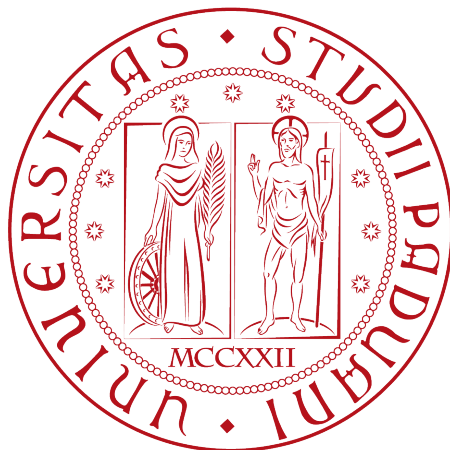


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA



**Sviluppo di una piattaforma web per la gestione
di contenuti multimediali volti alla fruizione in
realtà aumentata su app mobile**

Tesi di laurea triennale

Relatore

Prof. Claudio Enrico Palazzi

Laureando

Andrea Ballista Flandoli

ANNO ACCADEMICO 2014-2015

Andrea Ballista Flandoli: *Sviluppo di una piattaforma web per la gestione di contenuti multimediali volti alla fruizione in realtà aumentata su app mobile*, Tesi di laurea triennale, © Aprile 2015.

Dedicato alla famiglia

Sommario

Il presente documento tratta il dettaglio dello stage svolto dal laureando Andrea Ballista Flandoli presso l'azienda Experenti S.r.l.. Tale stage, della durata di 320 ore, aveva come finalità primaria la realizzazione di una piattaforma *web* atta alla gestione di contenuti multimediali, con sviluppo di *API ad hoc* per il recupero dei dati immessi nella piattaforma.

“I computer sono inutili. Ti sanno dare solo risposte.”

— Pablo Picasso

Ringraziamenti

Desidero, innanzitutto, ringraziare i miei genitori e mia sorella per avermi sostenuto ed aiutato in ogni momento, anche in quelli più difficili.

Un ringraziamento particolare va a Daniela, che con la semplicità di un sorriso mi ha dato la forza per affrontare ogni prova.

Un sentito ringraziamento va anche a tutti i miei amici e compagni di università, per avermi supportato (e sopportato) per tutta la durata di questo percorso di studi.

Desidero ringraziare dal profondo del cuore anche tutto lo staff di Experenti, che fin dal primo giorno di stage mi ha fatto sentire come se fossi arrivato in una seconda famiglia.

Infine, vorrei esprimere la mia gratitudine al Prof. Claudio Enrico Palazzi, relatore della mia tesi, per l'aiuto fornitomi relativamente alla stesura di questo documento.

Padova, Aprile 2015

Andrea Ballista Flandoli

Indice

1	Introduzione	1
1.1	Presentazione dell'azienda	1
1.2	Core business	1
1.2.1	Prodotti e servizi offerti	2
1.3	Processi aziendali	4
1.3.1	Modello agile	4
1.3.2	Strumenti a supporto dei processi	5
1.4	Organizzazione del testo	7
2	Il progetto	9
2.1	Proposta di stage	9
2.2	Motivazioni	9
2.3	Finalità	10
2.4	Vincoli	11
2.4.1	Vincoli tecnologici	11
2.4.2	Vincoli di metodo	12
2.4.3	Vincoli temporali	12
2.5	Prospettive	12
3	Dettaglio dello stage	15
3.1	Pianificazione di progetto	15
3.2	Dominio applicativo	16
3.2.1	Tecnologie per le app mobile in realtà aumentata	16
3.2.2	Tecnologie per lo sviluppo di applicazioni web	18
3.3	Svolgimento delle attività	21
3.3.1	Progettazione	21
3.3.2	Codifica	26
3.3.3	Verifica e validazione	34
4	Valutazioni retrospettive	37
4.1	Obiettivi raggiunti	37
4.2	Conoscenze acquisite	37
4.3	Valutazione personale	39
4.4	Screenshot finali	40
A	La realtà aumentata	45

B Glossario	47
Bibliografia	53

Elenco delle figure

1.1	Logo di Experenti S.r.l.	1
1.2	Esempio di realtà aumentata interno all' <i>app</i> Experenti	2
1.3	Esempio di realtà aumentata interno all' <i>app</i> Experenti	3
1.4	Logo dell' <i>app</i> Experenti	3
3.1	Diagramma di Gantt relativo alle attività pianificate	15
3.2	<i>Screenshot</i> relativo all'applicazione di lotta rivisitata ed inserita nell' <i>app</i> Experenti	17
3.3	Design pattern MVC applicato al framework Ruby on Rails	19
3.4	Diagramma delle classi del <i>Model</i> di <i>Experenti Self Service</i>	23
3.5	Diagramma vista <i>package</i> delle <i>API</i> di <i>Experenti Self Service</i>	24
3.6	Schermata di login di <i>Experenti Self Service</i>	31
3.7	Schermata di visualizzazione <i>tag</i> con <i>layout</i> a tabelle in <i>Experenti Self Service</i>	33
3.8	Schermata di visualizzazione <i>tag</i> con <i>layout</i> a griglia in <i>Experenti Self Service</i>	33
3.9	Schermata relativa al report di sicurezza generato dalla gemma <i>Brakeman</i>	35
4.1	Schermata di <i>Home</i> di <i>Experenti Self Service</i>	40
4.2	Schermata relativa alle <i>app</i> di un cliente in <i>Experenti Self Service</i>	41
4.3	Schermata relativa all'accesso alle sezioni dei <i>dataset</i> temporanei e ufficiali in <i>Experenti Self Service</i>	41
4.4	Schermata relativa ai <i>dataset</i> che hanno ricevuto conferma in <i>Experenti Self Service</i>	42
4.5	Schermata relativa alle opzioni di un <i>dataset</i> in <i>Experenti Self Service</i>	42
4.6	Schermata relativa ai passi di approvazione di un <i>dataset</i> in <i>Experenti Self Service</i>	43
4.7	Schermata relativa all'inserimento di un video in <i>Experenti Self Service</i>	43
A.1	Esempio di realtà aumentata interno all' <i>app</i> Experenti	45

Elenco delle tabelle

3.1	Tabella relativa al dettaglio ore per le attività pianificate	16
3.2	Tabella relativa ai <i>default routes</i> per la risorsa <i>users</i>	29

Capitolo 1

Introduzione

1.1 Presentazione dell'azienda

Experenti S.r.l. è un'azienda che ha come obiettivo primario l'integrazione di tecnologie innovative quali la realtà aumentata in ottica di *business*. La *startup*, nata nel 2012 come progetto di collaborazione tra l'Università di Padova e Mentis, società di consulenza strategica, nel 2014 è diventata una vera e propria realtà aziendale, espandendosi a tal punto da permettere l'apertura di una filiale a New York.



figura 1.1: Logo di Experenti S.r.l.

La filiale produttiva dell'azienda ha sede presso Massanzago (PD) in Via de Faverei 16, e conta nel suo organico due diversi *team*: uno con competenze relative al contesto commerciale, ed un altro focalizzato sulla parte tecnica e di gestione di progetto.

Va fatta una precisazione: nel momento antecedente allo stage in cui ho redatto il piano di lavoro in collaborazione con il mio tutor aziendale, mi era stato indicato che avrei lavorato presso l'azienda Mentis relativamente al progetto Experenti. Essendo, però, Experenti diventata nel frattempo una realtà aziendale a sé stante, in sede di questa relazione ogni volta che parlerò di “azienda ospitante” sarà in riferimento all'azienda Experenti, in quanto il mio stage si è svolto totalmente all'interno del suo contesto.

1.2 Core business

Il progetto è stato creato con l'obiettivo di sfruttare la tecnologia della realtà aumentata per fornire, attraverso l'integrazione dell'aspetto di coinvolgimento emotivo del *marketing* esperienziale, un servizio pubblicitario che permetta di imprimere nella memoria dei consumatori un determinato prodotto o servizio. Tra i vari *trend* presenti nell'ambito delle *startup*, quello della realtà aumentata è uno tra quelli

che hanno subito il maggiore sviluppo a causa del forte grado di interesse da loro suscitato e della molteplicità degli ambiti di possibile applicazione. Questo è dovuto al fatto che la realtà aumentata consiste nell'arricchimento di quanto viene percepito dalla sfera sensoriale delle persone con informazioni virtuali di qualunque origine e forma, concetto la cui applicabilità risulta utile in un numero pressoché infinito di ambiti. Tali informazioni possono essere fornite tramite l'utilizzo di una moltitudine di dispositivi diversi, quali *smartphone*, *tablet*, *pc* dotati di *webcam* e visori. La realtà aumentata viene spesso confusa con la realtà virtuale, ma la verità è che se ne distingue per una caratteristica fondamentale: nel periodo in cui l'utente accede alle informazioni extra, egli continua ad avere la completa percezione degli altri suoi sensi, godendo quindi di un arricchimento "puro"; la realtà virtuale, invece, fornisce alle informazioni presentate un aspetto primario e dominante, separandole in modo netto dall'ambiente circostante, riducendo quindi la percezione dimensionale dell'utilizzatore finale.

1.2.1 Prodotti e servizi offerti

Nello specifico, ciò che viene offerto ai clienti è la realizzazione di applicazioni *mobile* per la fruizione di contenuti multimediali quali video e modelli 3D in realtà aumentata. Il funzionamento di tali applicazioni prevede che, inquadrando con la fotocamera dello *smartphone* degli elementi chiamati "*tag*" (principalmente immagini o oggetti 3D), sia possibile accedere ai contenuti multimediali desiderati visualizzandoli sullo schermo dello *smartphone*, "aumentando" quindi la realtà con nuovi assi informativi virtuali.



figura 1.2: Esempio di realtà aumentata interno all'app Experenti



figura 1.3: Esempio di realtà aumentata interno all'app Experenti

Per quanto riguarda le modalità di sviluppo delle applicazioni, ai clienti vengono offerte diverse soluzioni in base alle loro necessità:

- **Contenuti interni all'app Experenti:**
i contenuti multimediali richiesti dal cliente vengono inseriti all'interno dell'app Experenti. Tale applicazione, *software mobile* multi-piattaforma disponibile per i sistemi operativi *Android* e *iOS*, funge da macro-contenitore. Al suo interno sono presenti una moltitudine di elementi multimediali di diverse origini (e destinazioni), e la loro fruizione in realtà aumentata avverrà attraverso l'utilizzo di tale app.



figura 1.4: Logo dell'app Experenti

- **Contenuti interni ad un'applicazione *mobile* personalizzata:**
quest'approccio porta allo sviluppo di un'*app* di realtà aumentata con la personalizzazione di elementi quali nome, colori e logo mantenendo, però, la struttura di base dell'applicazione *mobile* Experienci.
- **Contenuti interni ad un'*app* dedicata:**
questa soluzione offre una totale libertà nella personalizzazione dell'applicazione *mobile*, permettendo quindi lo sviluppo di *app* "ad hoc" costruite sulla base delle richieste dei clienti.

1.3 Processi aziendali

1.3.1 Modello agile

Il modello di sviluppo adottato dall'azienda è di tipo *agile*. Tale metodologia, basata sui principi sanciti dall' "*Agile manifesto*", permette una notevole flessibilità nelle attività coinvolte nel ciclo di vita del *software*. In particolare, tra i fondamenti promossi da questo metodo, la prontezza al cambiamento e l'incentivazione al mantenere una collaborazione continua ed una comunicazione in tempo reale con i clienti si rivelano di importanza cruciale nell'azienda, per via del suo settore di sviluppo altamente innovativo e dell'ambiente dinamico in cui esso ha contesto. Grazie a questi accorgimenti, si può apprezzare una minimizzazione del numero di problematiche presentatesi in corso di progetto (e, conseguentemente, un notevole abbassamento del rischio di fallimento).

Il modello *agile* si basa sul concetto di *user story*, ovvero un compito che l'utente ha bisogno di realizzare con il *software* che gli verrà fornito. E' molto meno rigido rispetto agli altri modelli di sviluppo in quanto permette di spaziare liberamente tra le varie fasi di un progetto, ma perché funzioni bene richiede molta esperienza, soprattutto nell'ambito relativo al *project management*.

Lo sviluppo del *software* viene organizzato attraverso delle "iterazioni", ovvero finestre di tempo la cui durata solitamente va da una a quattro settimane. Si rende necessario che ciascuna iterazione contenga quanto utile al rilascio di un incremento per il software in realizzazione, in modo da poter soddisfare sempre più i requisiti imposti dai clienti man mano che esse vengono portate a termine. Alla fine di ogni iterazione viene effettuata una valutazione sulle priorità del progetto in corso, identificando fin da subito gli eventuali cambiamenti necessari. Durante lo sviluppo vengono quindi rilasciate versioni intermedie del *software* per permettere agli *stakeholders* di riscontrare eventuali problemi: in questo modo si ottengono indicazioni più precise sui requisiti da soddisfare, e si possono effettuare degli interventi correttivi già a partire dall'iterazione successiva. Questo viene fatto in un'ottica che non si limita al semplice adempimento al contratto iniziale, ma che punta ad ottenere una totale soddisfazione del cliente rispetto al prodotto realizzato, fornendo un *software* di qualità sviluppato in tempi idealmente brevi.

Nonostante, però, l'azienda riesca ad attenersi alla maggior parte dei metodi imposti dal modello *agile*, per alcuni aspetti legati alla gestione di progetto quali, in primo luogo, la pianificazione, vengono impiegati metodi predittivi e non adattivi, dovuti alla necessità di rispettare le scadenze imposte dai clienti, che molto spesso sono tassative.

Il modello utilizzato all'interno dell'azienda ha come punto focale l'utilizzo di una *Kanban board*, uno strumento di comunicazione visiva che permette di visualizzare e gestire in maniera molto intuitiva il flusso di lavoro. Esso permette, infatti, di:

- tenere sotto costante controllo il flusso di lavoro;
- eseguire un facile spostamento dei *task* tra le diverse categorie;
- monitorare e migliorare il processo di sviluppo;
- verificare che la quantità di *work-in-progress* rimanga al di sotto di una certa soglia prestabilita.

Questo approccio funziona molto bene in Experenti poiché, come detto, le diverse attività facenti parte del ciclo di vita del *software* vengono suddivise in frazioni di tempo limitate, che si adattano particolarmente bene al funzionamento della *Kanban board*. Effettuare un costante monitoraggio sulle attività in corso riduce di molto i rischi legati al non rispetto delle scadenze, in quanto permette una pianificazione adeguata del tempo a disposizione per lo svolgimento dei lavori.

1.3.2 Strumenti a supporto dei processi

Segue una panoramica sui principali strumenti usati con funzione di supporto ai processi aziendali.

ERP

Per la gestione di gran parte dei suoi processi di *business*, l'azienda si appoggia ad **Odoo**, un *software cross-platform* di tipo *ERP*, ovvero *enterprise resource planning*, che per la memorizzazione dei dati utilizza il *DBMS PostgreSQL*.

Questo applicativo, in esecuzione su un server interno all'azienda, offre un grande numero di funzionalità, ma quelle che più vengono usate internamente ad Experenti riguardano:

- gestione della contabilità;
- gestione delle risorse umane;
- gestione dei progetti;
- gestione delle vendite;
- gestione degli acquisti;

- gestione delle campagne di *marketing*.

