'autorità decisionale al livello di proprietà e certezze operative.

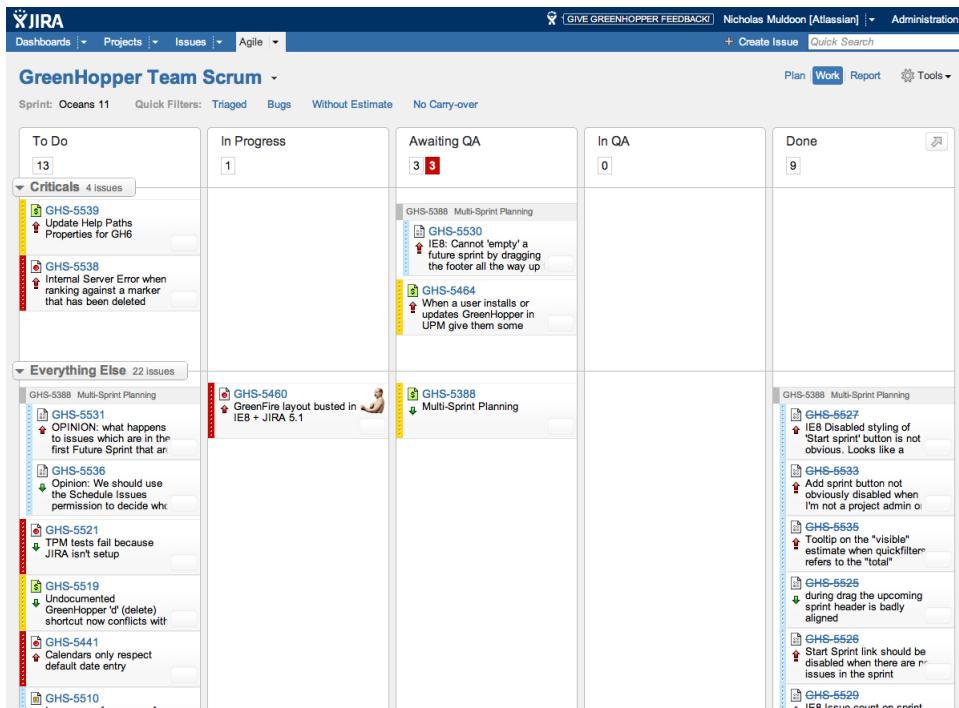


figura 1.8: Esempio di *scrum board* all'interno del servizio Jira

Data la forte difficoltà aziendale a pianificare in anticipo lo sviluppo software, causata dalla continua richiesta di modifiche e nuove funzionalità dei servizi *e-commerce* che ha in gestione, l'azienda fa ampio utilizzo di questa metodologia di pianificazione del lavoro utilizzando strumenti di *project management* come Wrike<sup>6</sup> e Jira<sup>7</sup>. I due software supportano il principio *Scrum* fornendo all'utente una *Scrum board*: tavola virtuale nella quale è possibile visualizzare il flusso di lavoro suddiviso in sezioni. Ad ogni sezione sono assegnati uno o più sviluppatori che annotano l'avanzamento del lavoro, i *bug* e i problemi riscontrati.

## 1.4 Strumenti e tecnologie

### 1.4.1 Ambienti di sviluppo

Il sistema operativo adottato dall'azienda è Mac OS X installato su macchine iMac. L'ambiente di sviluppo predefinito è Eclipse<sup>8</sup>, ma, data la natura aziendale, varia molto spesso in base al prodotto in fase di sviluppo, che può cambiare in maniera repentina.

### 1.4.2 Gestione dei progetti

I due principali strumenti utilizzati da The White Dog s.r.l. per il *project management* e l'*issue tracking*<sup>9</sup> sono rispettivamente Wrike e Jira.

<sup>6</sup><https://www.wrike.com/it/it/>

<sup>7</sup><https://www.atlassian.com/software/jira>

<sup>8</sup><https://eclipse.org/>

Wrike, sviluppato dall'omonima casa, è uno strumento per la collaborazione e il *project management*. Permette ai suoi utenti di modificare progetti, classificare le attività per importanza, tenere traccia dei programmi e collaborare con altri utenti dello stesso gruppo.

Jira, prodotto dall'azienda Atlassian, è un software di *bug tracking*<sup>9</sup>, *issue tracking*<sup>9</sup> e *project management*. Esso permette di tenere traccia delle azioni e dei problemi degli utenti, di distribuire i compiti all'interno del *team*, discutere del lavoro in atto con una visibilità completa e migliorare le prestazioni della squadra visualizzando dati in tempo reale.

#### 1.4.3 Versionamento

Il principale software di controllo di versione distribuito utilizzato dall'azienda è Git<sup>9</sup>. Git supporta lo sviluppo non lineare con diramazione e fusioni rapide e continue e comprende strumenti specifici per visualizzare e navigare una cronologia di sviluppo. Permette ad ogni sviluppatore una copia locale dell'intera cronologia di sviluppo e le modifiche vengono importate da un *repository*<sup>9</sup> ad un altro. I *repository*<sup>9</sup> possono essere pubblicati facilmente tramite protocolli HTTP, FTP, SSH, RSYNC o uno speciale protocollo git.

#### 1.4.4 Strumenti di automazione

The White Dog s.r.l. fa ampio utilizzo di strumenti di automazione per facilitare il *deployment*<sup>10</sup> del software prodotto.

Il principale strumento utilizzato dall'azienda è Docker<sup>10</sup>. Docker è un progetto *open-source* che automatizza il *deployment*<sup>10</sup> delle applicazioni all'interno di *container software*<sup>10</sup>, fornendo un'astrazione addizionale grazie alla virtualizzazione a livello di sistema operativo Linux. Docker utilizza le funzionalità di isolamento delle risorse del kernel Linux, come ad esempio *cgroups* e *namespaces* per consentire a container indipendenti di coesistere sulla stessa istanza di Linux, evitando l'installazione e la manutenzione di una macchina virtuale.

#### 1.4.5 Tecnologie di sviluppo

Vista la varietà delle ricerche e dei prodotti sviluppati dall'azienda The White Dog s.r.l., le tecnologie di sviluppo sono sempre in continua evoluzione e cambiamento. I servizi di mantenimento e sviluppo di nuove funzionalità offerti agli *e-commerce* di Diana Corp. la portano comunque a lavorare giornalmente con le seguenti tecnologie web come: Java, JavaScript, Node.js, MongoDB, PHP, HTML e CSS.

### 1.5 Ricerca e innovazione

R&D rappresenta il reparto di ricerca e sviluppo dell'azienda The White Dog s.r.l.. Ha a disposizione diversi dispositivi per la ricerca come *smartphone* di ultima generazione, *smart TV*, *smartwatch* e numerosi dispositivi per lo sviluppo *AR*<sup>9</sup> e

---

<sup>9</sup><https://git-scm.com/>

<sup>10</sup><https://www.docker.com/>

*VR<sub>G</sub>* come *Google Glass*<sup>11</sup>, *Oculus Rift Development Kit 2*<sup>12</sup>, *Gear VR*<sup>13</sup>, *Google Cardboard*<sup>14</sup> e *Leap Motion*<sup>15</sup>. Attraverso questi dispositivi l'azienda studia e sviluppa nuove modalità di interazione che l'utente finale può utilizzare nell'acquisto nei propri *stores* digitali.

Un esempio di progetto R&D è rappresentato proprio dal mio progetto di stage. Attraverso le tecnologie *VR<sub>G</sub>* sopracitate, il reparto R&D mira a creare un ambiente virtuale dove sia possibile la visualizzazione e l'acquisto di alcuni prodotti. L'utente che indosserà il visore si ritroverà ad ammirare l'interno di un negozio fisico, creato con una foto a 360 gradi o tramite modellazione 3D. Girando la testa, potrà comprendere l'esperienza *VR<sub>G</sub>* accorgendosi di visualizzare diversi angoli del negozio. Verrà data la possibilità di interagire, attraverso i dispositivi fisici del visore, con alcuni prodotti esposti nel negozio che, alla selezione, attiveranno un pannello informativo 3D. Il pannello stesso sarà interattivo, permettendo la consultazione delle informazioni e delle foto relative al prodotto. Infine, ogni prodotto sarà acquistabile aggiungendolo ad un carrello virtuale.

Questo progetto rientra completamente nelle competenze di R&D in quanto le tecnologie che dovranno essere utilizzate sono in larga parte embrionali e incomplete, perciò molto dovrà essere sviluppato "in casa".



**figura 1.9:** Visore per la realtà virtuale *Google Cardboard*

<sup>11</sup><https://www.google.com/glass/start/>

<sup>12</sup><https://www3.oculus.com/en-us/rift/>

<sup>13</sup><https://www3.oculus.com/en-us/gear-vr/>

<sup>14</sup><https://vr.google.com/cardboard/>

<sup>15</sup><https://www.leapmotion.com/>



## Capitolo 2

# Il quadro strategico

### 2.1 Strategie aziendali di stage

L'azienda The White Dog s.r.l. accoglie e offre l'attività di stage per due principali motivi:

- \* **Sperimentazione su progetti innovativi:** data la forte propensione alla ricerca dell'azienda, esistono numerosi campi che essa vorrebbe esplorare ma che a causa di altri progetti più prioritari e scarsità di tempo non può studiare. Offre quindi allo studente universitario un progetto di ricerca e sviluppo su tecnologie innovative ed interessanti, non pretendendo alcun risultato da subito inseribile nel mercato. Questo permette allo studente di vivere l'esperienza dello stage in piena libertà e serenità, riuscendo così a portare un notevole valore aggiunto personale che l'azienda è ben felice di accogliere;
- \* **Valutazione dello stagista:** l'azienda è in continua crescita e necessita di nuovo personale preparato e soprattutto capace di lavorare in costante sintonia col gruppo. Lo stage universitario permette all'azienda di scoprire persone che soddisfano questi due importanti requisiti per una futura assunzione.

Da parte sua The White Dog s.r.l. offre molto agli stagisti. I tutor aziendali supportano lo studente per tutto il periodo lavorativo, consigliandolo sia per quanto riguarda il piano di lavoro, sia sulle tecnologie da utilizzare sia effettuando proficue discussioni in vista della relazione finale. Allo studente viene offerto un ambiente di lavoro accogliente e strumenti aggiornati e all'avanguardia, supportandolo anche economicamente prevedendo un rimborso spese.

### 2.2 Il progetto di stage proposto

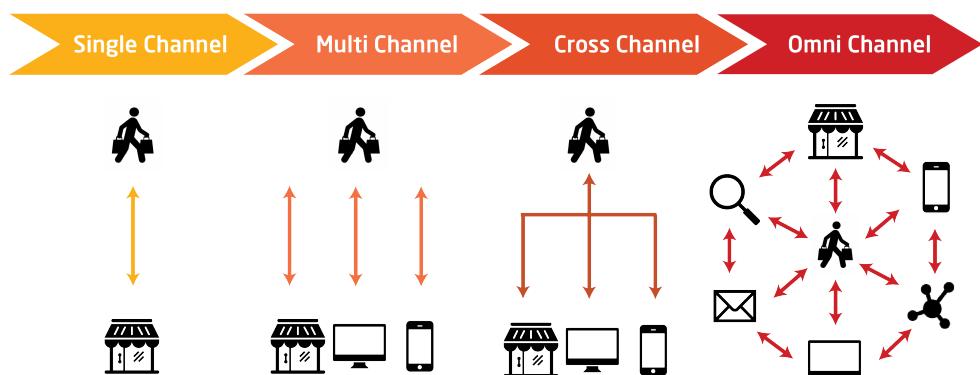
Il progetto propostomi nasce dalla costante volontà aziendale di ricercare nuove metodologie di interazione da proporre agli utenti dei suoi *e-commerce* in ottica *omni-channel*. Il modello *omni-channel* si sta lentamente ma inesorabilmente affermando come principale modello di *retailing*<sup>G</sup> a livello globale e si basa sulle seguenti caratteristiche:

- \* Concezione e management unitario della distribuzione;
- \* Processi basati sull'interazione, comunicazione e interdipendenza tra i *team* dedicati ai singoli canali;

- \* Approccio dinamico al consumatore, che richiede un monitoraggio in tempo reale delle evoluzioni dei comportamenti di acquisto e delle risposte alle iniziative promosse;
  - \* Predisposizione di adeguati strumenti IT e marketing in grado di sfruttare ed assecondare il fenomeno della cross-canalità dei processi di acquisto;
  - \* Impatto competitivo decisivo delle scelte organizzative e di investimento nell'IT e nel marketing digitale;
  - \* Impiego di indicatori di prestazioni e di sistemi di *monitoring* adeguati al nuovo contesto.