Inoltre, focalizzandosi sulle funzionalità più utilizzate dal reparto tecnico dell'azienda, va sottolineato che:

- all'interno della gestione dei progetti, è disponibile, tra le tante cose, una *Kanban board* integrata ed estremamente personalizzabile, che si rivela di fondamentale importanza e che viene fortemente usata, nel rispetto della metodologia di sviluppo adottata da Experenti;
- il *software* offre anche strumenti quali calendari e note condivise, che risultano davvero utili durante lo sviluppo.

La presenza di un *software* di questo tipo che permetta di unificare all'interno della stessa piattaforma ed in un'unica *database* i dati relativi a tutti questi aspetti legati al *business* aziendale risulta vantaggiosa, soprattutto rispetto ad altre soluzioni in cui i dati vengono frammentati e suddivisi in ambienti diversi.

Un altro punto di forza di questo *software* è il suo essere estremamente modulare e, conseguentemente grazie al numero di moduli disponibili in rete, facilmente estendibile.

Gestione delle versioni

Per la gestione del versionamento relativo alle *app* sviluppate dall'azienda, Experenti ha deciso di adottare *Subversion*, appoggiandosi al *client* grafico *TortoiseSVN* in quanto la maggior parte del team di sviluppo lavora in ambiente *Windows*. La totalità dei progetti sviluppati dall'azienda vengono posti in *repository* gestiti dal reparto tecnico e collocati in un *server* locale.

Sistemi operativi per lo sviluppo

Come anticipato, gran parte del reparto tecnico opera all'interno dell'ambiente *Microsoft Windows*, nello specifico utilizzando *Windows 8.1*. Una minoranza del team lavora invece all'interno dell'ambiente *MacOS*. Questa differenziazione non comporta problemi in quanto la quasi totalità degli strumenti di sviluppo adottati sono *cross-platform*, e anzi si rivela necessaria in quanto per lo svolgimento di attività quali la compilazione delle *app* per dispositivi con sistema operativo *iOS*, l'utilizzo di *MacOS* è assolutamente necessario.

L'ambiente *Linux*, e nello specifico *Ubuntu*, viene adoperato solamente nel contesto del server interno all'azienda e negli ambienti *cloud* da essa utilizzati, in quanto particolarmente adatto per le funzionalità da esso offerte.

1.4 Organizzazione del testo

- Il **secondo capitolo** fornisce una visione dall'alto del progetto di stage, descrivendone le motivazioni nel contesto aziendale e le finalità da raggiungere;
- Il **terzo capitolo** tratta il dettaglio dello stage, con approfondimenti sul prodotto realizzato e sulle modalità di sviluppo adottate;
- Il **quarto capitolo** descrive le valutazioni retrospettive relative al percorso conclusosi.

Relativamente alla stesura del testo, sono state adottate le seguenti convenzioni tipografiche:

- i termini di particolare rilevanza vengono evidenziati facendo uso del carattere **grassetto**;
- i termini in lingua straniera vengono indicati utilizzando il carattere *corsivo*.

Capitolo 2

Il progetto

2.1 Proposta di stage

Le attività ipotizzate per lo stage prevedevano un periodo iniziale di formazione atto alla familiarizzazione con le tecnologie impiegate all'interno del contesto aziendale nei campi della realtà aumentata e della programmazione di *app mobile* e di piattaforme *web*. Il *core* dello stage consisteva in un contributo attivo da parte dello *stagiaire* nelle attività interne alla progettazione ed alla codifica di una piattaforma *web* per la gestione di contenuti multimediali, chiamata *Experenti Self Service*.

2.2 Motivazioni

Il progetto di stage ha avuto origine grazie ad uno studio operato dal team di ricerca e sviluppo di Experenti il quale, in ottica di miglioramento costante dei processi aziendali, ha proposto l'accrescimento della gamma di servizi offerti attraverso la fornitura di una piattaforma di tipo **SaaS** (*Software as a Service*). Questa piattaforma, denominata **Experenti Self Service**, doveva consistere in un'applicazione *web* avente lo scopo di automatizzare gran parte delle attività aziendali, fornendo al cliente finale uno strumento di accesso diretto ai servizi offerti da Experenti che gli permettesse di gestire autonomamente la realizzazione delle applicazioni *mobile* da lui richieste. La visione finale della strategia aziendale in cui si collocava lo sviluppo della piattaforma prevedeva che l'applicazione racchiudesse un numero di funzionalità sempre maggiore, estendendosi anche agli ambiti legati all'acquisto dei servizi offerti dall'azienda tramite funzionalità di *e-commerce*, e a quello di delega nei confronti dei clienti riguardo alle scelte di personalizzazione delle proprie *app*.

Vista la velocità dei ritmi di realizzazione del *software* imposti dal settore in cui l'azienda opera, una delle criticità che più spesso ci si trova ad affrontare è il verificarsi di problemi nell'acquisizione dei materiali che i clienti devono fornire per lo sviluppo delle applicazioni *mobile* da loro richieste, quali ritardi e/o mancate consegne. In quest'ottica, l'azienda ha predisposto come priorità iniziale per la piattaforma *SaaS* in sviluppo l'implementazione di un sistema di gestione dei contenuti multimediali impiegati nello sviluppo delle *app*: attraverso di esso i clienti dovevano essere in grado di inserire autonomamente i propri contenuti, e di gestire gli eventuali cambiamenti

ad essi legati. Questo comportava inoltre che l'applicazione *web* integrasse delle *API* per comunicare con le *app* sviluppate da Experenti, in modo che i contenuti da mostrare in realtà aumentata si potessero aggiornare ogniqualvolta i clienti operassero delle modifiche su di essi.

Nonostante i costi non banali di realizzazione dovuti alla complessità di sviluppo di una piattaforma di questo tipo, l'azienda punta ad ottenere un grande vantaggio competitivo rispetto ai concorrenti, ed a diminuire fortemente i costi di sviluppo grazie ad un considerevole aumento del tasso percentuale di automazione dei processi interni.

2.3 Finalità

Gli obiettivi dell'attività di stage consistevano, in prima battuta, in un periodo iniziale di formazione atto all'apprendimento di aspetti legati alla realtà aumentata quali gli ambienti di sviluppo, i sistemi operativi *mobile*, gli strumenti *online* per l'elaborazione delle immagini e l'*app* Experenti. L'attenzione si è poi spostata sullo studio del *framework* **Ruby on Rails** per lo sviluppo di applicazioni *web*, con un approfondimento sulle soluzioni per gestirne le interazioni con le *app mobile* e sui diversi possibili *front-end framework* da utilizzare per la gestione dell'interfaccia grafica dell'applicazione *web*, con conseguente selezione del più adatto allo scopo. Al termine del periodo di formazione, le finalità discusse con il mio tutor aziendale prevedevano una mia partecipazione nelle attività di supporto alla progettazione di dettaglio dell'applicazione *web Experenti Self Service* e nell'effettiva codifica e *testing* dei suoi componenti.

Inizialmente tra gli obiettivi minimi stabiliti era previsto che l'applicazione comprendesse anche dei meccanismi di registrazione delle statistiche sui contenuti fruiti in realtà aumentata dagli utenti nel contesto delle *app mobile* in comunicazione con la piattaforma *SaaS*, quali ad esempio tempi di visualizzazione del contenuto e le interazioni effettuate con esso. All'interno degli obiettivi massimi, invece, erano stati inseriti lo studio dei possibili *front-end framework* per la gestione dell'interfaccia grafica dell'applicazione *web* e l'implementazione delle componenti necessarie all'inserimento dei contenuti multimediali tramite *back office*. Nelle prime settimane, però, tali obiettivi sono stati rivisti, in quanto l'azienda si è resa conto che, ai fini interni, la registrazione delle statistiche sull'utilizzo dei contenuti multimediali poteva essere delegata ad un successivo incremento realizzato sulla piattaforma, mentre elementi quali la presenza di un'interfaccia *web* con relativi *View* e *Controller* volta alle funzioni di *back office* per l'inserimento dei contenuti multimediali rappresentava una necessità fin dalla realizzazione del primo prototipo. L'azienda si è inoltre resa conto che, nel contesto dello stage, i tipi di contenuti multimediali che la piattaforma avrebbe dovuto essere in grado di gestire si sarebbero limitati ai video e alle immagini, in quanto l'inserimento di modelli 3D all'interno dell'applicazione *web* per una conseguente fruizione in realtà aumentata necessitava di ulteriori studi da parte del reparto tecnico prima che ne partisse la progettazione.

2.4 Vincoli

Segue una panoramica relativa ai vincoli imposti contestualmente alla realizzazione del *software* richiesto.

2.4.1 Vincoli tecnologici

Il *framework* il cui utilizzo è stato imposto dall'azienda per la realizzazione dell'applicazione *web* è stato *Ruby on Rails*. Questa scelta è dovuta al fatto che questo *framework* risulta particolarmente adatto alla metodologia di sviluppo adottata all'interno dell'azienda. Esso permette:

- una grande capacità di prontezza al cambiamento;
- la costruzione di un'applicazione fortemente basata sull'architettura *Model-View-Controller*;
- di spostarsi molto facilmente all'interno di tre ambienti facilmente configurabili, ovvero
 - *development*
 - *production*
 - *testing*

Questa suddivisione permette una gestione semplificata delle diversificazioni da specificare all'interno di un'applicazione *web* in base all'ambiente in cui si sta lavorando, oltre ad una più facile gestione del *deploy* del *software* in sviluppo;

- di ridurre al massimo il tempo dedicato alla configurazione in quanto si attiene strettamente al principio *Convention over Configuration*;
- un'implementazione molto più rapida rispetto ad altri linguaggi e *framework* concorrenti, grazie alla natura *object-oriented* del linguaggio *Ruby* ed alla vasta quantità di librerie *open-source* disponibili in rete grazie alla *community* a suo supporto;
- di ridurre la quantità di documentazione necessaria in quanto offre meccanismi di *self-documenting*, oltre al fatto che il codice *Ruby* è facilmente leggibile rispetto ad altre alternative;
- di realizzare facilmente architetture *REST* grazie all'enfasi da esso posta relativamente al *RESTful application design*;
- una notevole semplicità per gli sviluppatori di muoversi all'interno di più progetti in quanto, grazie alle convenzioni prestabilite, essi avranno pressoché la stessa struttura.

D'altro canto essendo un *framework* di recente introduzione, ad oggi non è supportato da tutti i meccanismi di *hosting* presenti in rete, ed inoltre è utilizzato in maniera decisamente inferiore rispetto ad alternative come *PHP* e *Java*, i quali presentano una base di sviluppatori più ampia.

L'utilizzo di questo *framework* in ambiente *Windows* ha comportato da parte mia la predisposizione di un ambiente di lavoro, che si è concretizzata in una *shell* dedicata con accesso a:

- un *database MySQL* locale da utilizzare ai fini dello sviluppo;
- un'installazione di *RubyInstaller Development Kit*, modulo necessario per effettuare la *build* e per l'utilizzo di estensioni scritte in C/C++. Esso è stato fondamentale per il corretto funzionamento di *MySQL Connector*, un *plugin* necessario per la comunicazione tra l'applicazione *web* e il *database* in ambiente *Windows*;
- una lista di *utility* la cui installazione risultava necessaria ai fini di un corretto funzionamento dell'applicazione *web* in ambiente di sviluppo, quali ad esempio *NodeJS*.

2.4.2 Vincoli di metodo

Durante tutta l'attività di stage, ho dovuto rispettare il vincolo posto dall'azienda relativamente all'utilizzo di *TortoiseSVN* per la gestione dell'avanzamento di versione relativamente alle componenti sviluppate di *Experenti Self Service*, che sono state inserite in un *repository* interno al *server* locale di Experenti.

2.4.3 Vincoli temporali

Il progetto prevedeva che lo stage si svolgesse in un totale di 320 ore di attività presso l'azienda ospitante, suddivise in circa 40 ore settimanali soggette a possibili variazioni nel caso di scadenze aziendali o di impegni di varia natura da entrambe le parti. All'interno di questo lasso di tempo era presente una suddivisione principale in due finestre di tempo pressoché della stessa durata tali per cui:

- il primo periodo doveva racchiudere tutte le attività relative alla formazione sulle tecnologie coinvolte nel progetto, sia quelle ad utilizzo diretto che quelle ad utilizzo indiretto;
- il secondo periodo doveva consistere nella realizzazione pratica del *software* richiesto, mettendo in pratica e fissando quanto appreso nel primo periodo.

2.5 Prospettive

Come anticipato, Experenti è un'azienda che sta vivendo un momento di forte espansione: un numero sempre maggiore di aziende infatti si interessa alla realizzazione di

applicazioni *mobile* in realtà aumentata per presentare i loro prodotti in maniera semplice, innovativa e di grande effetto. Per questo va specificato che il mio progetto di stage rientra all'interno di una strategia aziendale più ampia, la quale desidera:

1. aumentare il proprio organico, valutando l'inserimento a lungo termine in azienda di chi viene giudicato positivamente durante lo stage effettuato presso di essa. Avendo infatti lo stage una durata di 320 ore, risulta possibile per Experenti svolgere delle valutazioni oggettive sulle competenze del candidato, mentre lo *stagiaire* è in grado di capire se può essere di suo interesse crescere all'interno dell'azienda in oggetto;
2. espandersi, sviluppando progetti sempre più nuovi ed innovativi. In particolare, è sufficiente analizzare il prodotto sviluppato all'interno del mio stage per constatare l'attenzione e la motivazione posta dall'azienda nella ricerca di soluzioni che le permettano di raggiungere un grado di innovazione sempre maggiore, in particolar modo rispetto ai *competitor* presenti sul mercato.

Capitolo 3

Dettaglio dello stage

3.1 Pianificazione di progetto

Nel periodo antecedente allo stage, in collaborazione con il mio tutor aziendale è stato redatto un documento di piano di lavoro nel quale sono stati definiti i dettagli relativi alle mie attività presso l'azienda ospitante. La pianificazione iniziale in esso contenuta stabiliva che lo stage si sarebbe dovuto svolgere in circa 40 ore settimanali, per un totale di 320 ore di attività presso l'azienda ospitante svolte internamente all'orario d'ufficio, il quale va dal lunedì a venerdì dalle 9:00 alle 13:00 e dalle 14:30 alle 18:30. Le date concordate di inizio e fine stage erano, rispettivamente, 2014-09-08 e 2014-11-14.

Le attività che avrei dovuto svolgere presso Experenti sono state espresse attraverso il formalismo grafico del Diagramma di Gantt, grazie al quale la quantità di ore assegnata ad ogni attività è risultata facilmente identificabile sin dall'inizio, permettendomi di organizzare al meglio il tempo a mia disposizione e di constatare continuamente, in corso d'opera, la differenza tra le ore preventivate e le ore effettivamente spese per ciascuna di esse.

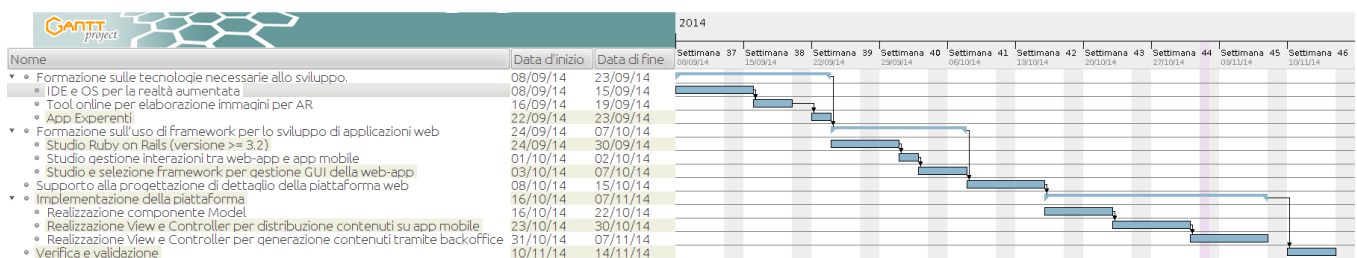


figura 3.1: Diagramma di Gantt relativo alle attività pianificate

Tale diagramma è rimasto inalterato, in quanto l'unica modifica del piano ad aver coinvolto le attività in esso descritte è consistita nel dare maggior peso all'implementazione di un *backoffice* per l'inserimento dei contenuti multimediali, ed al delineamento di un'interfaccia grafica per l'applicazione *web*, piuttosto che all'implementazione di

un sistema per la registrazione sulle statistiche relative ai contenuti visualizzati dagli utenti.

Punto	Descrizione	Ore lavoro
1(a)	Formazione su <i>IDE</i> e <i>OS</i> per la realtà aumentata	40
1(b)	Formazione su <i>tool</i> online per elaborazione immagini per <i>AR</i>	16
1(c)	Formazione sull' <i>app</i> <i>Experenti</i>	4
2(a)	Formazione su <i>Ruby on Rails</i> (versione ≥ 3.2)	36
2(b)	Formazione su gestione interazioni tra applicazione <i>web</i> e <i>app mobile</i>	20
2(c)	Formazione su <i>framework</i> per gestione <i>GUI</i> dell'applicazione <i>web</i>	24
3	Supporto alla progettazione di dettaglio della piattaforma <i>web</i>	38
4(a)	Realizzazione componente <i>Model</i>	30
4(b)	Realizzazione <i>View</i> e <i>Controller</i> per distribuzione contenuti su <i>app mobile</i>	40
4(c)	Realizzazione <i>View</i> e <i>Controller</i> per generazione contenuti tramite <i>backoffice</i>	40
5	Verifica e validazione	32
TOTALE:		320

tabella 3.1: Tabella relativa al dettaglio ore per le attività pianificate

3.2 Dominio applicativo

Segue una panoramica delle tecnologie e strumenti utilizzati contestualmente allo stage.

3.2.1 Tecnologie per le app mobile in realtà aumentata

Nella prima parte del periodo di formazione, l'apprendimento è stato focalizzato alle tecnologie impiegate dall'azienda per la realizzazione di *app* nel settore della realtà aumentata. Questa infarinatura iniziale è stata essenziale per comprendere il contesto tecnologico in cui mi stavo muovendo, e senza di essa non avrei potuto contribuire nella successiva attività di progettazione.

Il primo strumento con cui mi sono interfacciato è stato *Unity*, un sistema *cross-platform* principalmente atto alla creazione di videogiochi. Attraverso un *game engine* ed un *IDE* integrato, permette di sviluppare applicazioni per molti ambienti diversi. All'interno dell'azienda, viene utilizzato per lo sviluppo di *app mobile* per i sistemi operativi *Android* e *iOS*. La mia formazione su tale strumento è avvenuta attraverso due canali:

- una serie di *tutorial* disponibili in rete, che mi hanno aiutato nell'apprendimento dei comandi principali interni al *software*;
- diverse spiegazioni e lezioni ad opera del personale tecnico dell'azienda, che hanno chiarito i miei dubbi e grazie ai quali ho potuto approfondire alcuni aspetti.

Terminata l'introduzione a *Unity*, sono passato all'apprendimento di *Vuforia*, ovvero l' *SDK* utilizzata dall'azienda per la gestione della realtà aumentata all'interno delle *app* sviluppate. Il suo utilizzo è semplificato grazie alla presenza di una estensione *ad hoc* per *Unity*, grazie alla quale è possibile operare all'interno di un unico ambiente di lavoro. Questo *SDK* supporta diversi tipi di riconoscimento dei *tag*, i quali possono essere oggetti 2D o 3D, e offre funzioni molto particolari quali ad esempio i *virtual buttons*: essi sono dei tasti virtuali collocati nello spazio visibile attraverso la fotocamera del dispositivo in uso, e che si attivano non appena l'area da loro ricoperta all'interno del *tag* viene parzialmente (o totalmente) nascosta alla fotocamera del device da un qualunque altro oggetto (si parla di "occlusione"). Grazie ad essi è, quindi, possibile simulare la pressione di un pulsante, opportunamente configurabile, per l'attivazione di ulteriori contenuti in realtà aumentata.

Alla fine di questa prima parte di formazione, mi è stata data la possibilità di sviluppare un'*app mobile* con tema libero per mettere in pratica le nozioni acquisite. Ho apprezzato molto che mi sia stata data questa opportunità, e ne ho approfittato per realizzare una piccola *app* di combattimento tra *avatar* in realtà aumentata. Questo lavoro è stato particolarmente apprezzato all'interno dell'azienda tanto da, dopo le dovute rivisitazioni, essere incluso all'interno dell'applicazione *mobile* Experenti in uno dei suoi aggiornamenti.

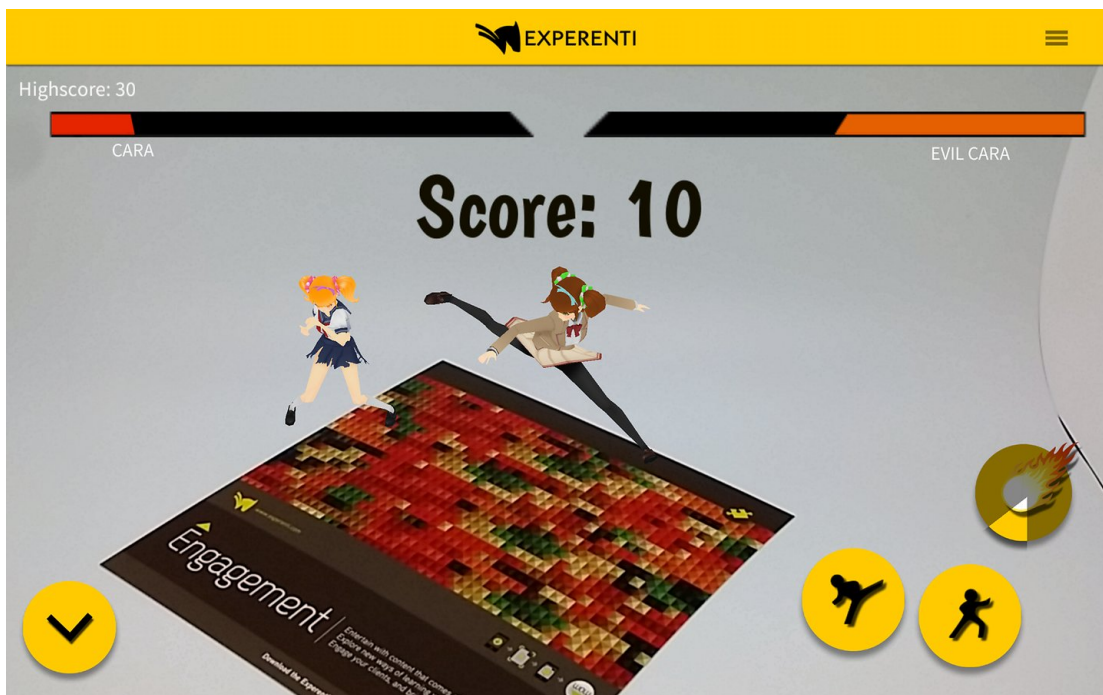


figura 3.2: Screenshot relativo all'applicazione di lotta rivisitata ed inserita nell'*app* Experenti

3.2.2 Tecnologie per lo sviluppo di applicazioni web

Nella seconda ed ultima parte del periodo di formazione, il mio studio è stato volto alle tecnologie che avrei dovuto utilizzare per la realizzazione dell'applicazione *web Experienci Self Service*.

Ruby on Rails

Ruby on Rails è un *framework open source* nato nel 2005 che punta a permettere di sviluppare applicazioni *web* scrivendo meno codice rispetto ad altri *framework* concorrenti. Un suo grande punto di forza è la tendenza a forzare l'utilizzo di molti pattern e principi guida impiegati nell'Ingegneria del Software. In particolare all'interno dei concetti su cui *Ruby on Rails* si basa, vi sono:

- **Convention over configuration (CoC)**: la creazione di un progetto in *Ruby on Rails* implica che le cartelle e i file relativi alle componenti dell'applicazione *web* seguano una struttura prefissata. Questa limitazione fa in modo che i programmatori debbano configurare solo gli aspetti che si differenziano dalle implementazioni standard, e semplifica di molto il lavoro in team sul medesimo progetto, in quanto riduce ampiamente il rischio che si presentino eventuali errori derivanti da una errata configurazione in corso d'opera;
- **Don't repeat yourself (DRY)**: tramite degli strumenti preconfezionati quali *partials* e *filters*, *Ruby on Rails* permette di eliminare le duplicazioni presenti nel codice, evitando di dover scrivere più volte le stesse cose e rendendo il codice più pulito.

Per quanto riguarda, invece, il punto di vista architetturale, alla base del *framework* vi sono due architetture principali:

- **Model View Controller (MVC)**: *Ruby on Rails* è interamente costruito intorno al pattern *MVC*. In questo modo, la logica di *business* è totalmente separata dall'interfaccia utente, gli interventi di manutenzione risultano facilitati ed è più semplice applicare il principio *DRY*.

Come visibile nella figura 3.3, il flusso delle richieste segue un certo ordine:

1. il browser in uso invia una richiesta *HTTP*;
2. i meccanismi di *routing* e *dispatching* interni al framework identificano il corretto *controller* e la corrispondente *action* che devono occuparsi di gestire la richiesta;
3. il *controller* interagisce con il *model*, istanziando nuove entità o manipolando quelle già esistenti. Questo è possibile grazie ad una comunicazione diretta tra il *model* ed il *database* utilizzato per l'immagazzinamento dei dati relativi all'applicazione *web*;
4. il *controller* invoca la corretta *view*;
5. la *view* determina la successiva schermata che verrà visualizzata dal browser.

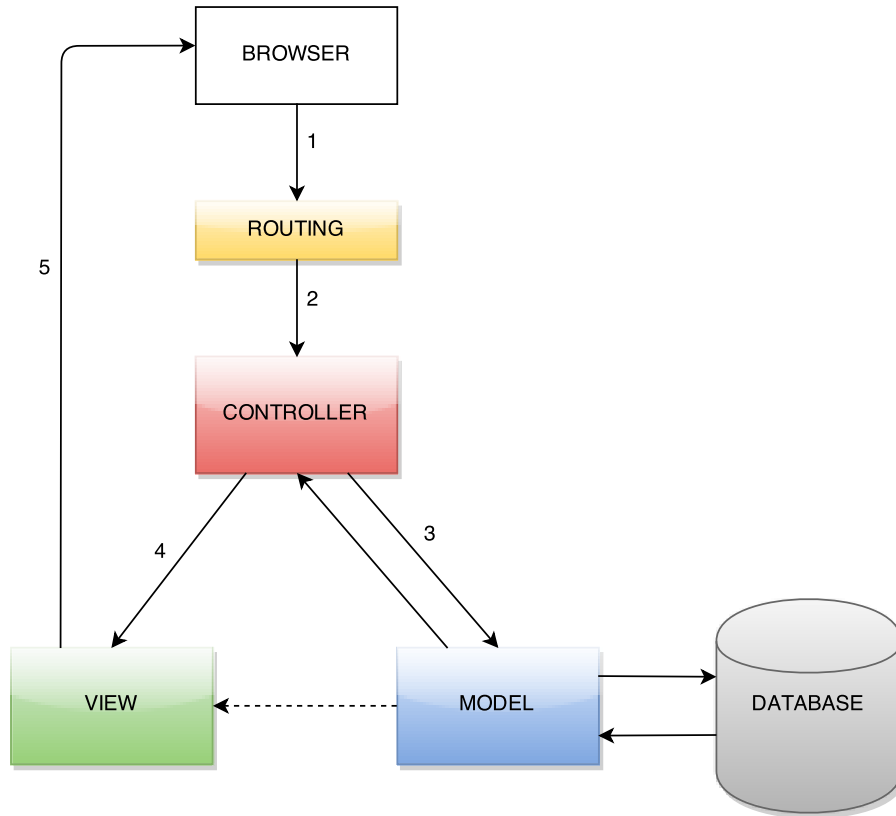


figura 3.3: Design pattern MVC applicato al framework Ruby on Rails

- **REpresentational State Transfer (REST):** questo tipo di architettura *software* è ideale per le applicazioni *web*, in quanto permette di organizzarle strutturandole attraverso risorse e gli *HTTP verbs* standard.

Nello specifico, l'utilizzo di questo *framework* è stato deciso dall'azienda: grazie ai principi su cui si basa e alle *utility* che mette a disposizione, *Ruby on Rails* rappresentava il candidato perfetto da adoperare per la realizzazione di un progetto di questo tipo all'interno di un'azienda che adotta il modello di sviluppo *agile*. Tale *framework* permette infatti uno sviluppo ad iterazioni, riduce enormemente il tempo che normalmente si impiegherebbe per gestire la configurazione del progetto, e tutte le funzionalità che offre si sposano a pieno con i principi sanciti dall'*Agile manifesto*, in particolar modo con quanto riguarda la prontezza di risposta al cambiamento.

Grazie a *Ruby on Rails* è estremamente semplice estendere le funzionalità del proprio *software*: in rete è infatti possibile trovare una vastissima quantità di *plugin* detti "gemme", la quale installazione risulta molto semplice e permette in pochi passi di accedere alle funzionalità aggiuntive desiderate. Tali gemme, una volta installate, sono riutilizzabili in più progetti: è sufficiente includerle nel *Gemfile*, un file interno ad ogni progetto *Ruby on Rails* che tiene traccia di tutti i *plugin* da cui dipende l'applicazione in sviluppo e, grazie all'utilizzo di uno strumento chiamato *bundler*, ne semplifica enormemente l'aggiunta, la rimozione o l'aggiornamento. Grazie alle

informazioni contenute nel *Gemfile* è inoltre possibile rendere portabili le applicazioni sviluppate, permettendo agli sviluppatori di risolvere le dipendenze necessarie con molta semplicità.

L'insieme di tutte queste caratteristiche unite al fatto che, nonostante la sua recente introduzione, *Ruby on Rails* vanta già una grande comunità e, come anticipato, un grande archivio di gemme disponibili in rete, lo ha reso fin dall'inizio il candidato ideale per lo sviluppo dell'applicazione *web* in esame.