Questo significa, da parte delle aziende, offrire un'unica *customer experience* capace di rispondere in modo adeguato alle aspettative del consumatore omnicanale che si informa sui prodotti in mobilità, li prova e li sperimenta in negozio per poi acquistarli in loco o online. In termini pratici, questo si implementa costruendo un unico profilo del consumatore, immagazzinandovi dati sulle ricerche da lui effettuate, sulle scelte e sui dati personali immessi per poi sfruttarli in ogni tipologia di *store*, sia digitale che fisico.

La realtà virtuale si pone nella strategia *omni-channel* come punto di connessione tra *store* digitale e fisico, permettendo all'utente di esplorare il negozio nella sua totalità e di osservarne i prodotti esposti da più angolazioni o indossati da modelli o manichini, nel caso dei capi di abbigliamento, il tutto con un alto livello di definizione. Nasce così l'idea di un *e-commerce VR*, progetto in grado di colmare, in parte, quel divario che da sempre ha distanziato *store* virtuale e negozio fisico.



**figura 2.1:** Schema rappresentativo delle differenze tra *single-channel*, *multi-channel*, *cross-channel* e *omni-channel*. Immagine tratta dal sito dell'azienda ON THE MARK: <http://on-the-mark.com/omnichannel-organization-designs-biggest-test/>

L'obiettivo di stage è, dunque, un'esplorazione tecnologica nel campo della *virtual reality*<sup>G</sup>. Il progetto mira ad arrivare ad un prototipo di *virtual showroom* dove poter esplorare ed interagire con i prodotti e permetterne l'acquisto.

Il progetto era stato inizialmente diviso in due parti, per due differenti studenti:

- \* La prima parte riguardava la progettazione e realizzazione del movimento in uno spazio 3D ed interazione con gli oggetti;

- \* La seconda parte trattava, invece, la progettazione e realizzazione di un’interfaccia di presentazione del prodotto, con integrazione al processo di acquisto mediante l’uso di sistemi *cloud* esterni.

Purtroppo, nessun altro studente oltre a me ha aderito al progetto, dunque abbiamo ritenuto opportuno rivisitare l’obiettivo di stage. Dopo una riunione effettuata prima dell’inizio dello stage con il mio tutor aziendale, abbiamo deciso di mantenere intatti tutti gli obiettivi di ricerca, abbassando il livello qualitativo richiesto. Questo perché lo scopo ultimo di questo stage non era sviluppare un’applicazione o un servizio immediatamente vendibile, ma di studiare le potenzialità e i limiti di questa nuova tecnologia. Questa volontà è stata dettata anche dal fatto che il *team* aziendale inizialmente non aveva alcuna certezza che la tecnologia *VR<sub>G</sub>* fosse applicabile al mondo *e-commerce*.

### 2.2.1 Piano di lavoro proposto

#### Piano temporale

In accordo col tutor aziendale, la durata massima dello stage è stata fissata a 320 ore, divise in 8 settimane lavorative di 5 giorni, 8 ore al giorno.

Il piano lavorativo è stato dunque pianificato per settimana nel seguente modo:

- \* **Settimana 1:** settimana dedicata completamente alla ricerca, per colmare il *deficit* culturale personale e aziendale sulle tecnologie *VR<sub>G</sub>*. Le attività principali previste sono: analisi dei requisiti funzionali del sistema da sviluppare e studio delle tecnologie e linguaggi disponibili riguardanti la realtà virtuale;
- \* **Settimana 2:** in base ai risultati ottenuti nella prima settimana, viene richiesta una scelta dell’hardware da utilizzare e un *framework<sub>G</sub>* di sviluppo, testandoli con un primo prototipo di scena 3D;
- \* **Settimana 3:** previste attività di raffinamento della scena 3D, progettazione e sviluppo degli oggetti e loro comportamento nello spazio 3D. Viene creato così un primo prototipo di *user interaction*;
- \* **Settimana 4:** previste attività di progettazione e sviluppo integrazione tra sistema *VR<sub>G</sub>* e *e-commerce*. Progettazione di *user interaction* per la fruizione dei contenuti provenienti dall’*e-commerce*;
- \* **Settimana 5:** settimana dedicata all’approfondimento di *user interaction* e del comportamento degli oggetti nell’ambiente virtuale;
- \* **Settimana 6:** previste attività di studio e prototipazione del possibile processo d’acquisto all’interno dell’ambiente virtuale;
- \* **Settimana 7:** la settima settimana rappresenta una *milestone* importante per il progetto: conclusione del prototipo e relativa documentazione, raggiungendo così gli obiettivi minimi;
- \* **Settimana 8:** l’ultima settimana viene dedicata completamente allo studio del modello emergente *omni-channel<sub>G</sub>* e come la realtà virtuale possa estendere questo modello. Vengono così raggiunti gli obiettivi massimi.

### Piano metodologico

Assieme al tutor aziendale, abbiamo fin da subito concordato la mia presenza durante l'orario d'ufficio, permettendo così un interazione intensa e costante.

Il lavoro di ricerca e sviluppo che ho effettuato è stato totalmente autonomo, con giornaliere interazioni con il personale solo per raccogliere e analizzare la documentazione, requisiti e *feedback* sull'andamento del progetto.

Le revisioni di progetto sono avvenute secondo la seguente metodologia:

- \* Riunione breve di 15 minuti ogni mattina;
- \* Riunione di 1 ora alla fine di ogni settimana come analisi retrospettiva.

Alle revisioni, oltre a me, hanno partecipato:

- \* **Valentino Baraldo**, *cloud engineer* e tutor aziendale. Oltre a svolgere il compito di tutor aziendale, mi ha supportato sulla progettazione architettonica del progetto e sull'utilizzo del servizio *API Gateway* di *Amazon Web Services*<sup>1</sup>;
- \* **Francesco Paggini**, *front-end developer*. Ha supervisionato il mio lavoro grafico nell'ambiente di sviluppo *Unity*<sup>2</sup>.

### Piano tecnologico

Inizialmente lo *stack* tecnologico propostomi riguardava solamente l'hardware che l'azienda aveva acquistato per questo progetto, senza alcun vincolo software. I dispositivi che permettevano la sperimentazione *VR<sub>G</sub>* erano:

- \* **Oculus Rift Development kit 2**: visore per la realtà virtuale per uso desktop. Possiede uno schermo Samsung OLED 2160x1200 pixel (1080x1200 per occhio), con un *refresh rate* a 90 Hz e un ampio angolo di visione a 110 gradi. Dotato di accelerometro, giroscopio, magnetometro e *tracking* posizionale a 360 gradi. Viene accoppiato ad una telecamera infrarossi per il rilevamento di profondità, assieme a 40 emettitori infrarossi all'interno dell'*headset*. Monta due lenti in alta definizione possedendo 6 gradi di libertà di rotazione;
- \* **Samsung Gear VR**: visore per la realtà virtuale per uso *mobile*. Possiede: accelerometro, giroscopio e sensore di prossimità, permettendo un campo visivo di 96 gradi. Il visore incorpora inoltre un'interfaccia utente fisica: *touch pad*, tasto indietro e tasto per il volume. Necessita l'inserimento di uno *smartphone* Samsung a partire dalla versione *Galaxy S6*;
- \* **Google Cardboard**: con il termine *Google Cardboard* non si intende specificare un particolare visore per la realtà virtuale prodotto fisicamente da *Google*, ma un insieme di linee guida suggerite da questa per costruire un dispositivo a basso costo per l'uso *mobile*. Non possiede accelerometro, giroscopio o sensore di prossimità, è solo un semplice "occhiale" che permette la visione stereoscopica. Ogni visore *Cardboard*, dunque, necessita l'inserimento di uno *smartphone* che supporti applicazioni *VR<sub>G</sub>*.

In azienda erano presenti due visori che implementavano tali linee guida: *Unofficial Cardboard* e *Tera VR Box*;

---

<sup>1</sup><https://aws.amazon.com/it/>

<sup>2</sup><https://unity3d.com/>

- \* **Leap Motion:** piccola periferica USB progettata per essere posta su una scrivania reale rivolta verso l'alto. Usando 2 telecamere e 3 LED infrarossi essa osserva un'area approssimativamente a forma di semisfera di circa un metro. E' progettata per identificare dita o oggetti simili come una penna, con una precisione di 0,01 mm.



**figura 2.2:** Samsung Gear VR

Dopo un periodo di ricerca e *testing* su queste tecnologie, abbiamo deciso di intraprendere la strada *mobile* a discapito di quella desktop. Questa decisione è stata dettata principalmente da due fattori:

- \* **Requisiti hardware elevati:** per poter offrire un'esperienza fluida e piacevole, *Oculus Rift Development kit 2* abbisogna di un PC dall'hardware elevato, non accessibile all'utenza media:
  - **GPU:** NVIDIA GTX 970 / AMD R9 290;
  - **CPU:** Intel i5-4590;
  - **RAM:** 8GB;
  - **Video output:** HDMI 1.3;
  - **USB Ports:** 3 porte 3.0 più una porta 2.0;
  - **OS:** Windows 7 SP1 64 bit o superiore.
- \* **Obiettivi aziendali:** nonostante fin da subito mi sia stato chiaro che non veniva preteso alcun prodotto finale utilizzabile, l'azienda sperava però di riuscire, con questo stage, a sviluppare un primo prototipo di *virtual showroom* da poter mostrare alle fiere tecnologiche alle quali partecipa. In quest'ottica, l'utilizzo di *Oculus Rift Development kit 2* sarebbe risultato troppo scomodo sia per il trasporto e l'installazione, che per l'utilizzatore finale.

Abbiamo deciso, dunque, di sviluppare sia per *Samsung Gear VR* che per *Google Cardboard*, entrambi dispositivi *mobile* a costo contenuto. La progettazione iniziale prevedeva di sviluppare codice che fosse il più possibile indipendente dalla specifica piattaforma, così da creare un unico progetto sia per *Samsung Gear VR* che per *Google Cardboard*. Purtroppo, dopo uno studio più approfondito, questo non fu pienamente possibile poiché gli *SDK* forniti rispettivamente da Smasung e Google sono

profondamente differenti, le metodologie di implementazione degli oggetti interattivi cambiano da una piattaforma all'altra e l'interfaccia per interazione offerta dai due dispositivi è diversa:

- \* **Samsung Gear VR** fornisce all'utilizzatore un *touch pad* per selezionare oggetti interattivi e scorrere del testo e un tasto indietro per tornare alla scena o pagina precedente;
- \* **Google Cardboard** permette all'utilizzatore solamente di selezionare un oggetto attraverso lo spostamento di un magnete come è visibile in figura [figura 1.9](#).