MySQL

La scelta del *database* in cui riversare i dati dell'applicazione è ricaduta su *MySQL*. Questa scelta è dovuta a diverse ragioni infatti, tra i vari *relational database management system*, *MySQL*:

- vanta notevole stabilità ed una grande comunità a supporto, rispetto ad alternative di più recente introduzione;
- è sicuramente uno dei più facilmente integrabili con il framework *Ruby on Rails*;
- era quello su cui sia io, sia i miei colleghi del reparto tecnico avevamo più esperienza.

Aptana Studio

Per quanto riguarda l'*integrated development environment (IDE)* con cui sviluppare *Experenti Self Service*, la scelta è stata totalmente delegata a me. Dopo un attento studio, ho deciso di utilizzare *Aptana Studio*, un *IDE open source* realizzato appositamente per lo sviluppo di applicazioni *web*. Le caratteristiche che mi hanno spinto verso questa scelta sono la presenza di funzionalità quali una *shell console* per *Ruby on Rails* integrata, quella di auto-completamento del codice e quella di una visualizzazione semplificata della gerarchia di progetto. Ulteriori caratteristiche quali la *dual-licensing GPL/APL* e la facilità di utilizzo dovuta al fatto che ha la medesima struttura di *Eclipse* (programma sul quale si basa e che già conoscevo) lo hanno favorito rispetto ad altri *IDE* concorrenti.

Bootstrap

La scelta del *framework* da utilizzare per la realizzazione del *front-end* ha richiesto uno studio più accurato. I principali candidati erano fin dall'inizio *Bootstrap* e *Polymer*, i più utilizzati in rete. Queste due alternative mi erano state inizialmente suggerite dai colleghi del reparto tecnico, i quali mi hanno chiesto di effettuare uno studio per capire quale dei due *framework* fosse più adatto contestualmente all'applicazione in sviluppo. Nello specifico, entrambi contengono una serie di strumenti atti al *responsive web design*, ovvero alla realizzazione di siti *web* che mantengano una visualizzazione corretta e piacevole qualunque sia il dispositivo con cui li si sta visualizzando, sia esso un computer fisso, o uno *smartphone*. Entrambi hanno

particolari punti di forza e debolezza: lo studio svolto ha però rivelato che, nonostante *Polymer* vantasse un notevole adempimento agli standard del *web* attraverso l'utilizzo di strumenti quali i *custom elements*, *Bootstrap* risultava molto più flessibile, più facilmente personalizzabile e più maturo grazie al supporto di una *community* davvero ampia.

Per questi motivi, e per il fatto che risultava molto più facilmente integrabile con *Ruby on Rails*, la mia scelta è ricaduta su *Bootstrap*: ho riportato il risultato del mio studio ai miei colleghi del reparto tecnico, gli argomenti a supporto sono stati ritenuti validi e la mia scelta è stata approvata.

3.3 Svolgimento delle attività

Come già anticipato, le mie attività pianificate relativamente alla realizzazione di *Experenti Self Service* comprendevano un supporto alla progettazione, lo sviluppo e parte delle attività di verifica e validazione della piattaforma. Al mio arrivo in azienda, l'attività di analisi era già stata debitamente svolta internamente dai miei colleghi del reparto tecnico, i quali avevano già identificato gli attori e i requisiti che il prototipo in sviluppo avrebbe dovuto soddisfare.

3.3.1 Progettazione

Durante l'attività di progettazione è stata delineata l'architettura generale dell'applicazione, ed in seguito tutto il gruppo dei componenti che avrei dovuto implementare durante l'attività di codifica, che grazie a questo si è potuta svolgere come pura trasposizione di quanto definito in quest'attività. In quanto ruolo di supporto, ho potuto affiancare il reparto tecnico e contribuire grazie alle nozioni apprese nel periodo di formazione iniziale.

Architettura generale

Da subito sono state definite quelle che avrebbero dovuto essere le entità principali impiegate nell'applicazione, ovvero:

- **User:** è l'utente che andrà ad utilizzare l'applicazione. Ogni utente per usufruire delle funzionalità offerte dalla piattaforma dovrà effettuare il *login* utilizzando l'*email* e la *password* stabilite in fase di registrazione;
- **App:** rappresenta una *app mobile* sviluppata da Experenti. Ogni *app* sarà associata ad un utente;
- **AppHistory:** indica un avanzamento di versione, con relativo *log*, relativo ad una *app* presente nel sistema;
- **Dataset:** è una collezione di *tag* associati ad una *app*;
- **Target:** rappresenta un *tag*, ovvero un'immagine che, non appena identificata tramite l'inquadratura della fotocamera all'interno dell'*app* ad esso associata, farà scattare la visualizzazione di un contenuto, il quale apparirà in realtà aumentata sullo schermo del dispositivo in uso;

- **Content:** è il contenuto multimediale che dovrà essere visualizzato sullo schermo del dispositivo non appena sarà stato inquadrato il corrispondente *tag*. Questa in realtà è una macro-entità che raccoglie tutti i file ed i meta-dati a loro annessi, associati ai contenuti multimediali e necessari alla loro fruizione in realtà aumentata. Come già anticipato, per la realizzazione del primo prototipo i contenuti saranno semplicemente dei video, i quali potranno avere associato un'immagine con il ruolo di *preview*;
- **Association:** è l'entità che rappresenta l'associazione esistente tra un *tag* ed un contenuto multimediale. Nel momento in cui esisterà un'associazione tra queste due entità, sulla corrispondente *app* si potrà accedere al contenuto in realtà aumentata semplicemente inquadrando il *tag* associato.

Come visibile in 3.4, oltre alle classi principali è stata definita tutta una serie di classi denominate *DraftName*. Esse appartengono ad un *package* chiamato *Draft* e sono state progettate per gestire i dati “temporanei” all'interno della piattaforma, ovvero tutta quella serie di dati, siano essi relativi ad *app*, *dataset*, *tag* o associazioni che sono state inserite nel sistema da utenti esterni e non hanno ancora ricevuto una conferma che li ufficializzi. Per fare in modo che le modifiche operate sulla piattaforma *SaaS* si rispecchino nelle *app mobile* sviluppate da Experenti è infatti necessario che esse vengano confermate: la conferma avverrà da parte degli amministratori del sistema se si tratta di operazioni quali la conferma alla creazione di un *dataset*, mentre sarà operata direttamente dall'utilizzatore stesso della piattaforma in caso si tratti di modifiche alle associazioni presenti tra *tag* e video di un'*app*.

La piattaforma è stata progettata attenendosi all'implementazione di *MVC* fornita dal *framework Ruby on Rails*, il che ha comportato una separazione netta delle componenti *Model*, *View* e *Controller* realizzate. Come anticipato, relativamente a *View* e *Controller*, l'applicazione doveva includere:

- componenti per la distribuzione dei contenuti multimediali inseriti nella piattaforma tramite *API*;
- componenti per l'inserimento dei contenuti multimediali nella piattaforma.

Questa necessità, ai fini della progettazione architetturale, si è riversata nella scelta di separare le due logiche, realizzando un *package* a sè per le componenti *View* e *Controller* relative alle *API*.

Le API

Progettando un *package* separato per la gestione delle *API* fornite dall'applicazione, l'architettura è risultata più pulita ed estendibile: fin dall'inizio per l'azienda era prioritaria la realizzazione di un semplice meccanismo di gestione delle *API*, con particolare attenzione al metodo di avanzamento di versione in quanto, vista la variabilità delle tecnologie coinvolte e vista la necessità di comunicare con *app* provenienti sia da ecosistemi *Android* che *iOS*, era già previsto che le *API* potessero evolvere con molta rapidità.

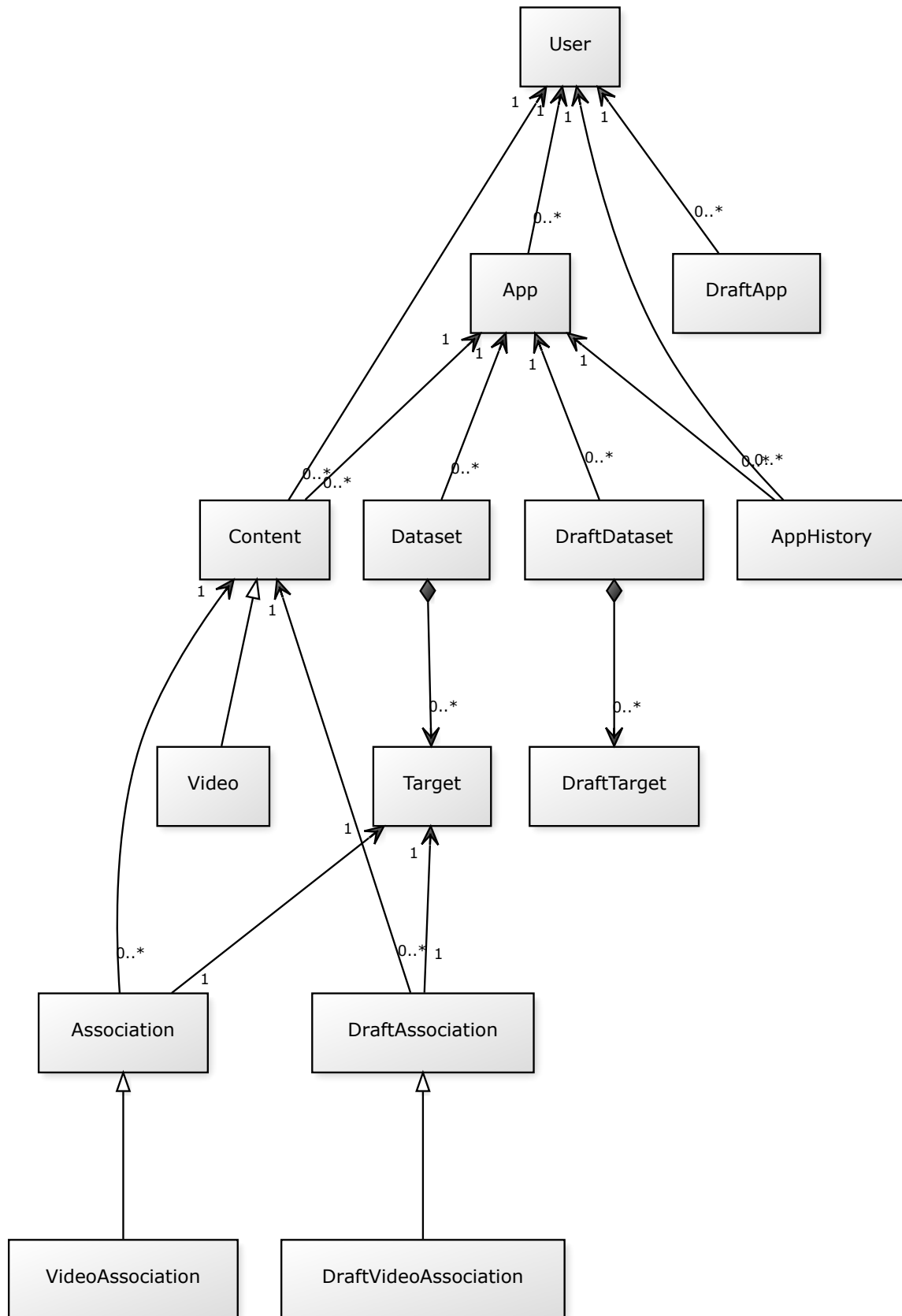


figura 3.4: Diagramma delle classi del *Model* di *Experiential Self Service*

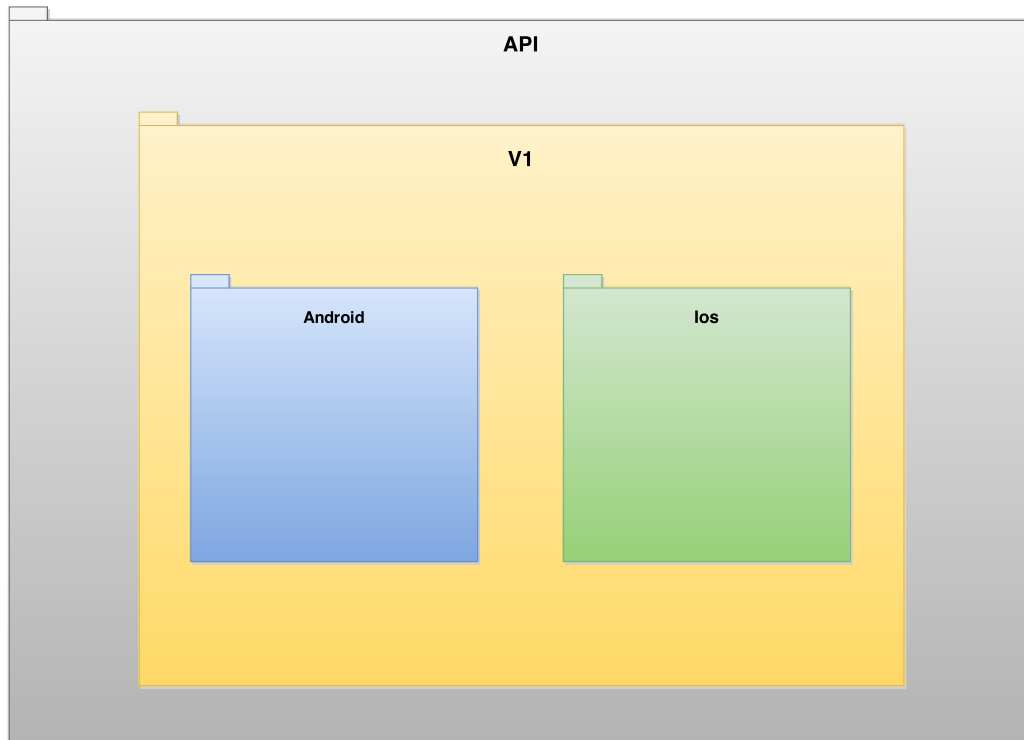


figura 3.5: Diagramma vista *package* delle *API* di *Experenti Self Service*

Quindi, come visibile in figura 3.5, è stato previsto un ulteriore *package* per ogni versione di *API* rilasciata o in sviluppo, per favorire l'estendibilità dell'architettura; all'interno di essi sono presenti due ulteriori *package* *Android* e *Ios* per permettere la separazione della logica di controllo in base alle necessità dei due diversi sistemi operativi.

Come anticipato, le *API* dell'applicazione dovevano principalmente servire come mezzo per lo scambio di informazioni tra le *app mobile* e la piattaforma *SaaS*. Attraverso uno studio svolto preventivamente con i colleghi del reparto tecnico, sono state identificate le *API* che l'applicazione *web* avrebbe dovuto necessariamente fornire. In tale occasione è stata anche concordata l'esigenza per le *API* di avere accesso, allo scopo di produrre informazioni "uscenti", ad alcune informazioni "entranti" accodate alle richieste. Nello specifico:

- le informazioni entranti quali, ad esempio, l'identificativo di un contenuto multimediale che si desidera scaricare, verranno raccolte attraverso dei parametri accodati all'*url* della richiesta *HTTP*. Tali parametri saranno soggetto di *parsing* e di una successiva validazione per evitare problematiche relative alla sicurezza;
- le informazioni uscenti verranno distribuite tramite la restituzione di *file* in formato *JSON*. Tale formato è particolarmente indicato per scambi di questo tipo in quanto universalmente riconosciuto e di facile *parsing* nel contesto dei dispositivi *mobile*.

Un esempio di risposta della piattaforma *SaaS* in formato *JSON* è la seguente:

```
{
  "status": "success",
  "data": {
    "datasets": {
      "id": 3,
      "name": "Dataset_Demo",
      "active": true,
      "created_at": "2014-11-01T10:42:50.000Z",
      "updated_at": "2014-11-01T11:08:35.000Z",
      "links": {
        "download_xml_url": "http://diy.experenti.com/api/v1/android/datasets/3/download_xml",
        "download_dat_url": "http://diy.experenti.com/api/v1/android/datasets/3/download_dat",
        "download_associations_url": "http://diy.experenti.com/api/v1/android/datasets/3/download_associations"
      }
    }
  }
}
```

I due attributi *status* e *data* sono comuni a tutti i file *JSON* ottenuti tramite le *API*: *status* è un flag indicatore del risultato della richiesta *HTTP*, la quale può avere successo (valore “*success*”) o fallire (valore “*error*”); *data* invece è il macro-contenitore per i dati richiesti. Ho predisposto questa struttura basandomi sugli standard e sulle *best-practice* ad oggi in uso nelle architetture *REST* in cui vi è uno scambio di informazioni tramite *file* di tipo *JSON*: essa è stata sottoposta ai colleghi del reparto tecnico e approvata.

Le principali *API* da sviluppare dovevano permettere alle *app*:

- il controllo della presenza di aggiornamenti per un *dataset* di una data *app*;
- il *download* delle associazioni esistenti tra i tag ed i contenuti multimediali relativi ad un *dataset*;
- il *download* di un video, con la possibilità di scaricare anche il *frame* di *preview* associato, per la sua presentazione in realtà aumentata.

A supporto di queste funzionalità è stata richiesta anche la redazione di un manuale con la specifica relativa ai formati degli *URL* per le richieste *HTTP* e a quelli dei *file JSON* restituiti dalla piattaforma *SaaS*, destinato ai colleghi del reparto tecnico che dovevano occuparsi dello sviluppo della controparte *mobile*. Tale documento è stato regolarmente prodotto, e la sua versione ultimata è stata consegnata in corrispondenza al termine dello stage: per permettere, però, al reparto tecnico di predisporre adeguatamente le *app* sviluppate in azienda ad un corretto interfacciamento con la piattaforma *SaaS*, sono state rilasciate e consegnate diverse bozze del documento già in corso d’opera.

Il database

Particolare attenzione è stata rivolta alla progettazione del *database* della piattaforma. In questo contesto era molto importante decidere la strategia per la definizione delle chiavi primarie e delle chiavi esterne relative alle varie entità, e stabilire delle norme riguardanti la nomenclatura delle tabelle. Grazie all’iniziale periodo di formazione, avevo appreso che il framework *Ruby on Rails* fa uso di *Active Record* per la gestione del *database* associato all’applicazione *web*, ovvero di un sistema di *Object Relational Mapping* che permette, attraverso l’utilizzo di convenzioni sulla nomenclatura, di prendere le proprietà e le relazioni scritte all’interno dell’applicazione e di trasporle direttamente all’interno del *database* associato senza alcun bisogno di scrivere direttamente codice *SQL*. Per quanto riguarda le *naming conventions*, *Ruby on Rails* richiede semplicemente che gli si indichi il nome delle classi in inglese, le quali devono avere nome singolare con iniziale maiuscola: a quel punto tramite il suo meccanismo interno di pluralizzazione, il *framework* troverà il corrispettivo termine al plurale e lo userà come nome per la relativa tabella del *database* (con iniziale minuscola ed usando il carattere “_” ove dovrebbero esserci degli spazi). Per quanto riguarda invece le *schema conventions*, relativamente alle tabelle del *database* il *framework*:

- come chiave primaria, utilizza di *default* una colonna di tipo *integer* chiamata *id* auto-incrementante creata da *Active Record*;
- come chiave esterna, suggerisce l’utilizzo di un attributo *integer* il cui nome è formato dal nome della tabella di riferimento in inglese singolare facendo uso del carattere “_” ove dovrebbero esserci degli spazi, ed accodando la dicitura “_id”, in modo che *Active Record* sia in grado di riconoscere le associazioni esistenti tra i modelli.

Ho suggerito quindi di adoperare le convenzioni adottate dal *framework*, in quanto ben congeniate e già predisposte per il lavoro, e la scelta è stata approvata: quindi, come esempio, la classe *App* ha come corrispettiva tabella nel *database* un’entità chiamata *apps*, mentre la tabella relativa alla classe *DraftDataset* è denominata *draft_datasets*.

3.3.2 Codifica

L’attività di codifica si è svolta senza particolari intoppi, grazie al periodo di formazione iniziale che mi ha permesso di essere in grado di implementare correttamente quanto stabilito in attività di progettazione. Il lavoro è cominciato con la realizzazione del componente *Model* della piattaforma e, conseguentemente grazie ad *Active Record*, alla definizione delle tabelle del *database MySQL*. Ogni entità implementata permette quantomeno le operazioni *CRUD*, ovvero *Create*, *Read*, *Update* e *Delete*.

Il *Model* è stato forse la componente il cui sviluppo è stato più influenzato dalla metodologia *agile* adottata in azienda: la sua realizzazione, infatti, non è avvenuta in blocco bensì tramite degli incrementi avvenuti attraverso il concetto di “migrazioni”. Esse consistono in un meccanismo offerto da *Ruby on Rails*, e più nello specifico da *Active Record*, per semplificare al massimo la creazione e la modifica delle entità

interne al *database*, senza che vi sia il bisogno di svolgere tali operazioni attraverso *query SQL*. Le migrazioni altro non sono che classi *Ruby*, il cui nome segue le convenzioni precedentemente citate, ma che viene preceduto da un *timestamp* seguito dal carattere “_”. Nell’esempio contenuto nel *listing 3.1*, si indica la volontà di inserire nel database una tabella chiamata *app_histories* con svariati attributi (tra cui *id*, aggiunto implicitamente, e gli attributi *created_at* e *updated_at*, anch’essi gestiti automaticamente da *Active Record*).

Listing 3.1: Esempio di classe relativa ad una migrazione in *Experenti Self Service*

```
class CreateAppHistories < ActiveRecord::Migration
  def change
    create_table :app_histories do |t|
      t.string :version
      t.date :release_date
      t.string :log
      t.belongs_to :app, index: true
      t.belongs_to :user, index: true

      t.timestamps
    end
  end
end
```

Nel momento in cui si fa partire l’esecuzione della migrazione, la tabella viene inserita nel *database*. La cosa più interessante relativa alle migrazioni è, però, la possibilità di invertirle: ad esempio, operando il cosiddetto *rollback* sulla migrazione del *listing 3.1* (naturalmente, dopo averla eseguita), il *framework* rimuoverà dal *database* la tabella appena inserita, senza lasciarne traccia. Per quanto riguarda invece le operazioni di modifica, si sono rivelate particolarmente utili le direttive *add_column* e *remove_column*, per aggiungere e rimuovere, rispettivamente, colonne relative ad attributi su tabelle già esistenti nel *database*.

Le migrazioni sono state di fondamentale importanza durante l’implementazione, in quanto hanno semplificato moltissimo le operazioni di incremento sul *database* richieste dalla metodologia *agile*, che normalmente richiedono invece uno sforzo non indifferente, rispettando a pieno il principio di prontezza al cambiamento. E’ capitato più di una volta, infatti, che, durante l’implementazione delle componenti *View* e *Controller*, io dovessi rimettere mano alla componente *Model*, con particolare riferimento al *database*, per estenderne le funzionalità o apportare delle modifiche (spesso in seguito al riscontro dell’esito dei test svolti dai colleghi del reparto tecnico).

Particolare attenzione è stata riservata alla gestione delle validazioni: in *Ruby on Rails* è infatti possibile indicare, all’interno delle classi relative alla componente *Model*, delle regole di validazione le quali, se non rispettate, impediscono ad un oggetto di essere memorizzato nel *database*. Questa funzionalità si è dimostrata di fondamentale importanza, in quanto ha permesso un adeguato controllo rispetto alla presenza di eventuali incoerenze nei dati immessi nel sistema. Alcuni tra i molti tipi possibili di validazione sono:

- *format*, che esegue la validazione testando che il valore in esame rispetti un'espressione regolare data;
- *length*, che permette la memorizzazione di un determinato valore solo se la sua lunghezza in caratteri è inferiore/superiore ad una certa soglia prefissata;
- *numericality*, che valida il valore dell'attributo solamente se formato da numeri;
- *presence*, il quale verifica che il valore dell'attributo in esame sia non vuoto;
- *uniqueness*, che permette di verificare che il valore dell'attributo in questione sia unico all'interno del sistema prima che l'oggetto ad esso associato sia salvato.

Listing 3.2: Esempio di validazioni interno alla classe *App* nella componente *Model*

```
class App < ActiveRecord::Base
  belongs_to :user

  has_many :contents, dependent: :destroy
  has_many :app_histories, dependent: :destroy
  has_many :draft_datasets, dependent: :destroy
  has_many :datasets, dependent: :destroy

  validates :name, :bundle_id, :version, :user_id, :contents_
    limit, :icon, :android_first_cat, :ios_first_cat, :short_
    descr, :long_descr, :support_email, :support_url,
    presence: true

  validates :user_id, numericality: { greater_than: 0, message
    : "isn't set to a proper value"}

  validates :name, length: { maximum: 20 }

  validates :short_descr, length: { maximum: 80 }

  validates :long_descr, length: { maximum: 4000 }

  has_attached_file :icon,
    :url => "/storage/#{Rails.env}#{ENV['RAILS_TEST_NUMBER']}/
      official_apps/icons/:id/:basename.:extension",
    :path => ":rails_root/public/storage/#{Rails.env}#{ENV['
      RAILS_TEST_NUMBER']}/official_apps/icons/:id/:basename
      .:extension"

  validates_attachment_content_type :icon,
    :content_type => ['image/png', 'image/jpeg', 'image/jpg']

  validates_uniqueness_of :bundle_id
  validates_uniqueness_of :name

  ...
end
```


Relativamente all'ambito della coppia *View - Controller*, va fatta una precisazione. Le *View* sviluppate all'interno di *Ruby on Rails* non sono vere e proprie classi, ma pagine con estensione *html.erb* o *json.jbuilder*: per entrambi la prima estensione del *file* indica il formato del *template* utilizzato, mentre la seconda estensione indica lo strumento *handler* che viene utilizzato nel *framework* al momento della compilazione per la corretta costruzione della pagina da visualizzare. Grazie a questo meccanismo, l'implementazione dei componenti *View* non è stata particolarmente onerosa: la maggior attenzione è stata giustamente posta nel corretto sviluppo della logica di controllo ad esse associata.

Il routing

Durante la codifica, la corretta implementazione del *routing* è stata prioritaria. Come anticipato, il meccanismo di *routing* interno a *Ruby on Rails* riconosce le *URL* delle richieste e ne effettua il *dispatching* verso il corretto metodo del *Controller* designato alla gestione della richiesta. Questo avviene tramite la dichiarazione di **risorse** all'interno del file di configurazione dei *routes*: una dichiarazione di tipo *resources :users* senza ulteriori vincoli predispone di *default* 7 differenti *routes* all'interno dell'applicazione, corrispondenti alle operazioni *CRUD* con corrispettivi verbi *HTTP*.

Verbo HTTP	Percorso	Controller#azione	Utilizzo
<i>GET</i>	/users	users#index	Mostra la lista degli utenti
<i>GET</i>	/users/new	users#new	Mostra un <i>form</i> per creare un utente
<i>POST</i>	/users	users#create	Crea un nuovo utente
<i>GET</i>	/users/:id	users#show	Mostra i dettagli di un utente
<i>GET</i>	/users/:id/edit	users#edit	Mostra un <i>form</i> per modificare un utente
<i>PATCH/PUT</i>	/users/:id	users#update	Aggiorna i dati di un utente
<i>DELETE</i>	/users/:id	users#destroy	Distrugge i dati relativi ad un utente

tabella 3.2: Tabella relativa ai *default routes* per la risorsa *users*

In questo contesto, per ogni tipo di risorsa presente nel sistema sono stati estesi o limitati i *routes* definiti nella tabella 3.2. Buona parte dell'attività di codifica relativa al *routing* è stata incentrata sulla gestione delle *API*, come visibile nella porzione di codice riportato nel *listing* 3.3. Per permettere la gestione dei contenuti multimediali da *backoffice* e la loro distribuzione tramite *API* è stato necessario implementare due definizioni di *routing* diverse per la medesima risorsa *contents*, separandoli attraverso l'uso dei *namespaces*: la risorsa *contents* in ambiente di *backoffice* consente infatti tutta una serie di operazioni *CRUD* che non sono disponibili nel contesto delle *API*, le quali riducono al minimo necessario le richieste effettuabili, guadagnando in efficienza.

Listing 3.3: Porzione della definizione dei *routes* per le *API* della piattaforma web

```

namespace :api do
  namespace :v1 do
    namespace :android do
      resources :contents, defaults: { format: 'json' }, only:
        [:download] do
        collection do
          get ":id/download", to: "contents#download"
          get ":id/download_keyframe", to: "contents#download_
            keyframe"
        end
      end
    end
  end
  ...
end
end

```

Per quanto riguarda il *download* dei video con relativi *frame* di *preview* tramite *API*, sono state definite delle norme da rispettare nella richiesta *HTTP*: in particolare, perché lo scaricamento vada a buon fine, è necessario accordare alla richiesta un *token* criptato attraverso una procedura comprendente l'utilizzo di algoritmi di *hash* prestabiliti. Questo è stato fatto in quanto le *API* sono state sviluppate all'interno di un contesto *stateless*, per cui si rendeva necessario proteggere l'accesso a contenuti privati tramite meccanismi di autenticazione differenti rispetto alla semplice autenticazione tramite *email* e *password* già utilizzata in *Experenti Self Service*.

L'autenticazione

Per quanto concerne la gestione dei meccanismi di autenticazione, durante l'attività di codifica ho utilizzato le funzionalità offerte dalla gemma *Devise*. Visti i requisiti imposti sulla sicurezza che l'applicazione *web* doveva soddisfare, l'implementazione di un meccanismo di autenticazione *ad hoc* sarebbe risultato in un'operazione ad alto rischio dato il costo in termini di ore di lavoro richieste e vista la complessità di sviluppo di un modulo in sé. Per questo motivo è stato scelto di appoggiarsi ad un modulo esterno quale *Devise* in quanto, dopo un approfondito studio, è stato considerato la migliore alternativa presente tra le varie possibili visto il modo in cui gestisce la sicurezza delle sessioni e data la semplicità di personalizzazione delle logiche di controllo da esso offerta. Questa gemma attraverso una serie di *helpers*, ad esempio

- *authenticate_user!*, che permette di impostare dei filtri nei componenti *Controller* relativi a *View* accessibili solo dopo aver effettuato il *login* (allo scopo di rendere tali pagine inaccessibili ad utenti non autenticati),
- *user_signed_in?* che permette di verificare se l'utente attualmente collegato ha eseguito o meno il *login*,

- *current_user* che è un riferimento all'utente che sta utilizzando la piattaforma,

ha reso possibile una protezione ottimale per tutte le pagine che dovevano risultare inaccessibili da parte di utenti non autenticati.