La sezione 3.5.1 parla in maniera approfondita della progettazione, delle scelte e dei compromessi che abbiamo dovuto adottare per rendere il progetto il più possibile indipendente dalla piattaforma.

Riguardo allo stack software, alla fine della prima settimana di ricerca è andato delineandosi dome segue:

- \* Per lo sviluppo dell'ambiente tridimensionale e del comportamento degli oggetti presenti in esso ho scelto di utilizzare il *framework<sub>G</sub>* **Unity**. *Unity* è uno strumento di *authoring<sub>G</sub>* integrato e multi-piattaforma per la creazione di videogiochi 3D o altri contenuti interattivi, quali visualizzazioni architettoniche o animazioni in tempo reale. Permette di modellare ambienti e oggetti 3D tramite un editor, fornendo strumenti per modificarne la forma, il colore, posizione e l'interazione con gli altri oggetti e l'ambiente come la forza di gravità, la collisione e molto altro. Per ogni oggetto, offre la possibilità di definirne anche un comportamento attraverso script che andranno agganciati allo stesso. I linguaggi offerti da *Unity* per la composizione degli script sono *JavaScritp* e *C#*. Entrambi i linguaggi offrono le stesse potenzialità e sul piano prestazionale sono allo stesso livello. Ho deciso però di scegliere *C#* poiché è risultato essere il più utilizzato all'interno della *community Unity*;
- \* Per interfacciarsi con il visore, Samsung e Google mettono a disposizione degli ***SDK<sub>G</sub>*** dedicati e particolari oggetti *Unity* per l'esperienza *VR<sub>G</sub>*. Gli *SDK<sub>G</sub>* permettono, negli script, di utilizzare alcuni metodi per catturare l'interazione con l'interfaccia fisica del visore, come la selezione e lo scorrimento del testo. La sezione 3.5.4 parla in maniera approfondita dell'utilizzo di questi metodi e di come sia possibile rendere interattivo un oggetto. Per quanto riguarda gli oggetti *VR<sub>G</sub>* forniti, il più importante e fondamentale tra questi è la *Camera VR*. In *Unity* ogni scena possiede una o più camere che definiscono il punto di vista dell'utente. Una normale camera creata all'interno dell'ambiente non permette l'esperienza *VR<sub>G</sub>*, non essendo collegata ai sensori di inclinazione e spostamento del visore, nel caso di *Samasung Gear VR*, o del telefono, nel caso di *Google Cardboard*. Una volta inserita nella scena la *Camera VR* specifica per ogni piattaforma, essa permette fin da subito la visualizzazione a 360 gradi. Gli *SDK<sub>G</sub>* e gli oggetti vengono forniti tramite un particolare pacchetto dall'estensione *.unity* da importare nel proprio progetto;
- \* Per sperimentare l'interazione dell'applicazione con sistemi *cloud* esterni, dove recuperare e inviare informazioni, il mio tutor aziendale mi ha consigliato l'utilizzo di **API Gateway** di **Amazon Web Service**. *API Gateway* è un servizio che permette di definire *endpoint* per le operazioni HTTP, come GET, POST, PUT eccetera, dove la parte *front-end* e la parte *back-end* di un'applicazione possono

collegarsi e dialogare fra loro. Dato che lo sviluppo di un *back-end* non faceva parte degli obiettivi di stage, l'API creata è stata di tipo *mock*. Ciò significa che ogni operazione HTTP effettuata dall'applicazione verso l'API, riceve risposte statiche create all'interno di *API Gateway*.



**figura 2.3:** Stack tecnologico di sviluppo per Samsung Gear VR. Immagine tratta dal sito SAMSUNG GEAR VR DEVELOPERS US: <http://www.samsung.com/us/samsungdeveloperconnection/developer-resources/gear-vr.html>

## 2.2.2 Obiettivi aziendali

Nel *Piano di Lavoro* presentatomi, l'azienda espone gli obiettivi minimi e massimi che si aspetta di veder raggiunti alla fine delle 320 ore si stage:

\* **Obiettivi minimi:**

1. Studio delle tecnologie disponibili in ambito *VR* e stesura di un documento riassuntivo che offre un *overview* dello stato attuale della realtà aumentata;
2. Progettazione e sviluppo di un ambiente virtuale con: una scena e oggetti definiti, un comportamento associato agli oggetti, un prototipo di *user interaction* e scambio di informazioni di base con un sistema di *e-commerce*.

**\* Obiettivi massimi:**

1. Studio e prototipazione di diversi modelli di *user interaction* con l'ambiente e con gli oggetti finalizzati alla presentazione di un bene vendibile;
2. Studio e implementazione di possibili nuovi processi di acquisto in ambito *VR<sub>G</sub>*.

**2.2.3 Obiettivi personali**

Sono venuto a conoscenza di questo progetto durante l'evento di Stage IT 2016, organizzato da Confindustria Padova in collaborazione con l'Università di Padova e Venezia. Mi ha fin da subito colpito e appassionato per le tecnologie che mi avrebbe permesso di studiare, come ad esempio *Unity*. La computer grafica è da sempre un mio personale interesse e la realtà aumentata è un ambito per me molto affascinante e ricco di opportunità.

Le tecnologie proposte dall'azienda purtroppo non rientrano nel percorso di studi, dunque gli obiettivi formativi personali che mi sono posto riguardano lo studio e la sperimentazione delle tecnologie, senza pretendere di arrivare ad un risultato non prototipale:

**\* Obiettivi minimi:**

1. Conoscenza ad alto livello delle tecnologie (hardware e software) attualmente disponibili nel mercato atte a creare ambienti virtuali;
2. Conoscenza ad alto livello dei concetti principali di *e-commerce* e relative tecnologie di riferimento usate per la vendita online.

**\* Obiettivi massimi:**

1. Capacità di identificare, progettare e sviluppare ambienti virtuali, selezionando le tecnologie attualmente disponibili più appropriate per il caso d'uso;
2. Presa di coscienza dei concetti multi-channel e omni-channel e come le nuove modalità di vendita si integrino con questi modelli emergenti.

## Capitolo 3

# Il progetto di e-commerce VR

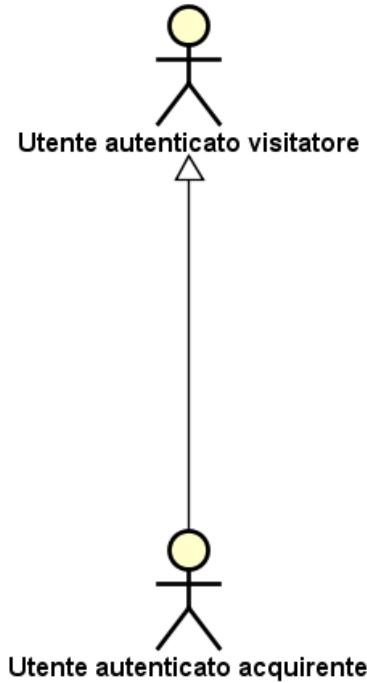
### 3.1 Analisi dei requisiti

Questa sezione tratta dei casi d'uso e dei requisiti che il *team* ha ricavato durante la discussione nella prima riunione di stage. Tale analisi ha subito un sostanzioso cambiamento durante la settimana 6, settimana dedicata alla prototipazione del possibile processo d'acquisto all'interno dell'ambiente virtuale. A causa della sua scarsa usabilità, abbiamo deciso di non utilizzare una tastiera virtuale all'interno dell'ambiente tridimensionale per dare la possibilità all'utente di immettere i propri dati e gli estremi di pagamento. Ciò ha portato il *team* a prevedere una registrazione ad un normale sito di *e-commerce* prima di effettuare l'autenticazione all'app, dove dar la possibilità all'utente di immettere i propri dati di pagamento. La sezione 3.2.2 spiega in maniera approfondita le sperimentazioni e le discussioni effettuate che hanno portato a questa importante decisione.

#### 3.1.1 Caratteristiche degli utenti

Obbligare l'utente ad immettere dati sensibili e strettamente personali, come gli estremi di pagamento, prima dell'effettivo utilizzo del servizio non è una buona prassi. L'utente, che si appresta per la prima volta ad utilizzare l'applicazione, potrebbe non conoscere l'azienda di produzione e potrebbe non fidarsi. Dunque sono stati delineati due principali tipologie d'utente:

- \* **Utente registrato visitatore:** utente che ha effettuato la registrazione ma che ha deciso di non immettere i propri dati della carta di credito. Ad esso è permesso visualizzare l'ambiente *VR<sub>G</sub>*, selezionare i prodotti e conoscerne le caratteristiche, aggiungerli al carrello, che sarà reso persistente, ma non di concluderne l'acquisto;
- \* **Utente registrato acquirente:** utente che ha effettuato la registrazione immettendo anche i dati della carta di credito. Ad esso è permessa la totale esperienza, incluso l'acquisto dei prodotti presenti nel carrello.



**figura 3.1:** Gerarchia degli utenti. L’utente autenticato acquirente può effettuare tutte le operazioni dell’utente registrato visitatore ma in più ha la possibilità di acquistare gli oggetti presenti nel carrello

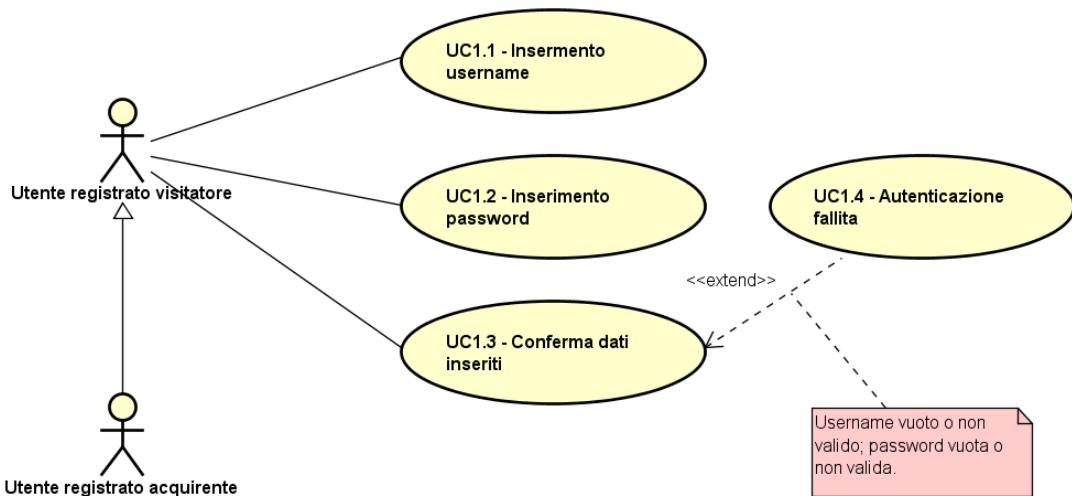
### 3.1.2 Casi d’uso

Verranno di seguito elencati tutti i casi d’uso individuati dal *team* che spiegano in che modo un utente possa interagire con l’applicazione. Per ogni caso d’uso viene mostrato uno schema *UML* che ne rappresenta il flusso operativo.  
Non vengono considerati i casi d’uso relativi alla registrazione al sito/e-commerce poiché tale funzionalità non è stata da me implementata non essendo di mia competenza.