Data la necessità per l'applicazione di includere una diversa categorizzazione degli utenti all'interno del sistema, che prevedeva:

- *admin*, ovvero gli amministratori del sistema, interni allo *staff* di Experenti,
- *reseller*, gli utenti corrispondenti ai rivenditori di Experenti,
- clienti, ovvero gli utilizzatori finali con i privilegi di base,

questi *helper* sono stati ampiamente utilizzati all'interno dei *Controller* per capire come costruire correttamente le pagine di *View* da restituire all'utente in base ai privilegi assegnatigli.

Il *login* all'interno della piattaforma viene effettuato inserendo *email* e *password* fornite in fase di registrazione, e le informazioni relative alla password vengono adeguatamente criptate prima di essere memorizzate nel *database*.

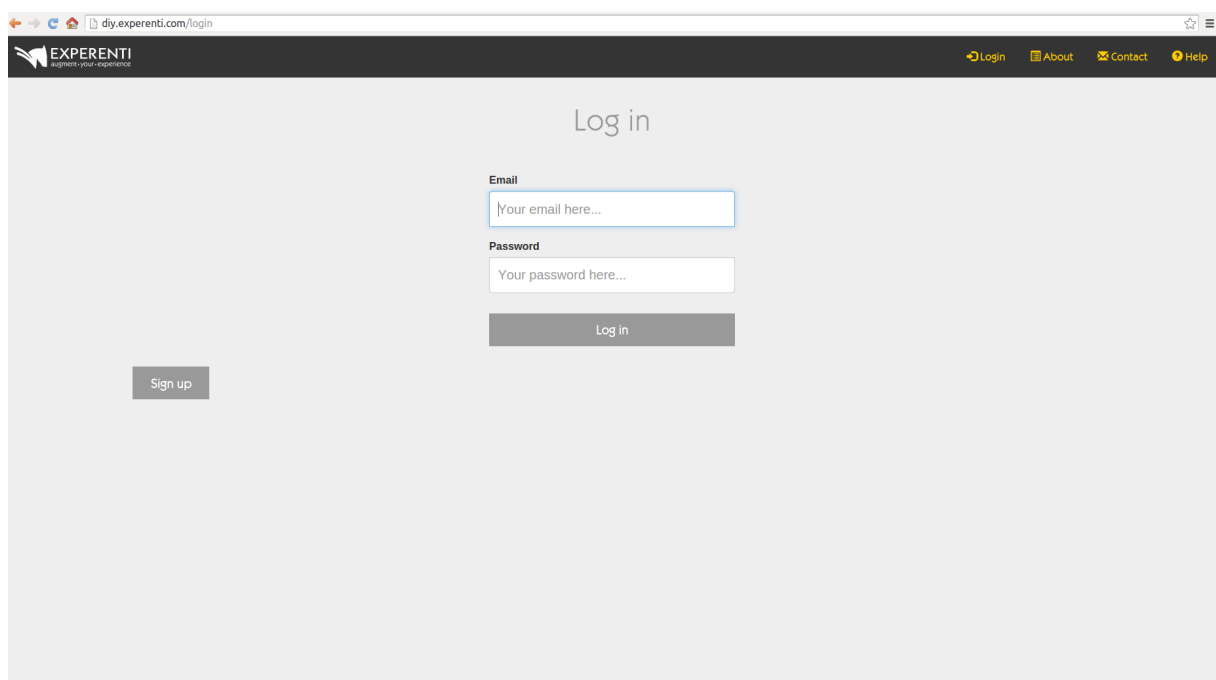


figura 3.6: Schermata di login di *Experenti Self Service*

I contenuti multimediali

I componenti relativi ai contenuti multimediali utilizzati all'interno di *Experenti Self Service* sono stati implementati appoggiandosi ad una gemma chiamata *Paperclip*.

Questo *plugin* è una libreria compatibile con *Active Record* per l'inserimento, il salvataggio e la cancellazione degli allegati di tipo *file*. E' stata scelta in quanto offre una gran quantità di funzioni utili alla gestione dei contenuti multimediali, tra cui i più importanti sono la presenza di meccanismi semplificati per l'accesso e la memorizzazione dei *file* all'interno di percorsi specifici, e la validazione operata sulla dimensione e sul formato dei *file* inseriti. Queste funzioni si sono rivelate di fondamentale importanza all'interno dell'applicazione *web*, in quanto hanno permesso di seguire facilmente le norme prestabilite in azienda per la memorizzazione dei *file* all'interno della piattaforma e di impedire il caricamento di *file* con formato non consono rispetto ai fini del suo funzionamento.

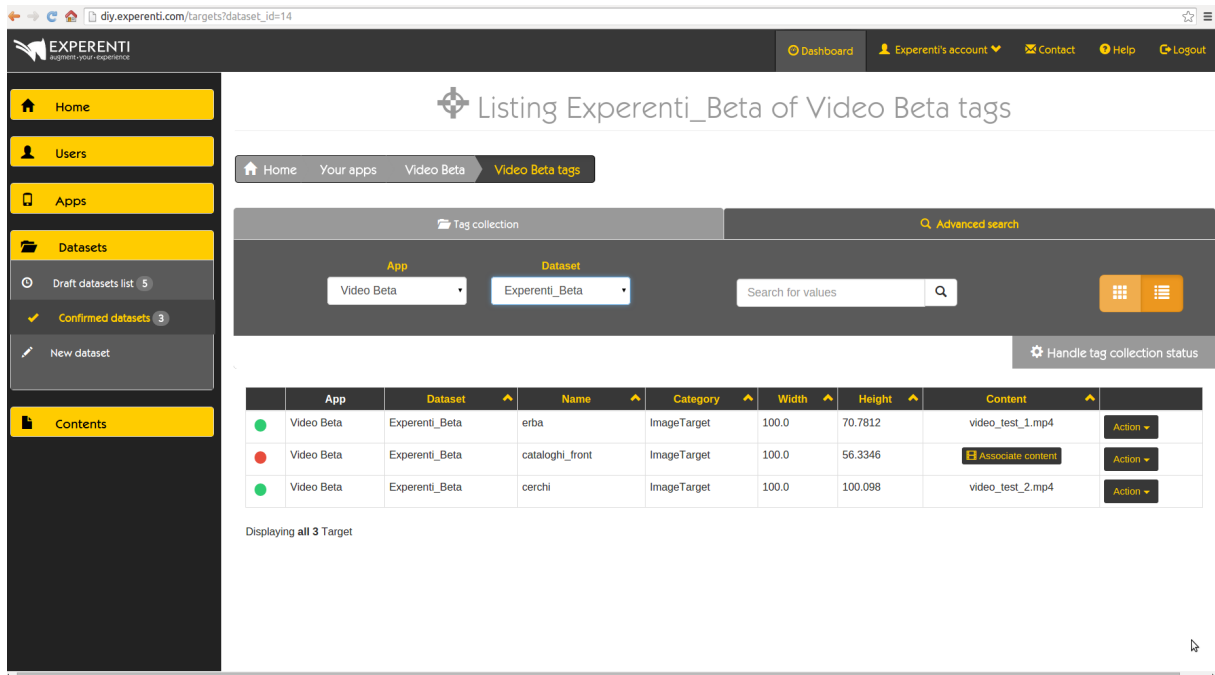
In particolare, come anticipato la configurazione di *default* offerta da *Ruby on Rails* mette a disposizione tre diversi ambienti di lavoro, ovvero:

- *development*, l'ambiente di sviluppo dell'applicazione;
- *production*, predisposto per le applicazioni che hanno subito un *deploy* e sono state messe in esercizio (solitamente appoggiandosi ad un servizio di *hosting* esterno);
- *test*, ambiente dedicato al debug ed allo svolgimento dei test.

Vista questa distinzione, e data la necessità per ognuno di questi tre ambienti di accedere a diversi contenuti multimediali in base al contesto di lavoro, era necessario fin da subito predisporre che la memorizzazione di tali contenuti fosse suddivisa in maniera netta in base all'*environment* di lavoro. Proprio grazie alla gemma *Paperclip*, che offre un totale controllo per quanto riguarda la scelta del percorso in cui avviene lo *storage* di ogni *file* immesso nell'applicazione *web*, sono state personalizzate le modalità di salvataggio dei *file* utilizzando cartelle con percorso determinato dall'ambiente di lavoro in cui essi erano utilizzati: in questo modo è stato possibile fin da subito separare i video utilizzati per i test da quelli utilizzati internamente allo sviluppo, e l'identificazione è stata resa immediata grazie alla presenza di una cartella dedicata per ognuno dei tre ambienti messi a disposizione dal *framework*.

L'usabilità

Durante lo sviluppo, sono state spese diverse ore allo scopo di rendere l'interfaccia grafica sviluppata per l'applicazione *web* facilmente utilizzabile. Alcuni colleghi del reparto tecnico con esperienza pregressa relativamente alle *best-practices* da impiegare per migliorare l'usabilità di un sito *web*, infatti, mi hanno dato più volte pareri e consigli, grazie ai quali è stato possibile da parte mia, all'interno di un'iterazione dedicata, effettuare un intervento di riorganizzazione del flusso delle informazioni accessibili attraverso la piattaforma *SaaS*, allo scopo di renderla più usabile. Questo si è tradotto principalmente nell'implementazione di un'interfaccia di visualizzazione dei contenuti attraverso un *layout* a griglia, visibile in figura 3.8, che risulta preferibile al *layout* a tabelle poiché più intuitivo. All'interno della piattaforma è stata comunque lasciata la possibilità agli utenti di passare in qualunque momento al *layout* a tabelle, in caso esso risulti per loro di maggiore gradimento, come visibile in figura 3.7.

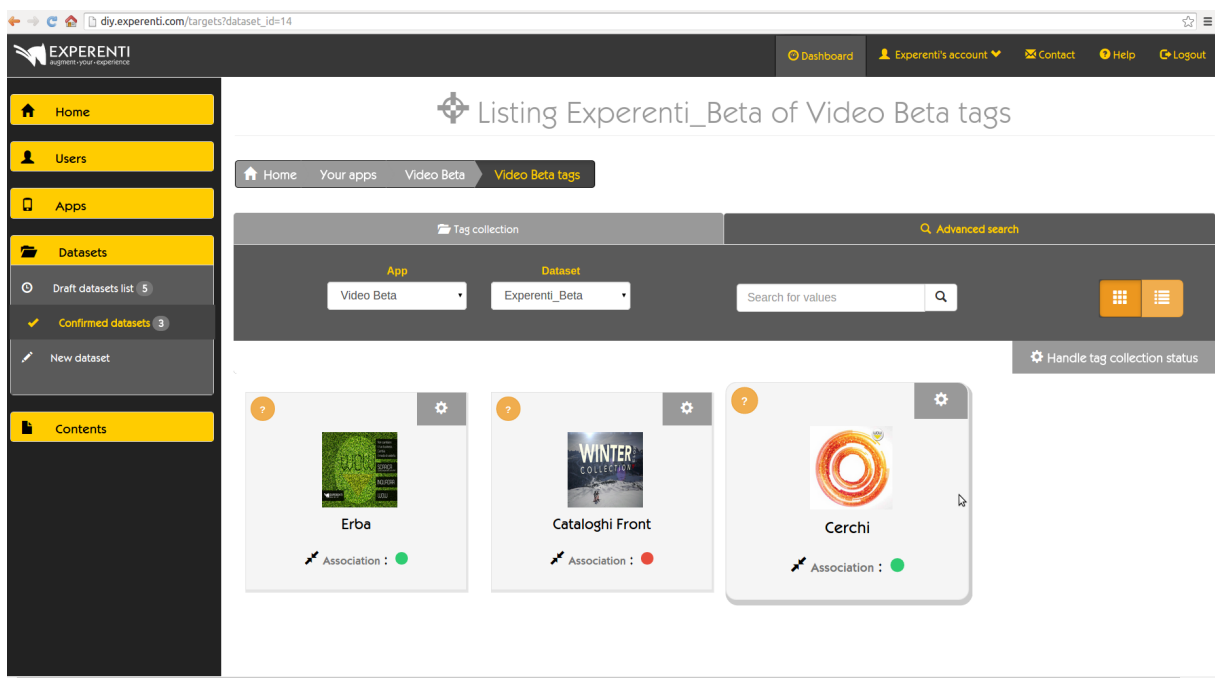


The screenshot shows the 'Listing Experenti_Beta of Video Beta tags' page. The interface includes a sidebar with navigation options: Home, Users, Apps, Datasets, and Contents. The main content area displays a table of tags with columns: App, Dataset, Name, Category, Width, Height, Content, and Action. The table lists three tags: 'erba', 'cataloghi_front', and 'cerchi', all associated with 'Video Beta' and 'Experenti_Beta'.

	App	Dataset	Name	Category	Width	Height	Content	Action
●	Video Beta	Experenti_Beta	erba	ImageTarget	100.0	70.7812	video_test_1.mp4	Action ▾
●	Video Beta	Experenti_Beta	cataloghi_front	ImageTarget	100.0	56.3346	Associate content	Action ▾
●	Video Beta	Experenti_Beta	cerchi	ImageTarget	100.0	100.098	video_test_2.mp4	Action ▾

Displaying all 3 Target

figura 3.7: Schermata di visualizzazione tag con layout a tabelle in *Experenti Self Service*



The screenshot shows the same 'Listing Experenti_Beta of Video Beta tags' page, but with a grid layout. The tags are displayed as cards with images and labels: 'Erba', 'Cataloghi Front', and 'Cerchi'. Each card includes an 'Association' status indicator (green dot for 'Erba' and 'Cerchi', red dot for 'Cataloghi Front').

figura 3.8: Schermata di visualizzazione tag con layout a griglia in *Experenti Self Service*

3.3.3 Verifica e validazione

Contestualmente all'applicazione sviluppata, i principali test svolti hanno riguardato il corretto funzionamento relativo all'inserimento di dati nella piattaforma ed al loro relativo recupero tramite le *API* sviluppate. L'attività di verifica e validazione è stata svolta quasi totalmente da alcuni colleghi del reparto tecnico, che in questo modo hanno potuto valutare, durante l'intero svolgimento dello stage, la correttezza dei componenti da me implementati, evitando conflitti di interesse: tali test hanno avuto quasi sempre esito positivo, ma in alcuni casi è stato necessario da parte mia rivedere il modo in cui avevo implementato parte delle funzionalità richieste, effettuando quindi degli interventi correttivi allo scopo di risolvere i problemi sollevati dagli esiti di tali test. Questo talvolta si è tradotto in un semplice intervento sul codice relativo alle componenti sviluppate, mentre altre volte è stato fatto attraverso il meccanismo delle migrazioni interno a *Ruby on Rails*, che mi ha permesso di andare ad operare delle correzioni relative allo schema del *database* associato all'applicazione. In qualunque caso, le criticità sollevate dall'esito dei test svolti sono sempre state risolte all'interno dell'iterazione immediatamente successiva a quella in cui mi è stato riferito il problema che richiedeva un intervento correttivo.

Il mio contributo in quest'attività ha riguardato, quindi, principalmente lo svolgimento di un'analisi statica per verificare che i requisiti inizialmente elaborati relativi alla sicurezza dell'applicazione *web* fossero soddisfatti (con conseguente ricerca degli strumenti più adatti per il raggiungimento di tale scopo).

Verifica della sicurezza

In previsione di un *deploy* dell'applicazione *web* realizzata e vista la delicatezza dei dati trattati, particolare attenzione è stata posta al fattore sicurezza. Grazie alla moltitudine di gemme dedicate disponibili per *Ruby on Rails*, è stato possibile effettuare test di analisi statica allo scopo di rilevare eventuali falle di sicurezza nel sistema. Nello specifico, è stata utilizzata la gemma *Brakeman*, la quale permette di svolgere un'analisi completa dell'applicazione e di generare un *report* contenente il dettaglio sui test svolti, sulle eventuali problematiche rilevate e sulle possibili soluzioni.

I rischi da cui mette in guardia tale *plugin* comprendono, tra i molti:

- *SQL Injection*, ovvero quando un utente risulta in grado di manipolare un valore gestito in maniera non protetta all'interno di una *query SQL*;
- *Command Injection*, si verifica quando un utente risulta in grado di manipolare un valore gestito in maniera non protetta all'interno di un comando *shell*;
- *Cross-Site Request Forgery*, una debolezza che permette ad un eventuale intruso di svolgere azioni all'interno di un sito *web* come fosse un utente autenticato;
- *Unsafe Redirects*, ovvero il caso in cui delle funzionalità di *redirect* dipendano in maniera diretta da valori manipolabili dagli utenti, utilizzabile per camuffare dei normali *link* per farli condurre a siti malevoli;

- *File Access*, debolezza che si verifica quando l'accesso ad alcuni file viene regolato attraverso parametri inseriti dagli utenti, con il potenziale rischio di permettere loro di accedere a qualunque file presente sulla macchina ospitante.

Brakeman Report				
Application Path	Rails Version	Brakeman Version	Report Time	Checks Performed
C:/row/dev/experenti_saas	4.1.6	2.6.3	2014-11-01 10:51:07 +0100 2.906259 seconds	BasicAuth, ContentTag, CreateWith, CrossSiteScripting, DefaultRoutes, Deserialize, DetailedExceptions, DigestDoS, EscapeFunction, Evaluation, Execute, FileAccess, FilterSkipping, ForgerySetting, HeaderDoS, I18nXSS, JRubyXML, JSONParsing, LinkTo, LinkToHref, MailTo, MassAssignment, ModelAttrAccessible, ModelAttributes, ModelSerialize, NestedAttributes, NumberToCurrency, QuoteTableName, Redirect, RegexDoS, Render, RenderDoS, ResponseSplitting, SQL, SQLCVEs, SSLVerify, SafeBufferManipulation, SanitizeMethods, SelectTag, SelectVulnerability, Send, SendFile, SessionSettings, SimpleFormat, SingleQuotes, SkipBeforeFilter, StripTags, SymbolDoS, TranslateBug, UnsafeReflection, ValidationRegex, WithoutProtection, YAMLParsing

Summary

Scanned/Reported	Total
Controllers	20
Models	7
Templates	51
Errors	0
Security Warnings	15 (0)
Ignored Warnings	0

Warning Type	Total
File Access	8
SQL Injection	7

Security Warnings

Confidence	Class	Method	Warning Type	Message
Medium	API::V1::Android::ContentsController	download	File Access	Model attribute used in file name near line 12: send_file(Content.find(params[:id]).data.path, :filen...
Medium	API::V1::Android::ContentsController	download_keyframe	File Access	Model attribute used in file name near line 16: send_file(Video.where(:content_id => (Content.find(pa...
Medium	API::V1::Android::DatasetsController	download_xml	File Access	Model attribute used in file name near line 19: send_file(Dataset.find(params[:id]).xml.path, :filena...
Medium	API::V1::Android::DatasetsController	download_dat	File Access	Model attribute used in file name near line 23: send_file(Dataset.find(params[:id]).dat.path, :filena...
Medium	API::V1::Ios::ContentsController	download	File Access	Model attribute used in file name near line 10: send_file(Content.find(params[:id]).data.path, :filen...
			File	Model attribute used in file name near line 14:

figura 3.9: Schermata relativa al report di sicurezza generato dalla gemma *Brakeman*

Come visibile in figura 3.9, sono state impiegate diverse ore per risolvere tutti i problemi di sicurezza legati ai *warning* con potenziale rischio categorizzato come *High* (lo 0 scritto in colore rosso indica che sono stati risolti tutti). Al momento della consegna del prototipo risultavano irrisolti solo alcuni *warning* minori, il cui potenziale rischio associato è stato valutato come irrisorio ai fini dell'applicazione in sviluppo.

Capitolo 4

Valutazioni retrospettive

4.1 Obiettivi raggiunti

Come già detto, parte degli obiettivi è cambiata nel periodo immediatamente antecedente lo stage, sulla base di una rivalutazione delle priorità dell'azienda in relazione al *software* in sviluppo. Nonostante questo, vi è da dire che tali obiettivi rappresentavano una minoranza rispetto all'insieme generale stabilito al momento della redazione del documento di piano di lavoro.

Gli obiettivi iniziali sono stati quindi raggiunti, ed i requisiti imposti per *Experienti Self Service* soddisfatti, in quanto tutte le componenti richieste sono state sviluppate, fatta eccezione per gli obiettivi relativi alla realizzazione di un meccanismo di registrazione di statistiche sui contenuti fruiti in realtà aumentata e per gli obiettivi relativi alla realizzazione di quanto necessario alla gestione dei contenuti 3D all'interno della piattaforma. Per come è stata strutturata l'applicazione, però, l'implementazione dei componenti necessari ad offrire tali funzionalità sarà facilmente attuabile già a partire dalle prossime iterazioni che interesseranno il *software* sviluppato, soprattutto grazie ai sopracitati meccanismi di migrazione offerti dal *framework Ruby on Rails*.

Il prototipo realizzato è piaciuto molto all'azienda, ed un apprezzamento particolare è stato espresso per l'attenzione dedicata allo sviluppo dell'interfaccia grafica dell'applicazione *web*.

4.2 Conoscenze acquisite

Uno degli aspetti che più ho apprezzato relativamente a questa esperienza di stage, è stata la equa suddivisione tra periodo di formazione e periodo di svolgimento pratico delle attività. Entrambi i periodi hanno avuto la durata più consona a fare in modo che la mia preparazione relativa alle tecnologie coinvolte non fosse né troppo teorica, né troppo pratica. Inoltre, come già detto, la formazione non è stata rivolta semplicemente alle tecnologie che avrei dovuto impiegare direttamente nel progetto, ma anche a quelle indirettamente coinvolte: questo mi ha dato una visione più completa relativamente al processo di sviluppo aziendale e mi ha permesso di ampliare fortemente il mio bagaglio culturale.

Nello specifico, questo stage mi ha permesso di apprendere diversi aspetti legati a tecnologie a me totalmente sconosciute quali:

- *Ruby on Rails*, la tecnologia che mi ha coinvolto e interessato di più. Non avrei mai creduto fosse possibile sviluppare in maniera così innovativa un'applicazione *web*, attenendosi con facilità ad architetture quali *MVC* e *REST* ma riuscendo a mantenere un controllo totale sul grado di accessibilità della piattaforma; avendo a disposizione una quantità così vasta di *utility* ma gestendone comunque con molta semplicità l'aggiunta di funzionalità esterne. Nonostante alcune difficoltà iniziali dovute alla sintassi non sempre intuitiva, sono riuscito a svolgere le attività che lo hanno coinvolto senza particolari intoppi;
- *Bootstrap*, un altro *framework* la cui semplicità d'uso mi ha sorpreso molto fin dal momento in cui ho effettuato uno studio su di esso per confrontarlo a *Polymer*, in quanto totalmente estraneo alla materia. Una volta capiti i meccanismi fondamentali, ne ho fatto un forte uso per personalizzare l'interfaccia grafica dell'applicazione *web*, la quale ne ha guadagnato in termini di usabilità;
- *Unity*, nonostante sia stato necessario investire diverse ore per comprenderne le dinamiche di base, si è rivelato uno strumento estremamente interessante con cui lavorare. Sono state chiare fin da subito le grandi potenzialità di questo applicativo;
- *Vuforia*, grazie all'estensione per *Unity*, lavorare con questo *SDK* è stato più semplice. L'attenzione posta in questo studio è stata ancora maggiore in quanto di diretto interesse per le dinamiche coinvolte nell'applicazione *web* che è stata sviluppata;
- *C#*, lo *scripting* relativo alle componenti interne a *Unity* è stato fatto avvalendosi di questo linguaggio. La trattazione è stata troppo sommaria per apprenderne gli aspetti più profondi, ma è stato comunque interessante capirne le basi e realizzare esempi pratici.

Sono uscito da questo stage fortemente motivato ad approfondire quanto appreso relativamente ad ognuna di queste tecnologie, ma con un'attenzione particolare per *Ruby on Rails* per lo sviluppo di architetture *REST* in quanto credo fortemente nelle potenzialità di questo *framework*.

Naturalmente, oltre al contesto tecnologico questo stage mi ha arricchito anche di altre concetti quali, in primo luogo, l'importanza cruciale di una corretta pianificazione delle attività, di uno studio approfondito svolto a priori per la scelta delle tecnologie da impiegare in un progetto e dell'esperienza in sé come fattore fondamentale per l'apprendimento di tecnologie e metodologie.

4.3 Valutazione personale

Come già detto, le tecnologie che ho utilizzato contestualmente allo stage mi erano totalmente estranee, e non avrebbero potuto essere incluse in nessuno degli insegnamenti interni al Corso di Laurea in Informatica, in quanto in parte esterne alle finalità del corso di studi, ed in parte di introduzione troppo recente. Credo però fortemente che un'infarinatura all'interno del Corso di Laurea Triennale relativamente a concetti quali le architetture *REST* sarebbe di grande aiuto per gli studenti che devono affrontare degli stage di questo tipo. Nonostante questo però, le nozioni e le metodologie di lavoro apprese nei corsi di Ingegneria del Software, Basi di dati, Tecnologie web e Programmazione ad oggetti si sono rivelate fondamentali perché lo stage si svolgesse al meglio, sia per gli aspetti legati al periodo di formazione, sia per quanto attinente alla realizzazione del *software* richiesto.

In generale, dopo aver affrontato questa esperienza, posso confermare la mia soddisfazione nell'aver scelto di frequentare questo Corso di Laurea, in quanto anche a stage terminato continuo a pensare che sia strutturato in maniera davvero ottima: forse le nozioni interne ad una minoranza degli insegnamenti necessiterebbero di un aggiornamento, ma in linea generale reputo invidiabile la formazione che questo Corso di Laurea offre rispetto ad altri atenei, in quanto vi è davvero un ottimo bilanciamento tra l'aspetto teorico e l'aspetto pratico, con un forte orientamento verso il mondo del lavoro.

Per quanto mi riguarda, vivere un'esperienza di stage come quella descritta in questa relazione prima di aver affrontato gli insegnamenti di questo Corso di Laurea sarebbe stato estremamente complicato, se non impensabile, perché sarei stato privo della base di conoscenze, concetti e metodi di lavoro che esso offre, e che mi ha permesso di svolgere al meglio le attività assegnatemi. Lo stage mi ha infatti dato la possibilità di mettere in pratica i principi ed i meccanismi appresi durante gli anni del Corso di Laurea, e trovo che esso sia davvero un passo fondamentale da svolgere a chiusura del proprio percorso di studi, in quanto senza uno stage di questo tipo è impossibile effettuare un confronto tra quello che è il mondo universitario e quello lavorativo.

Concludo sottolineando che, a mio parere, l'insegnamento più importante che questo corso di studi lascia a chi lo porta a termine è la capacità di auto-apprendere: corsi come Ingegneria del Software, dove si vive una vera esperienza prolungata di lavoro in *team* con tutte le fatiche annesse, formano non solo dal punto di vista nozionistico, ma anche da quello relativo alla maturazione personale e informatica, specialmente riguardo a compiti come lo svolgimento di un corretto studio delle tecnologie necessarie alla realizzazione di un progetto.