#### Caso d’uso UC1: Autenticazione

- \* **Attori:** utente registrato visitatore, utente registrato acquirente;
- \* **Descrizione:** prima di accedere all’ambiente virtuale, l’utente viene invitato ad immettere l’*username* e la *password* scelti in fase di registrazione all’interno di una scena 2D. Potrà immettere tali dati manualmente utilizzando la normale tastiera del proprio telefono;
- \* **Precondizione:** l’applicazione è avviata e mostra la pagina di login;
- \* **Postcondizione:** l’autenticazione è andata a buon fine e l’applicazione passa in modalità *VR*.
- \* **Scenario principale:**

1. L'utente può inserire l'*username* (UC1.1);
  2. L'utente può inserire la *password* (UC1.2);
  3. L'utente può confermare i dati inseriti premendo sul pulsante di login (UC1.3).
- \* **Estensioni:** l'utente può visualizzare un messaggio di errore se i dati immessi non corrispondono a quelli di registrazione o se il campo *username* e *password* sono vuoti (UC1.4).

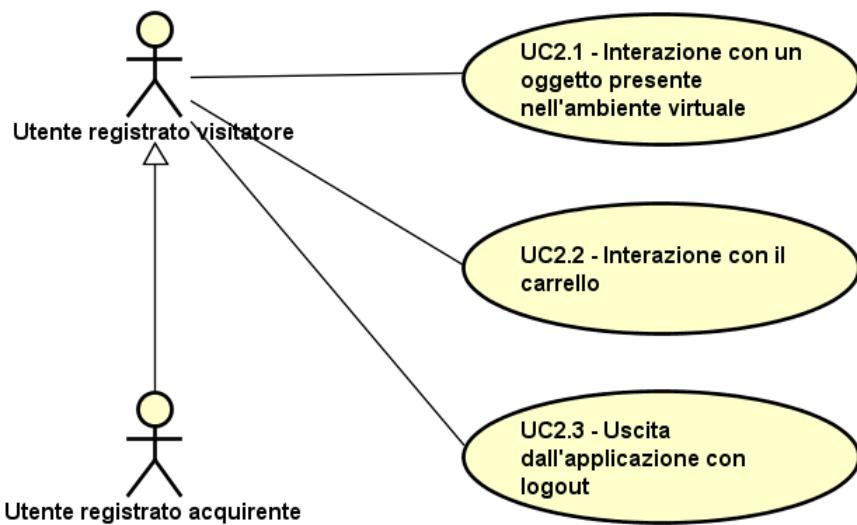


**figura 3.2:** UC1: Autenticazione

### Caso d'uso UC2: Interazione con l'ambiente virtuale

- \* **Attori:** utente registrato visitatore, utente registrato acquirente;
- \* **Descrizione:** l'utente, dopo l'autenticazione e collegato il telefono al visore, si ritrova ad osservare un ambiente virtuale. L'ambiente è formato da una stanza, la quale è visibile a 360 gradi se l'utente muove la testa mentre indossa il visore. Vi sono alcuni prodotti, segnalati da appositi marcatori, con i quali l'utente può interagire attivando un pannello informativo dove vengono visualizzati il nome, una descrizione, il prezzo e alcune foto. E' possibile aggiungere tali prodotti nel carrello, attivabile selezionando un'area segnalata da un'apposita scritta. I prodotti all'interno del carrello sono acquistabili;
- \* **Precondizione:** l'utente ha effettuato con successo l'autenticazione;
- \* **Postcondizione:** l'utente interagisce con gli oggetti presenti nell'ambiente virtuale e se è acquirente può comprare gli oggetti presenti nel carrello;
- \* **Scenario principale:**

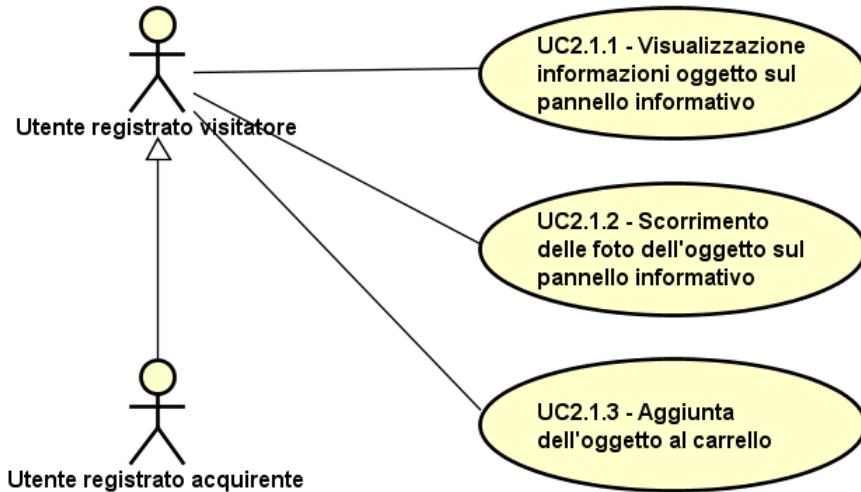
1. L'utente può interagire con un oggetto, segnalato da un apposito simbolo, presente nell'ambiente virtuale (UC2.1);
2. L'utente può interagire con il carrello presente nell'ambiente e segnalato da un'apposita scritta (UC2.2);
3. L'utente può uscire dall'applicazione effettuando così il logout (UC2.3).



**figura 3.3:** UC2: Interazione con l'ambiente virtuale

#### Caso d'uso UC2.1: Interazione con un oggetto

- \* **Attori:** utente registrato visitatore, utente registrato acquirente;
- \* **Descrizione:** l'utente interagisce con un oggetto presente nell'ambiente tramite l'interfaccia fisica del visore, attivando il pannello informativo. All'interno di questo sono riportate tutte le informazioni del prodotto e le sue foto. Dal pannello è possibile aggiungere l'oggetto al carrello;
- \* **Precondizione:** l'utente ha effettuato con successo l'autenticazione;
- \* **Postcondizione:** l'utente interagisce con l'oggetto attivando il pannello informativo.
- \* **Scenario principale:**
  1. Visualizzazione informazioni dell'oggetto sul pannello informativo (UC2.1.1);
  2. Scorrimento foto dell'oggetto sul pannello informativo (UC2.1.2);
  3. Aggiunta oggetto al carrello selezionando l'apposito pulsante (UC2.1.3).



**figura 3.4:** UC2.1: Interazione con un oggetto

#### Caso d'uso UC2.2: Interazione con il carrello

- \* **Attori:** utente registrato visitatore, utente registrato acquirente;
- \* **Descrizione:** entrambe le tipologie possono interagire con il carrello, segnalato nell'ambiente da un'apposita scritta. All'interno di esso sono visibili gli oggetti precedentemente inseriti. E' possibile, attraverso l'interfaccia fisica del visore, eliminarli uno ad uno o svuotare completamente il carrello. Se l'utente registrato è acquirente allora può procedere al pagamento;
- \* **Precondizione:** l'utente ha effettuato l'autenticazione;
- \* **Postcondizione:** l'utente visualizza i prodotti presenti nel carrello, potendoli acquistare se è acquirente;
- \* **Scenario principale:**
  1. L'utente visualizza il pannello tridimensionale che rappresenta il carrello dove sono elencati i prodotti aggiunti precedentemente (UC2.2.1);
  2. L'utente può eliminare singolarmente un oggetto dal carrello (UC2.2.2);
  3. L'utente può svuotare completamente il carrello (UC2.2.3);
  4. Se l'utente è acquirente allora può procedere con l'acquisto dei prodotti (UC2.2.4).

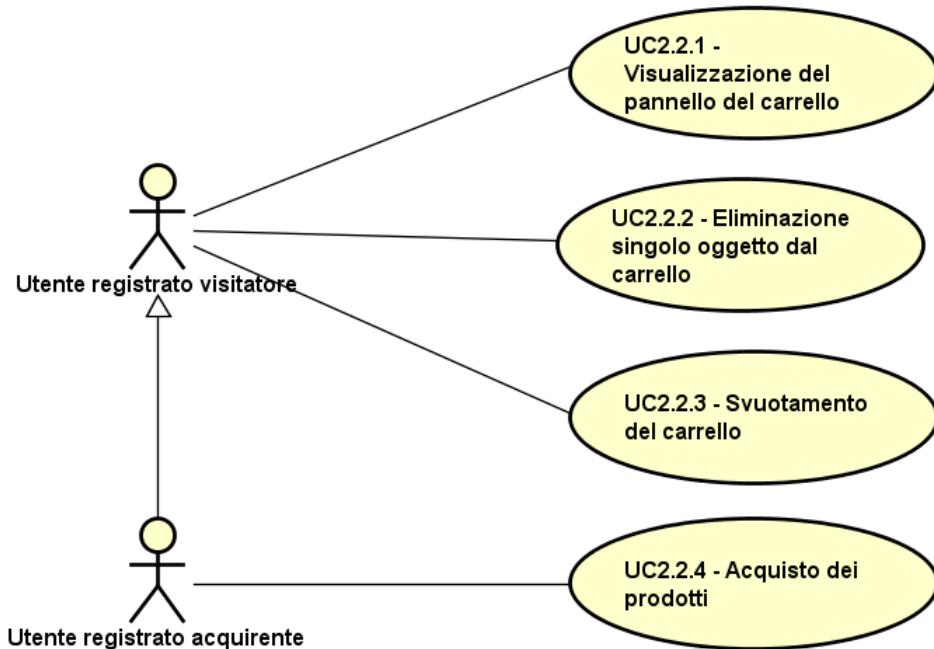


figura 3.5: UC2.2: Interazione con il carrello

### 3.1.3 Requisiti

Vengono di seguito elencati tutti i requisiti funzionali che l'applicazione deve soddisfare in base ai casi d'uso trovati. Un requisito funzionale rappresenta una *feature* che l'applicazione deve mettere a disposizione all'utente per garantirgli un'esperienza completa.

Ogni requisito funzionale è rappresentato da un codice identificativo RFx e da una descrizione che ne illustra lo scopo.

Requisito	Descrizione
RF1	L'applicazione deve permettere all'utente di potersi autenticare utilizzando <i>username</i> e <i>password</i> specificati in fase di registrazione.
RF1.1	L'applicazione deve permettere all'utente di inserire l' <i>username</i> .
RF1.2	L'applicazione deve permettere all'utente di inserire la <i>password</i> .
RF1.3	L'applicazione deve permettere all'utente di confermare i dati di autenticazione ed effettuare così il login.
RF2	L'applicazione deve permettere all'utente di visualizzare l'ambiente virtuale una volta indossato il visore e di interagire con esso.
RF2.1	L'applicazione deve permettere all'utente di interagire con un oggetto presente nella scena attraverso l'interfaccia fisica presente nel visore.
RF2.1.1	L'applicazione deve visualizzare le informazioni relative all'oggetto selezionato all'interno di un pannello informativo posto davanti all'utente.
RF2.1.2	L'applicazione deve permettere la visualizzazione e lo scorrimento delle foto dell'oggetto selezionato all'interno del pannello informativo.
RF2.1.3	L'applicazione deve permettere l'aggiunta al carrello dell'oggetto selezionato.
RF2.2	L'applicazione deve permettere l'interazione col carrello segnalato da un'apposita scritta all'interno dell'ambiente virtuale.
RF2.2.1	L'applicazione deve visualizzare tutti gli oggetti presenti nel carrello precedentemente aggiunti.
RF2.2.2	L'applicazione deve permettere all'utente di eliminare un singolo oggetto dal carrello.
RF2.2.3	L'applicazione deve permettere di svuotare completamente il carrello.
RF2.2.4	L'applicazione deve permettere l'acquisto dei prodotti se l'utente è registrato acquirente.

**tabella 3.1:** Tabella dei requisiti funzionali che l'applicazione deve soddisfare

## 3.2 Progettazione

Questa sezione descrive le più importanti e peculiari fasi di progettazione dell'applicazione. Molte delle decisioni prese sono state discusse da me assieme al *team* di The White Dog s.r.l., cercando il più possibile di accontentare gli *stakeholder<sub>G</sub>*, ovvero il signor Stefano Mocellini, fondatore dell'azienda, attraverso l'ancora giovane e incompleta tecnologia *VR<sub>G</sub>*.

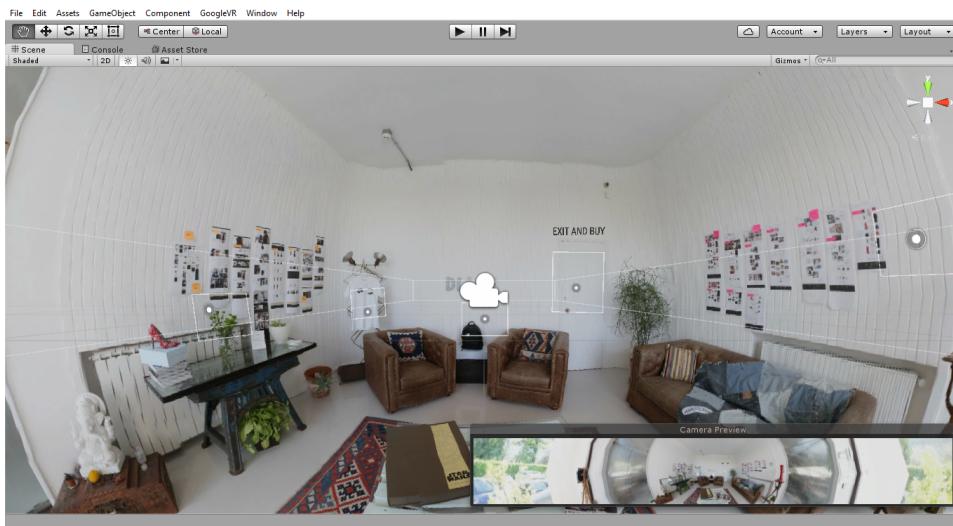
### 3.2.1 Portabilità dell'applicazione

Una delle sfide più impegnative di questo progetto era riuscire a sviluppare l'applicazione per entrambi i dispositivi: *Samsung Gear VR* e *Google Cardboard*. Prima di conoscere a basso livello tali tecnologie, avevamo progettato l'implementazione di un'interfaccia che riconoscesse la tecnologia in uso e che ne richiamasse così i metodi specifici. Purtroppo, dopo aver sperimentato gli *SDK<sub>G</sub>* forniti da entrambe le piattaforme e seguito i due principali tutorial<sup>1</sup>, mi sono reso conto che una totale portabilità non era possibile a causa delle forti differenze tra le due piattaforme.

La prima differenza, e la più importante, è la **MainCamera** o Camera Principale. In un progetto *Unity*, ogni scena possiede una o più camere che rappresentano il punto di vista dell'utente. Una camera normale non permette l'esperienza *VR<sub>G</sub>* poiché non è collegata ai sensori di movimento e rotazione del dispositivo. All'interno dei pacchetti *Unity* offerti agli sviluppatori dalle due case, sono presenti le rispettive camere che, tramite script ad esse agganciati, permettono l'esperienza *VR<sub>G</sub>* una volta posizionate a piacimento all'interno della scena. Tali script si collegano direttamente ai sensori del dispositivo che è diverso per le due piattaforme. Il visore *Samsung Gear VR* possiede al suo interno tutti i sensori di movimento e rotazione necessari e lo script, agganciato alla *MainCamera*, legge i dati derivanti da essi. Invece, i dispositivi *Google Cardboard* non possiedono alcun sensore e la *MainCamera* legge i dati di rotazione e movimento direttamente dai sensori all'interno dello *smartphone*. Questa profonda differenza hardware causa una forte differenza tra i due script della *MainCamera*, obbligando ad inserire una camera specifica per ogni piattaforma all'interno della stessa scena.

---

<sup>1</sup> *Samsung Gear VR:* <http://www.samsung.com/us/samsungdeveloperconnection/developer-resources/gear-vr.html>  
*Google Cardboard:* <https://developers.google.com/vr/unity/>



**figura 3.6:** Camera VR posizionata all'interno dell'ambiente tridimensionale creato in Unity

La seconda importante differenza è l'implementazione degli **oggetti interattivi** all'interno della scena *Unity*. Di base, sia per *Samsung Gear VR* sia per *Google Cardboard*, un oggetto per essere interattivo necessita di due caratteristiche:

- \* **Mesh Collider:** è necessario aggiungere tale proprietà all'oggetto che si vuole rendere interattivo. Tale proprietà rende un oggetto "tangibile", ovvero può essere colpito da un altro oggetto presente nella scena. Questo permette alla *Camera VR* di interagire con l'oggetto inviando un "raggio virtuale" verso quest'ultimo che, scontrandovisi, potrà attivare gli eventi che si desidera;
- \* **InteractiveScriptVR:** script, da agganciare all'oggetto, che cattura l'evento di selezione attraverso i metodi esposti dall'*api* della particolare piattaforma. All'interno di questi metodi si può implementare il vero comportamento dell'oggetto interattivo.

L'**InteractiveScriptVR** differisce profondamente da una piattaforma all'altra. Per quanto riguarda *Google Cardboard*, lo script implementa la classe **IGvrGazeResponder**, la quale espone i metodi per la cattura della selezione tramite lo spostamento del magnete posto lateralmente nel visore.

In *Samsung Gear VR* invece, l'**InteractiveScriptVR** non implementa alcuna interfaccia ma necessita l'utilizzo del namespace **VRStandardAsset.Utils** per richiamare i metodi esposti dallo script **VRInteractiveItem**, script che va agganciato all'oggetto assieme all'**InteractiveScriptVR**.

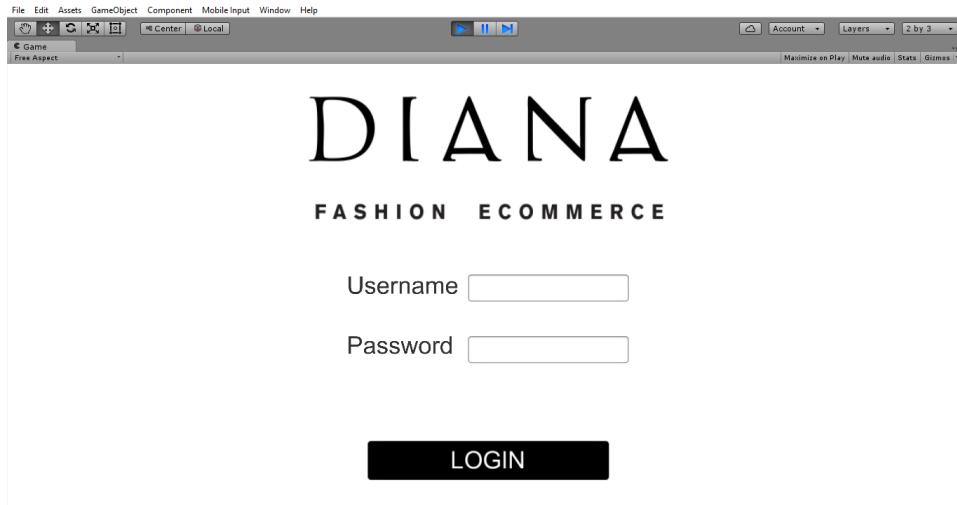
Questa differenza di implementazione degli oggetti interattivi rende davvero difficoltosa la loro portabilità da una piattaforma all'altra. Dunque, dopo una lunga discussione avvenuta nell'ufficio R&D, abbiamo deciso di abbandonare la totale portabilità dell'applicazione, raggiungendo però un compromesso: la separazione della cattura dell'evento di selezione dall'effettivo comportamento dell'oggetto, implementandoli in due script differenti. Il primo, che cambia da piattaforma a piattaforma, richiama il secondo dopo aver catturato l'evento di selezione, il quale rappresenta il vero comportamento dell'oggetto ed è indipendente dalla piattaforma in uso. In questo modo è stato

possibile creare dei *prefabs* (oggetti portabili all'interno dell'ambiente *Unity* contenenti le informazioni di forma, colore, script agganciati eccetera) che contenessero solo le informazioni del comportamento dell'oggetto dopo la cattura della selezione.

### 3.2.2 Usabilità dell'applicazione

La fase di acquisto inizialmente prevista comprendeva l'immissione dei propri dati di pagamento all'interno dell'ambiente *VR<sub>G</sub>* tramite una tastiera tridimensionale posta davanti all'utente, dove ogni tasto era selezionabile attraverso l'interfaccia fisica del visore. Dopo la sperimentazione di tale tastiera, abbiamo constatato la sua scarsa usabilità, causando un processo d'acquisto lungo e tedioso. Abbiamo dunque optato per una soluzione più semplice: l'utente prima di entrare nella scena *VR<sub>G</sub>* viene invitato a immettere i propri dati di login attraverso la normale tastiera del telefono, così da accedere al proprio account personale. Tale account deve essere precedentemente creato nel sito/e-commerce dedicato, dove possono essere immessi anche i dati di pagamento. L'implementazione di tale sito esce dagli obiettivi di questo progetto, dunque non mi sono occupato dello sviluppo di quest'ultimo. Una volta autenticato, lo *smartphone* passa in modalità *VR<sub>G</sub>*, invitando l'utente ad indossare il visore. All'interno dell'ambiente, all'utente non viene più richiesto di immettere dati, così da poter sperimentare in tutta serenità l'esperienza virtuale.

Questa soluzione però ha mostrato al *team* la profonda differenza tra le piattaforme *Samsung Gear VR* e *Google Cardboard*. Ogni applicazione *Samsung Gear VR*, una volta avviata, richiede all'utente di inserire lo *smartphone* all'interno del visore prima di poter effettivamente essere utilizzata. Questo obbliga lo sviluppatore a prevedere tutte scene di tipo *VR<sub>G</sub>* durante lo sviluppo dell'applicazione, contrariamente a *Google Cardboard* che permette la coesistenza di tutte le tipologie scenografiche: 2D, 3D e *VR<sub>G</sub>*. Dunque, non è stato possibile inserire una scena 2D iniziale di login per la piattaforma Samsung e, a causa dei tempi ridotti di stage, non ho potuto sviluppare una soluzione a tale problema.



**figura 3.7:** Schermata 2D di login

### 3.2.3 Costruzione della scena 3D

Dopo un'attenta fase di ricerca iniziale sulla tecnologia *Unity* e *VR<sub>G</sub>*, ho potuto constatare che vi sono due modi principali per costruire una scena 3D:

- \* **Tramite modellazione 3D dell'intero ambiente virtuale;**
- \* **Tramite l'applicazione di una foto a 360 gradi di una stanza ad una sfera o cubo inversi.**

La prima soluzione permette una vera esperienza 3D, dando la possibilità all'utente di spostarsi nelle tre dimensioni, prevedendo l'utilizzo di un gamepad. Ogni oggetto è ben definito in un punto dello spazio tridimensionale e osservabile da tutte le angolazioni. Purtroppo però per costruire un ambiente 3D di qualità accettabile sono necessarie profonde conoscenze di modellazione, conoscenze che fuoriescono dall'obbiettivo di questo stage.

La seconda soluzione non permette una totale esperienza 3D, poiché applica una foto 2D a 360 gradi ad una sfera o un cubo inversi creati in *Unity*. L'effetto così creato simula la tridimensionalità, non permettendo all'utente lo spostamento tra gli oggetti ma solo una visione stereoscopica della stanza. Il livello qualitativo raggiunto però è ottimo, se la foto viene realizzata con le giuste apparecchiature. Abbiamo deciso dunque di utilizzare la seconda metodologia per la creazione dell'ambiente.

Abbiamo così realizzato la foto a 360 gradi utilizzando una macchinetta fotografica di alta qualità presente nell'azienda seguendo la guida presente nel sito lightspacewater<sup>2</sup>. Posizionata la macchinetta fotografica sul cavalletto al centro di una stanza, appositamente arredata allo scopo nell'azienda Diana Corp., abbiamo effettuato 3 foto, una sovraesposta, una sottoesposta e una ad esposizione normale, ogni 45 gradi orizzontalmente. Abbiamo poi ripetuto l'operazione inclinando la macchinetta 45 gradi verso il basso ed infine 45 gradi verso l'alto.

Le foto così ottenute, ci hanno permesso di creare la foto a 360 gradi tramite il software Autopano<sup>3</sup>, che l'ha creata automaticamente a meno di qualche punto di agganciamento inserito manualmente. Infine abbiamo applicato la foto come texture di una sfera inversa creata tramite l'editor di *Unity*.

---

<sup>2</sup><http://www.lightspacewater.net/Tutorials/PhotoPano2/paper/>

<sup>3</sup><http://www.kolor.com/autopano/>

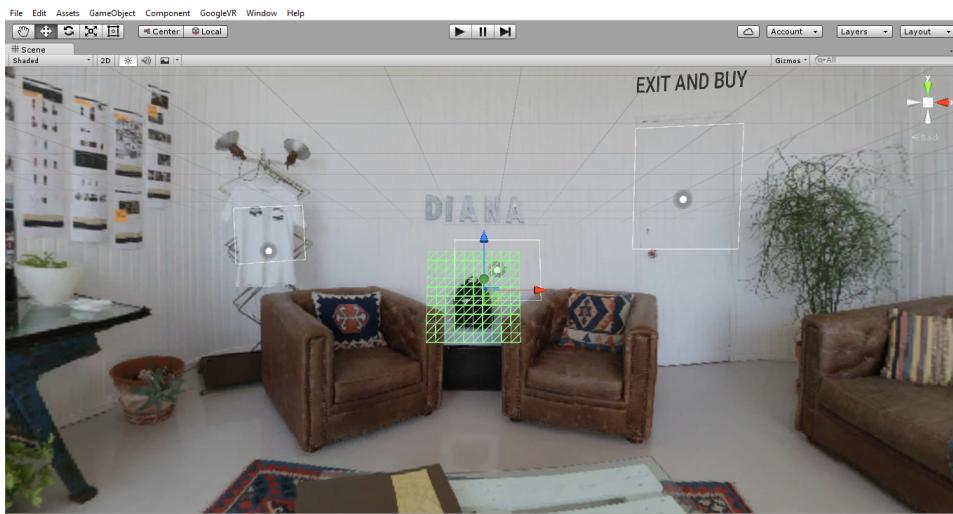


**figura 3.8:** Sfera inversa a cui è stata applicata la foto 360 della stanza come texture

### 3.2.4 Interazione con gli oggetti all'interno della scena

All'interno della scena creata tramite la foto a 360 gradi, gli oggetti presenti nella stanza non sono veri oggetti tridimensionali modellati in *Unity*. A quest'ultimi, come descritto nella sezione 3.5.1, è possibile applicare la proprietà *Mesh Collider*, per renderli tangibili, e lo script che cattura l'evento di selezione. Purtroppo nella soluzione da noi adottata gli oggetti fanno parte della foto che crea lo sfondo, perciò non è possibile applicargli queste due proprietà.

Abbiamo così deciso di porre davanti ad ogni oggetto un pannello tridimensionale creato tramite l'editor di *Unity* e ad esso agganciarigli le caratteristiche citate. In più al pannello viene disabilitata la proprietà *Mesh Renderer*, proprietà che lo rende visibile. In questo modo viene simulata l'interattività dell'oggetto che, alla selezione tramite l'interfaccia fisica del visore, attiva il pannello informativo relativo all'oggetto selezionato.



**figura 3.9:** Pannello tridimensionale posto davanti ad un oggetto presente nella stanza per renderlo interattivo

### 3.2.5 Progettazione e integrazione con AWS API Gateway

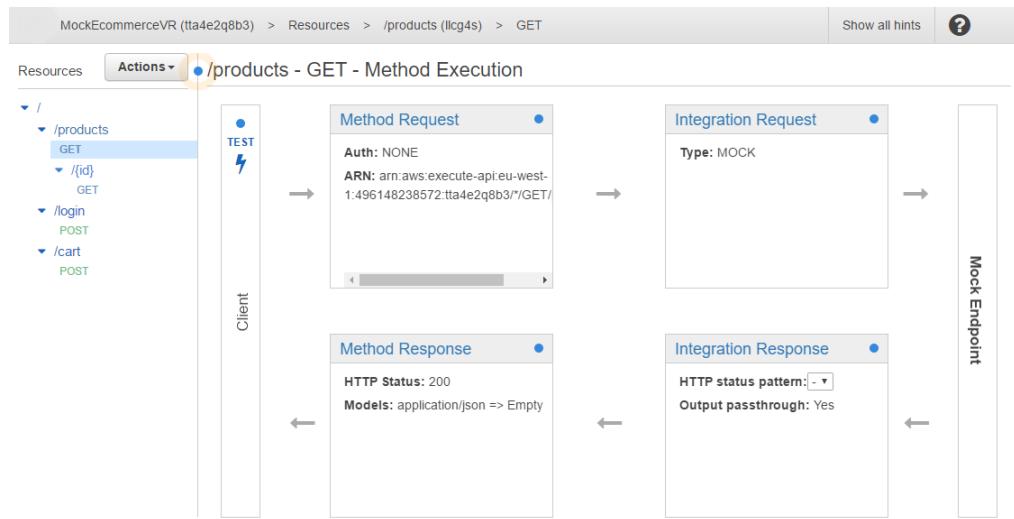
Uno degli obiettivi dello stage era sperimentare la comunicazione tra l'applicazione creata con *Unity* e sistemi esterni tramite protocollo HTTP, come ad esempio un *e-commerce*. Per testare tali funzionalità in sicurezza, senza dover creare o modificare gli *e-commerce* aziendali, il tutor mi ha suggerito di utilizzare il servizio *API Gateway* di *Amazon Web Service*.

Amazon API Gateway è un servizio completamente gestito che semplifica agli sviluppatori la creazione, la pubblicazione, la manutenzione, il monitoraggio e la protezione delle API su qualsiasi scala. Essa agisce come porta d'entrata attraverso la quale le applicazioni possono accedere a dati, logica di business o funzionalità dei servizi di back-end. Gestisce tutte le attività di accettazione ed elaborazione relative alle chiamate API simultanee, inclusi gestione del traffico, controllo di accessi e autorizzazioni, monitoraggio e gestione delle versioni delle API. Infine permette la creazione di *API Mock*, ovvero API che non si agganciano ad alcun back-end ma che rispondono messaggi prefissati ad ogni chiamata HTTP. Quest'ultima funzionalità ci ha spinto ad utilizzare questo servizio, dato che lo sviluppo di un vero back-end non rientrava all'interno degli obiettivi di stage.

Ho creato dunque tre risorse e al loro interno le relative chiamate HTTP:

- \* **/login:** risorsa riguardante le operazioni di autenticazione. Contiene la chiamata:
  - POST: riceve un file JSON contenente l'username e la password immesse dall'utente e restituisce un *token* di autenticazione.
- \* **/products:** risorsa riguardante le operazioni sui prodotti. Contiene la chiamata:
  - GET: ritorna un file JSON contenente le informazioni riguardanti tutti i prodotti all'interno della scena come nome, descrizione e foto.
- \* **/products/*id*:** risorsa riguardante le operazioni su un singolo prodotto. Contiene la chiamata:

- GET: ritorna un file JSON contenente le informazioni su un singolo prodotto in base all'id passato.
- \* **/cart**: risorsa riguardante le operazioni sul carrello. Contiene le chiamate:
  - POST: riceve un file JSON contenente i prodotti presenti all'interno del carrello.



**figura 3.10:** Flusso della chiamata GET all'interno della risorsa */products* in AWS API Gateway

Per quanto riguarda *Unity*, invece, ho sfruttato la classe *WWW* che consente di effettuare chiamate GET per recuperare dati JSON o foto:

```

1 // Endpoint GET creato tramite API Gateway che restituisce un file
2 // JSON contenente le informazioni sui prodotti
3 string url = "https://execute-api.amazonaws.com/prod/products";
4
5 // Costruzione dell'oggetto WWW e passaggio dell'endpoint come
6 // singolo parametro
7 WWW www = new WWW(url);
8 yield return www;
9
10 // Recupero delle informazioni
11 string info = www.text;

```

e chiamate POST, potendo inviare *headers* e corpo:

```

1 // Endpoint POST creato tramite API Gateway che riceve i dati
2 // di login e restituisce un token di autenticazione
3 string url = "https://execute-api.amazonaws.com/prod/login";
4
5 // Creazione del file JSON
6 string JSON = "{\"username\":\"" + "\"" + Username + "\","
7     + "\"password\":\"" + "\"" + Password + "\"}";

```

```

8 byte[] body = System.Text.Encoding.UTF8.GetBytes(JSON);
9
10 // Creazione degli headers necessari per il riconoscimento
11 // del file JSON
12 Dictionary<string, string> headers =
13     new Dictionary<string, string>();
14 headers.Add("Content-Type", "application/json");
15 headers.Add("X-HTTP-Method-Override", "POST");
16
17 // Costruzione dell'oggetto WWW e passaggio dell'endpoint come
18 // singolo parametro
19 WWW www = new WWW(url, body, headers);
20 yield return www;

```

---

### 3.3 Sviluppo

Questa sezione descrive nel dettaglio il personale sviluppo delle più importanti funzionalità dell'applicazione attraverso l'editor grafico e il linguaggio di scripting C# offerto dal framework [Unity](#).

#### 3.3.1 Dati persistenti attraverso le scene

L'applicazione sviluppata possiede due scene: la prima offre una schermata di login dove l'utente può digitare la propria *username* e *password*; la seconda permette la visualizzazione dell'ambiente virtuale dopo aver indossato il visore. *Unity* prevede la possibilità di creare più scene all'interno dello stesso progetto e il passaggio tra queste è gestibile attraverso script che caricano la scena successiva dopo aver catturato un particolare evento. Nel mio specifico caso, dopo che l'utente ha digitato l'*username* e la *password* e selezionato il pulsante di login, viene effettuata la chiamata POST all'API contenente le due informazioni digitate. Se la chiamata va a buon fine, viene restituito un *token* di autenticazione e viene così caricata la scena *VRG*.

Il *token* di autenticazione serve all'utente in fase di acquisto dei prodotti, il quale permette al sistema di riconoscere quest'ultimo associandogli i dati di pagamento immessi in fase di registrazione. Purtroppo però, di default, *Unity* distrugge ogni oggetto presente all'interno di una scena prima di passare a quella successiva. Questo comportamento non è accettabile per il *token* di autenticazione, il quale deve vivere fino alla fase di acquisto. Ho creato dunque un oggetto generico, o *Empty Object*, all'interno della prima scena di autenticazione e attraverso il metodo *DontDestroyOnLoad* (), offerto da *Unity*, l'ho reso persistente:

```

1 public class UserSessionScript : MonoBehaviour {
2
3     private string token;
4
5     // Il metodo Awake() deriva dalla classe MonoBehaviour
6     // e al suo interno e' possibile impostare
7     // tutte le caratteristiche iniziali che l'oggetto
8     // deve avere prima che la scena abbia inizio
9     void Awake () {
10         DontDestroyOnLoad (this);
11     }

```

```

12
13     public string getToken() {
14         return token;
15     }
16
17     public void setToken(string token) {
18         this.token = token
19     }
20 }
```

---

In questo modo, l'oggetto *UserSession* sarà presente nella scena successiva e si potrà così recuperare le informazioni che contiene. Questa operazione, però, necessita di un'altro script di *utilities* per recuperare l'oggetto a runtime all'interno di una scena nella quale prima non era presente:

```

1 public class UserSessionUtils : MonoBehaviour {
2
3     // Il metodo Find() offerto da Unity permette
4     // di recuperare il riferimento di un oggetto
5     // all'interno della scena passandogli il nome
6     // come parametro
7     public string getUserToken() {
8         GameObject userSession = GameObject.Find("UserSession");
9         return userSession.GetComponent<UserSessionScript>().getToken();
10    }
11
12    public void setUserToken(string token) {
13        GameObject userSession = GameObject.Find("UserSession");
14        userSession.GetComponent<UserSessionScript>().setToken(token);
15    }
16 }
```

---

Dunque, una volta recuperato il *token* di autenticazione dalla chiamata POST, è possibile memorizzarlo all'interno dell'oggetto *UserSession* tramite lo script *UserSessionUtilities* e caricare così la scena successiva:

```

1     WWW www = new WWW(url, body, headers);
2     yield return www;
3
4     // JsonData e' una classe che effettua il
5     // parsing degli oggetti JSON in Unity. In questo
6     // caso recupera il token di autenticazione
7     JsonData jsonvale = JsonMapper.ToObject(www.text);
8     string username = jsonvale ["token"].ToString ();
9
10    UserSessionUtils usu = new UserSessionUtils ();
11    usu.setUserToken (token);
12
13    // Metodo che carica la scena denominata 1
14    // in fase di compilazione
15    SceneManager.LoadScene (1);
```

---

### 3.3.2 Il pannello informativo

*Unity* permette la creazione di due principali tipologie di oggetti all'interno di una scena:

- \* **3D Object:** oggetti sviluppati nelle tre dimensioni che possono contenere le informazioni di forma, colore, tangibilità, rigidità, comportamento eccetera. Un esempio di questi oggetti è rappresentato dal cubo, dalla sfera o dal cilindro. Questi elementi possono essere modificati e assemblati per creare ambienti e oggetti sempre più complessi;
- \* **UI:** oggetti 2D che compongono l'interfaccia utente, ovvero pannelli dove è possibile scrivere e scorrere un testo, visualizzare un'immagine, selezionare pulsanti eccetera.

Per la creazione del pannello informativo ho fatto ampio utilizzo di questa seconda tipologia di oggetti, trovandola adatta allo scopo.

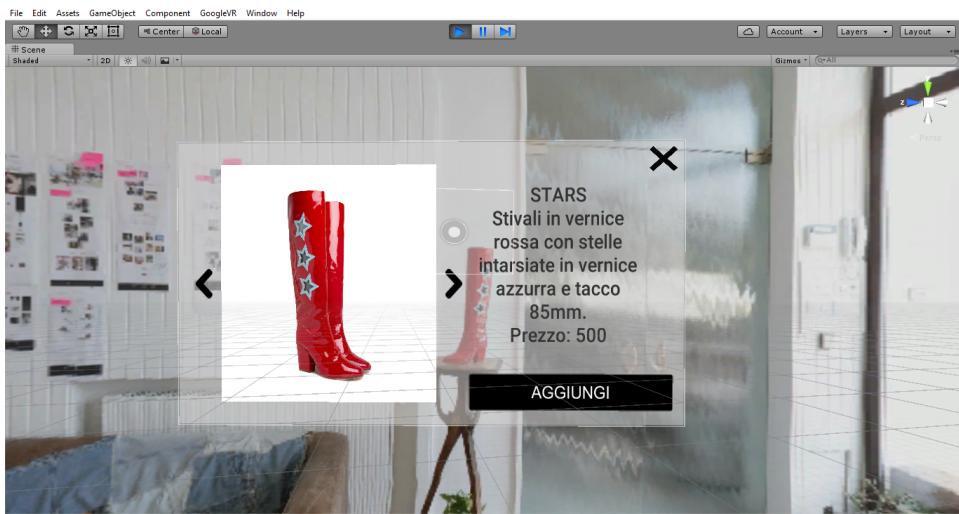
Per costruire un'interfaccia utente in *Unity* è necessario innanzitutto la presenza dell'oggetto *EventSystem* all'interno della scena. Tale oggetto permette la gestione degli eventi, catturando i vari input provenienti da qualsiasi terminale come mouse, tastiera, gamepad o, in questo caso, dispositivo *VR<sub>G</sub>*. Sia Samsung che Google offrono una loro versione di tale oggetto, dato che i possibili input di *Samsung Gear VR* sono più numerosi rispetto a quelli di *Google Cardboard*.

Per poter essere visibili e interattivi, ogni oggetto UI deve essere figlio di un oggetto *Canvas*. Il *Canvas*, di default, è visibile come un pannello opaco all'interno della scena e serve sostanzialmente a raggruppare gli oggetti UI per farli comunicare con l'*EventSystem*.

Vi sono tre tipologie di *Canvas*:

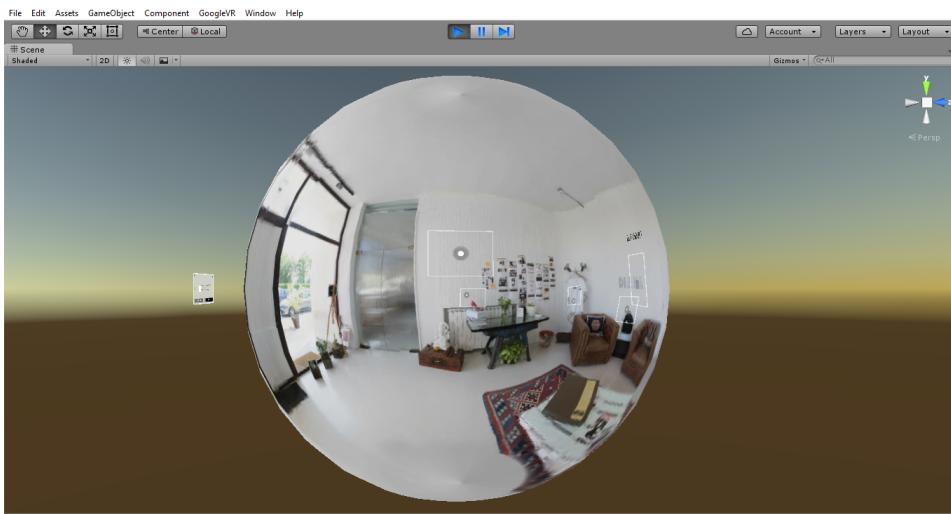
- \* **Screen Space - Overlay:** questa modalità di *rendering* pone gli elementi dell'interfaccia utente fissi all'interno della scena e davanti alla camera dell'utente. Se lo schermo viene ridimensionato o viene cambiata la risoluzione, il *Canvas* cambierà automaticamente dimensioni per adattarsi;
- \* **Screen Space - Camera:** questa modalità di *rendering* è molto simile alla *Screen Space - Overlay*, con l'unica differenza che è possibile scegliere la distanza e l'inclinazione degli oggetti UI rispetto alla camera;
- \* **World Space:** questa modalità di *rendering* è profondamente differente dalle prime due poiché premette di visualizzare gli oggetti UI in un qualsiasi punto dello spazio tridimensionale, potendoli riposizionare a runtime.

Data la varietà di punti interattivi all'interno della stanza tridimensionale, ho ritenuto necessario l'utilizzo della modalità *World Space*, poiché desideravo che il pannello si potesse posizionare davanti ad ogni oggetto interattivo alla selezione di quest'ultimo.



**figura 3.11:** Pannello informativo attivato dopo la selezione di un particolare oggetto all'interno della scena

Per quanto riguarda il suo spostamento nello spazio tridimensionale, inizialmente avevo previsto come punto di partenza il centro della sfera, disattivando ovviamente la proprietà *Mesh Renderer* che ne permette la visualizzazione. Alla selezione di un oggetto presente nella stanza, lo script che gestisce il pannello calcola il punto medio tra l'oggetto e l'osservatore e vi si posiziona, mostrando le informazioni. Purtroppo però, gli oggetti UI non possiedono tale proprietà e se si desidera renderli invisibili è necessario disattivarli del tutto tramite il metodo `SetActive()`. Questo pone un nuovo problema: non è possibile riferirsi ad un oggetto disattivato per riattivarlo a runtime. La soluzione finale trovata e discussa col *team* di sviluppatori, non è sicuramente ottimale ma è funzionale. Tale problema rischiava di non farmi rispettare il piano di lavoro concordato e c'erano ancora molti punti che necessitavano di essere esplorati e sperimentati. Abbiamo quindi deciso che il punto d'origine e di ritorno del pannello sarebbe stato esterno alla sfera. In questo modo esso sarebbe rimasto sempre attivo fin dall'inizio ma invisibile poiché esterno alla stanza tridimensionale.



**figura 3.12:** Pannello informativo attivo all'esterno della sfera

### 3.3.3 Creazione e parsing di oggetti JSON in Unity

Unity offre al programmatore la classe `JsonUtility` che permette la trasformazione di un oggetto in una stringa JSON:

---

```

1 MyClass myObject = new MyClass();
2 myObject.level = 1;
3 myObject.timeElapsed = 47.5f;
4 myObject.playerName = "Dr\u00a9Charles\u00a9Francis";
5
6 string json = JsonUtility.ToJson(myObject);
// la stringa json e' uguale a:
// {"level":1,"timeElapsed":47.5,"playerName":"Dr Charles Francis"}

```

---

e viceversa:

---

```

1 string json = {"level":1,"timeElapsed":47.5,
2           "playerName":"Dr\u00a9Charles\u00a9Francis"};
3 MyClass myObject = JsonUtility.FromJson<MyClass>(json);

```

---

Purtroppo però, non viene offerta alcuna funzionalità di parsing di file JSON, a me necessaria per leggere le informazioni dei prodotti recuperate tramite la chiamata GET all'API. Dopo alcune ricerche, ho scoperto l'esistenza della libreria `LitJSON`<sup>4</sup>, libreria scritta in C# e compatibile con tutti i linguaggi .Net. Essa, dopo aver importato il file `LitJSON.dll` all'interno del progetto Unity, permette di leggere il file JSON scaricato come se fosse un semplice array, accedendo direttamente ai vari campi tramite le doppie parentesi quadre:

---

```

1 // Chiamata GET per il recupero delle infomrazioni di un prodotto
2 www = new WWW(url);
3 yield return www;
4

```

---

<sup>4</sup><https://lbv.github.io/litjson/>

```

5 // Crea un parser per il file JSON scaricato
6 jsonvale = JsonMapper.ToObject(www.text);
7
8 id = jsonvale ["id"].ToString ();
9
10 name = jsonvale ["name"].ToString ();
11
12 description = jsonvale ["description"].ToString ();
13
14 price = jsonvale ["price"].ToString ();
15
16 text = name + "\n" + description + "\n" + "Prezzo: " + price;
17
18 // Il file JSON contiene i link per scaricare le immagini
19 www_img1 = new WWW (jsonvale ["img1"].ToString ());
20 yield return www_img1;
21 imageArray [0] = www_img1.texture;
22
23 www_img2 = new WWW (jsonvale ["img2"].ToString ());
24 yield return www_img2;
25 imageArray [1] = www_img2.texture;
26
27 www_img3 = new WWW (jsonvale ["img3"].ToString ());
28 yield return www_img3;
29 imageArray [2] = www_img3.texture;

```

---

### 3.3.4 Creazione a runtime di oggetti interattivi

Per la realizzazione del carrello interattivo è stato necessario uno studio approfondito sulla creazione di oggetti scenici a runtime in *Unity*. Questo perché inizialmente il carrello deve risultare vuoto e riempirsi a mano a mano che l'utente decide di aggiungere un prodotto, ovvero mentre la scena è in *play mode*.

La soluzione adottata prevede la creazione, tramite l'editor grafico, di un pannello UI generico contenente i componenti per la foto, per il nome e per il prezzo dell'ipotetico prodotto che sarà aggiunto al carrello. All'oggetto grafico così creato, viene agganciato lo script *ListItemController* contenente i campi logici del pannello che verranno riempiti con le informazioni dello specifico prodotto:

```

1 public class ListItemController : MonoBehaviour {
2     private string id;
3
4     private Text description;
5
6     private Image icon;
7
8     public string getId() {
9         return id;
10    }
11
12    public void setId(string id) {
13        this.id = id;
14    }
15

```

```

16     public Text getDescription() {
17         return description;
18     }
19
20     public void setDescription(Text description){
21         this.description = description;
22     }
23
24     public Image getIcon() {
25         return icon;
26     }
27
28     public void setIcon(Image icon){
29         this.icon = icon;
30     }
31 }
```

---

Il pannello grafico e lo script agganciato vengono uniti in un unico pacchetto per formare un *prefab* di nome *ListItem*, ovvero un oggetto prefabbricato pronto per essere istanziato all'interno della scena. Così, lo script incaricato alla gestione del carrello grafico, ovvero *ListController*, dopo aver controllato tutti i prodotti presenti nel carrello logico, gestito dallo script *ShoppingBagScript*, instanzia tanti *ListItem* quanti sono i prodotti presenti nel carrello logico e li posiziona in lista all'interno del pannello del carrello:

```

1 public void printItem(Item[] items) {
2     // Per ogni oggetto presente nel pannello logico
3     // che non e' stato ancora stampato nel carrello
4     while (alreadyItemPrint <
5             shoppingBag.GetComponent<ShoppingBagScript> ().getCounter ()) {
6
7         // Istanzia l'oggetto prefab all'interno della scena
8         GameObject newItem =
9             Instantiate (ListItemPrefab) as GameObject;
10        ListController controller =
11            newItem.GetComponent<ListController> ();
12
13        // Recupera le informazioni del prodotto logico per
14        // inizializzare il pannello grafico del prodotto
15        controller.setId(items [alreadyItemPrint].getId ());
16        controller.Description.text =
17            items [alreadyItemPrint].getInfo ();
18        controller.Icon.sprite =
19            Sprite.Create (items [alreadyItemPrint].getImg (),
20            new Rect (0, 0, items [alreadyItemPrint].getImg ().width,
21            items [alreadyItemPrint].getImg ().height), new Vector2 (0.5f, 0.5f));
22
23        // Posiziona ogni prodotto in lista all'interno
24        // del pannello del carrello
25        newItem.transform.SetParent (ContentPanel.transform);
26        newItem.transform.position += new Vector3 (0, 0, 32);
27        newItem.transform.rotation = ContentPanel.transform.rotation;
28        newItem.transform.localScale = Vector3.one;
29
30        // Se l'array dei prodotti e' pieno
```

```
31     // effettua il resize
32     if (alreadyItemPrint == ItemPrefabs.Length) {
33         GameObject[] newItemPrefabs =
34             new GameObject [ItemPrefabs.Length * 2];
35         for (int i = 0; i < ItemPrefabs.Length; i++) {
36             newItemPrefabs [i] = ItemPrefabs [i];
37         }
38         ItemPrefabs = newItemPrefabs;
39     }
40     ItemPrefabs [alreadyItemPrint] = newItem;
41     alreadyItemPrint++;
42 }
43 }
```

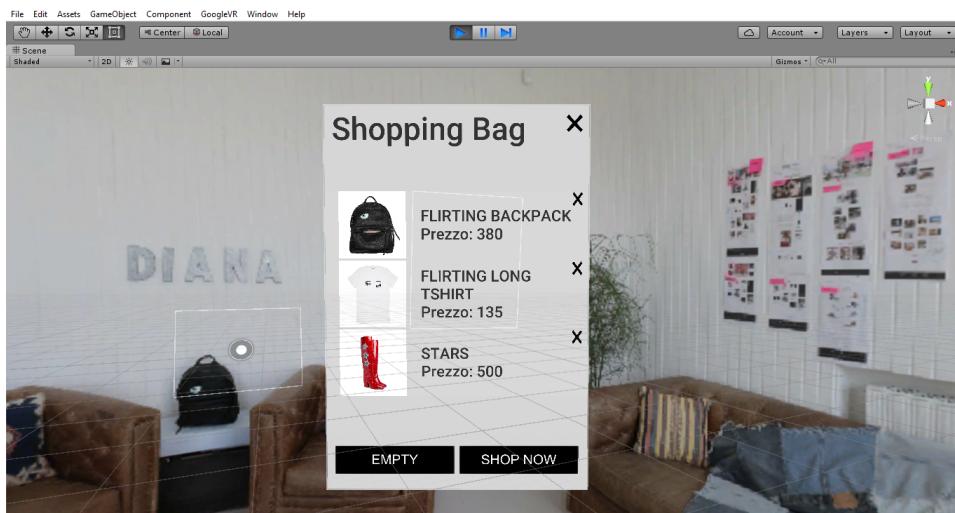


figura 3.13: Carrello riempito dei prodotti presenti nella stanza

# **Capitolo 4**

## **Analisi retrospettiva**

In questo capitolo analizzerò i risultati ottenuti confrontandoli con gli obiettivi prefissati, le conoscenze acquisite e quali tra queste ritengo debbano essere integrate nel corso di laurea.

### **4.1 Bilancio dei risultati rispetto agli obiettivi prefissati**

In questa sezione analizzerò i risultati ottenuti e li confronterò con gli obiettivi che l'azienda si era prefissata di ottenere.

### **4.2 Bilancio formativo**

In questa sezione analizzerò le conoscenze, le abilità e le competenze apprese durante l'attività di stage.

### **4.3 Analisi critica del rapporto formativo tra stage e corso di laurea**

In questa sezione discuterò quali conoscenze, apprese durante lo stage, ritengo debbano essere integrate nel corso di laurea.

### **4.4 Valutazioni personali**

In questa sezione effettuerò delle valutazioni personali riguardo al progetto e allo stage.



# Glossario

## A

- \* **Authoring:** gli applicativi d'autore sono quei software verticali che consentono la realizzazione di una comunicazione multimediale, articolata e riproducibile su personal computer. L'intento è quello di poter produrre e veicolare contenuti (immagini statiche, animazioni grafiche, filmati video, commenti sonori, effetti audio e altro) su supporti come CD-ROM, DVD, via web, ma anche attraverso un circuito, chiuso o aperto, di display distribuiti e connessi tra loro in rete.
- \* **Augment reality:** arricchimento della percezione sensoriale umana mediante informazioni, in genere manipolate e convogliate elettronicamente, che non sarebbero percepibili con i cinque sensi.

## B

- \* **Bug tracking:** applicativo software usato generalmente dai programmatori per tenere traccia delle segnalazioni di bug all'interno dei software, in modo che tali errori siano mantenuti sotto controllo, con una descrizione della riproducibilità e dei dettagli ad essi correlati.

## C

- \* **Command-and-control:** principio di *management* dove si afferma il mantenimento dell'autorità in un processo decisionale distribuito.
- \* **Content management system:** è uno strumento software, installato su un server web, il cui compito è facilitare la gestione dei contenuti di siti web, svincolando il webmaster da conoscenze tecniche specifiche di programmazione web.
- \* **Container software:** soluzione al problema di come ottenere il software eseguibile in modo affidabile quando viene spostato da un ambiente informatico all'altro. Sono costituiti da un intero ambiente di runtime: un'applicazione, oltre a tutte le sue dipendenze, librerie e altri file binari e file di configurazione necessari per eseguirlo, impacchettati in un unico pacchetto.

**D**

- \* **Deployment:** consegna o rilascio al cliente, con relativa installazione e messa in funzione o esercizio, di una applicazione o di un sistema software tipicamente all'interno di un sistema informatico aziendale.

**E****F**

- \* **Framework:** architettura logica di supporto (spesso un'implementazione logica di un *design pattern*) su cui un software può essere progettato e realizzato.

**G****H****I**

- \* **Issue tracking:** pacchetto software che gestisce e mantiene liste di problemi im maniera organizzata.

**J****K****L****M****N****O**

- \* **Operations:** funzioni di un'impresa coinvolte nella messa a disposizione per il cliente di un determinato prodotto o servizio.

**P****Q****R**

- \* **Retailing:** o vendita al dettaglio, costituisce l'ultimo anello della catena di distribuzione. Il venditore al dettaglio (negozi, supermercato, eccetera) acquista

quantità, relativamente elevate, di merce dal produttore o da un grossista e rivende quantità più contenute ai consumatori per ottenere un profitto.

- \* **Refactoring:** tecnica strutturata per modificare la struttura interna di porzioni di codice senza modificarne il comportamento esterno, applicata per migliorare alcune caratteristiche non funzionanti del software.
- \* **Repository:** è un ambiente di un sistema informativo, in cui vengono gestiti i metadati, attraverso tabelle relazionali; l'insieme di tabelle, regole e motori di calcolo tramite cui si gestiscono i metadati prende il nome di metabase.

## S

- \* **Software development kit:** insieme di strumenti per lo sviluppo e la documentazione di software.
- \* **Social management system:** software che permette la gestione dei propri social network, collezionando contenuti presenti in essi o interagendovi automaticamente effettuando operazioni desiderate.
- \* **Stakeholder:** soggetto direttamente o indirettamente coinvolto in un progetto o in un'attività di un'azienda.

## T

- \* **Testing:** il *software testing* è un'attività di investigazione condotta per fornire alle parti interessate informazioni sulla qualità del prodotto o del servizio in prova.

## U

- \* **UML:** è un linguaggio di modellazione e specifica basato sul paradigma orientato agli oggetti. UML consente di costruire modelli object-oriented per rappresentare domini di diverso genere. Nel contesto dell'ingegneria del software, viene usato soprattutto per descrivere il dominio applicativo di un sistema software e/o il comportamento e la struttura del sistema stesso. Il modello è strutturato secondo un insieme di viste che rappresentano diversi aspetti della cosa modellata (funzionamento, struttura, comportamento, e così via), sia a scopo di analisi che di progetto, mantenendo la tracciabilità dei concetti impiegati nelle diverse viste. Oltre che per la modellazione di sistemi software, UML viene non di rado impiegato per descrivere domini di altri tipi, come sistemi hardware, strutture organizzative aziendali, processi di business.
- \* **Unit testing:** per test di unità si intende l'attività di *testing* di singole unità software. Per unità si intende il minimo componente di un programma dotato di funzionamento autonomo.

**V**

- \* **Virtual reality:** realtà simulata attraverso dispositivi elettronici, come visori, cuffie e sensori di movimento.

**W****X****Y****Z**

# Bibliografia