4.4 Screenshot finali

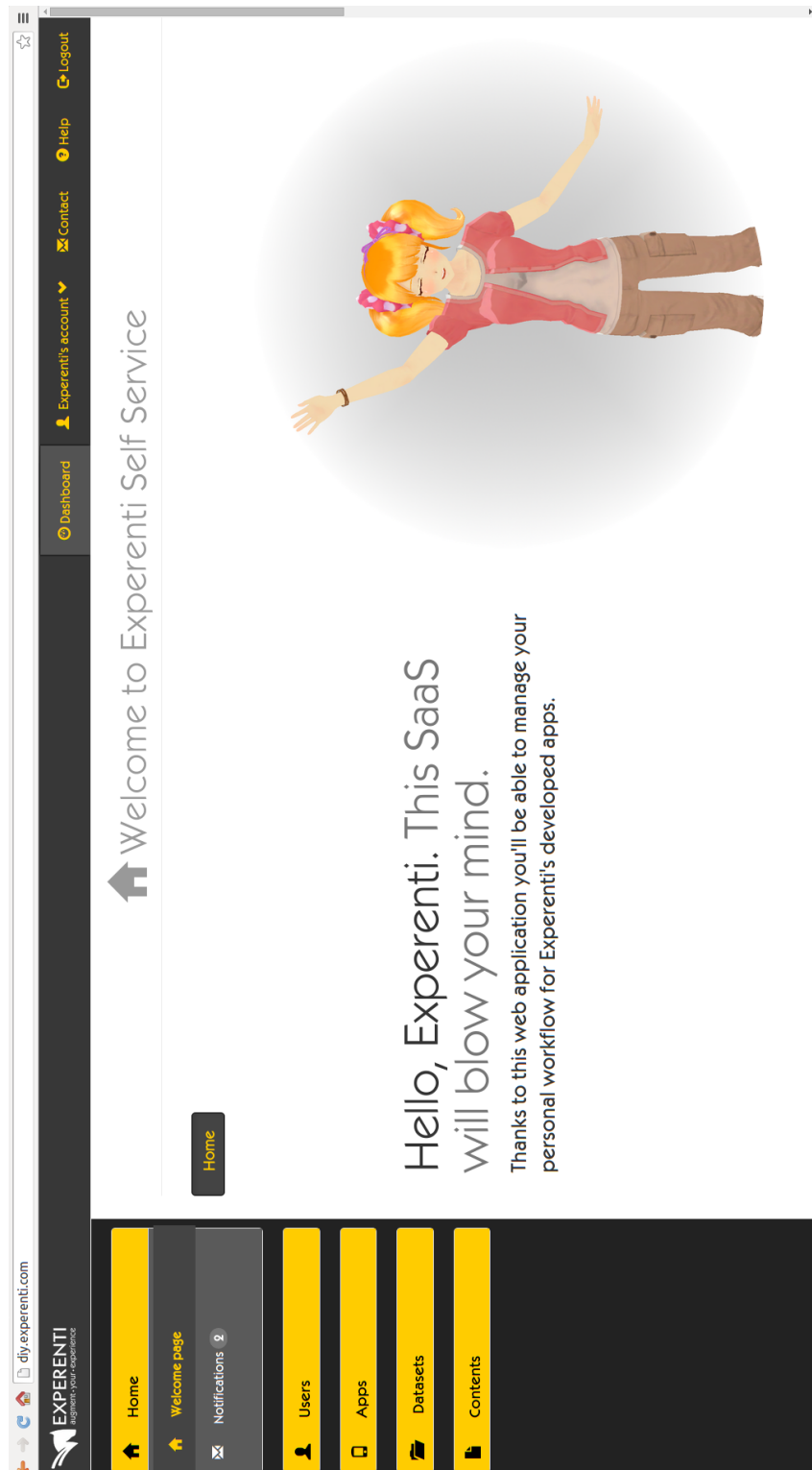


figura 4.1: Schermata di *Home* di *Experenti Self Service*

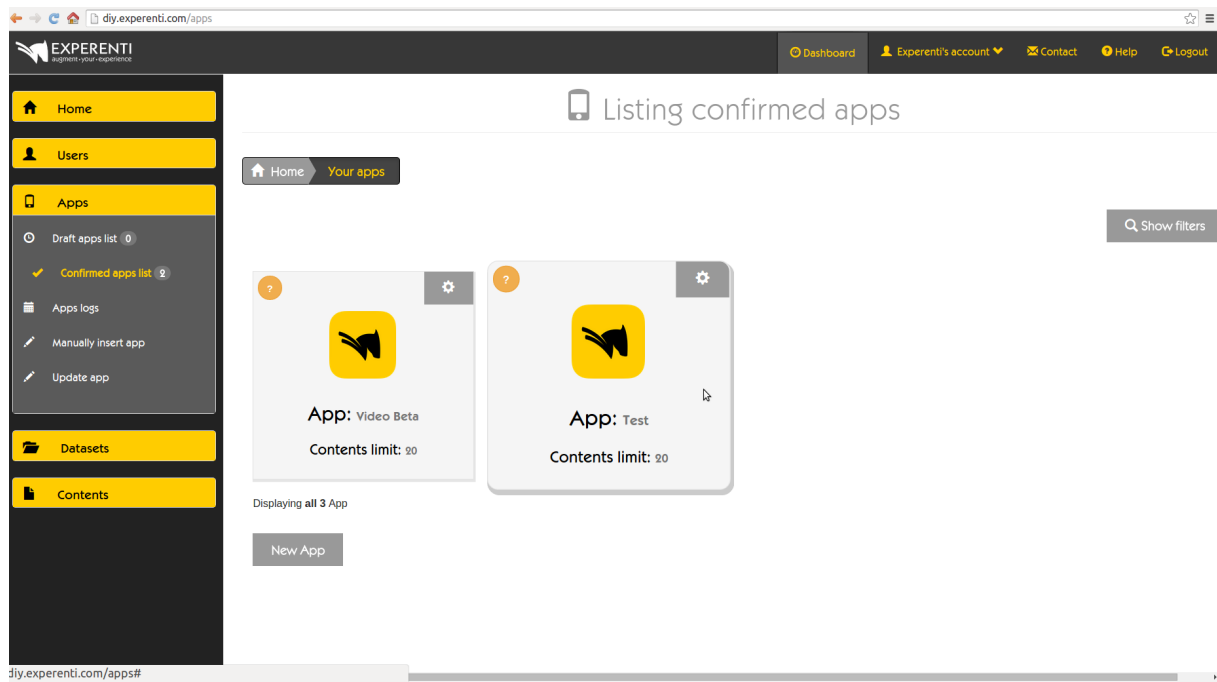


figura 4.2: Schermata relativa alle *app* di un cliente in *Experenti Self Service*

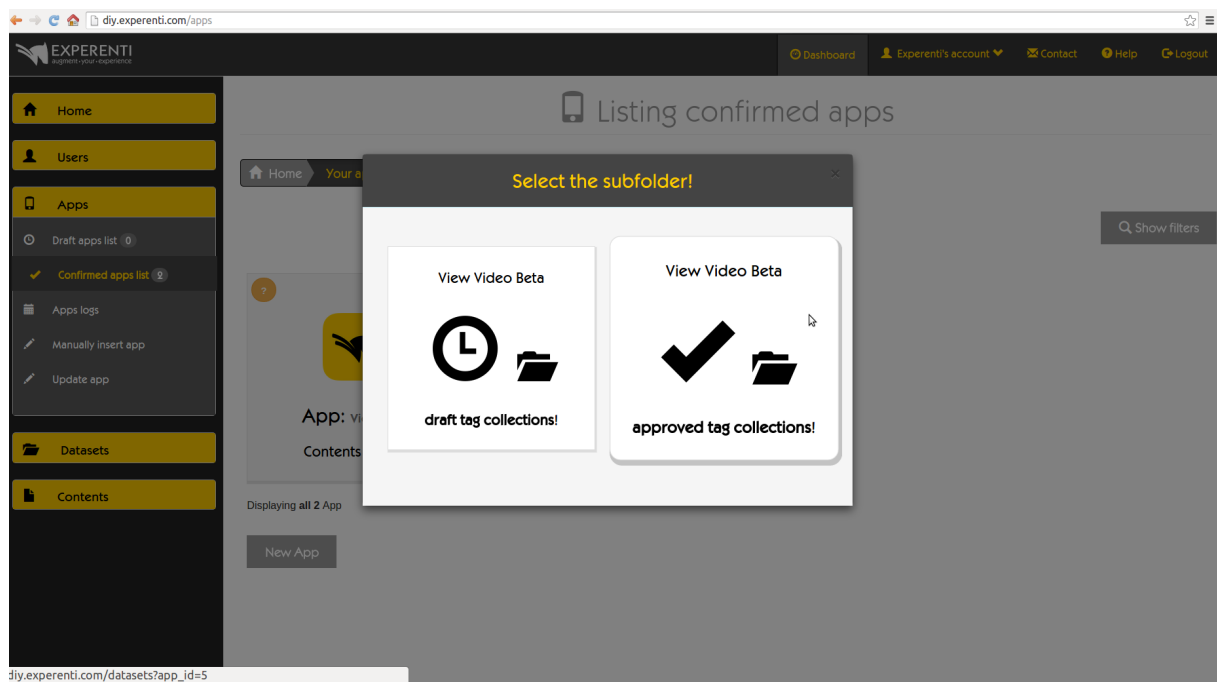


figura 4.3: Schermata relativa all'accesso alle sezioni dei *dataset* temporanei e ufficiali in *Experenti Self Service*

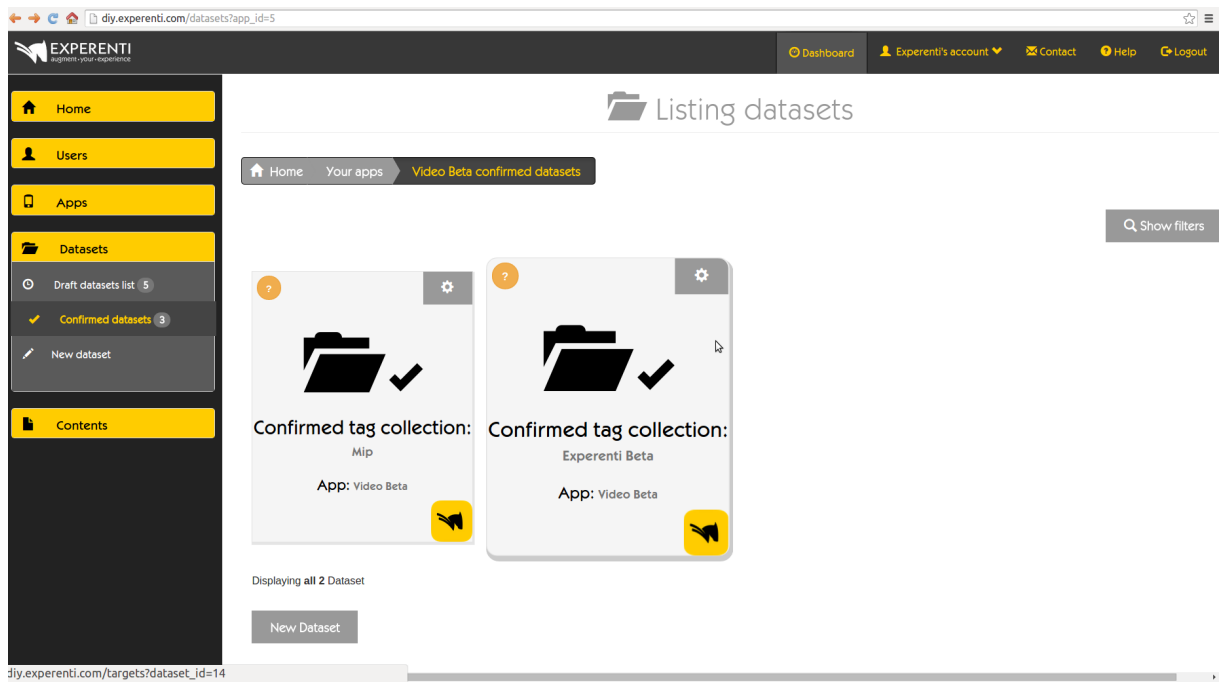


figura 4.4: Schermata relativa ai *dataset* che hanno ricevuto conferma in *Experenti Self Service*

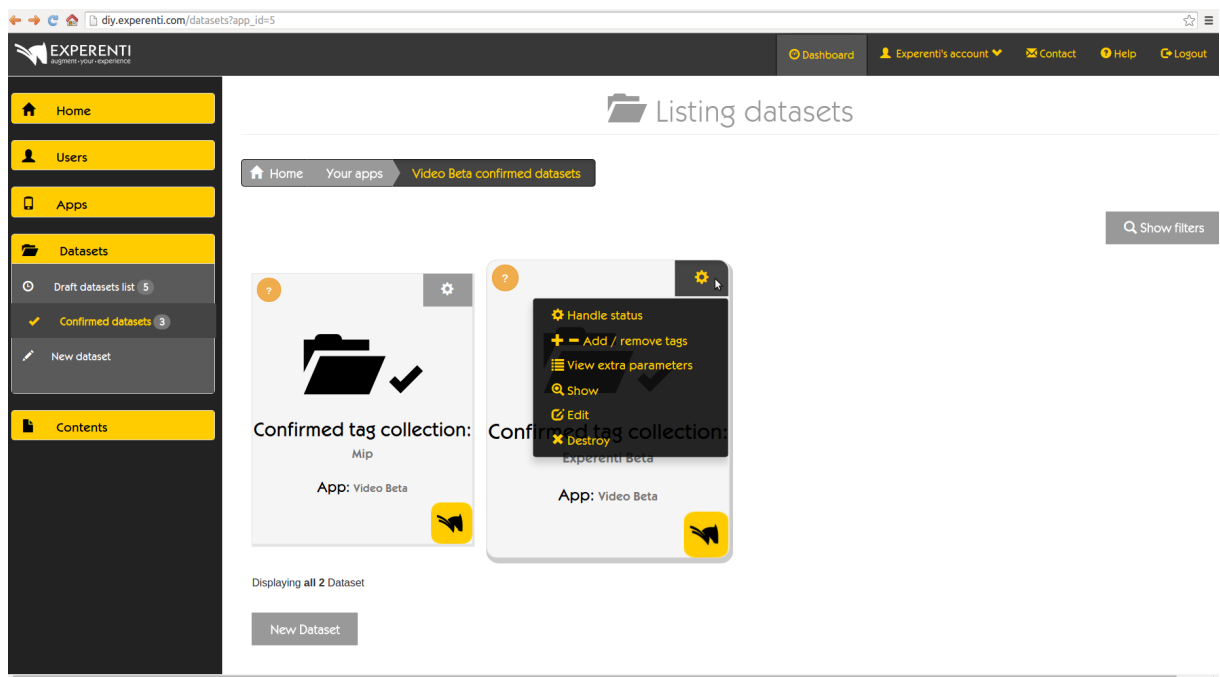


figura 4.5: Schermata relativa alle opzioni di un *dataset* in *Experenti Self Service*

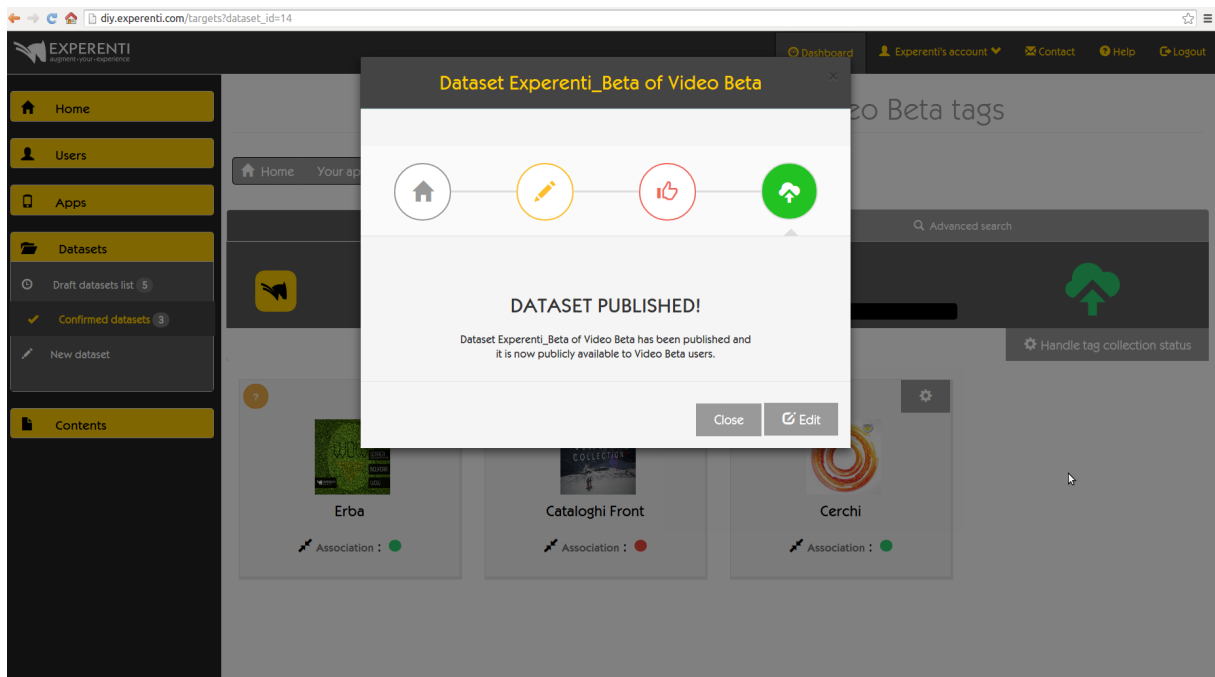


figura 4.6: Schermata relativa ai passi di approvazione di un *dataset* in *Experenti Self Service*

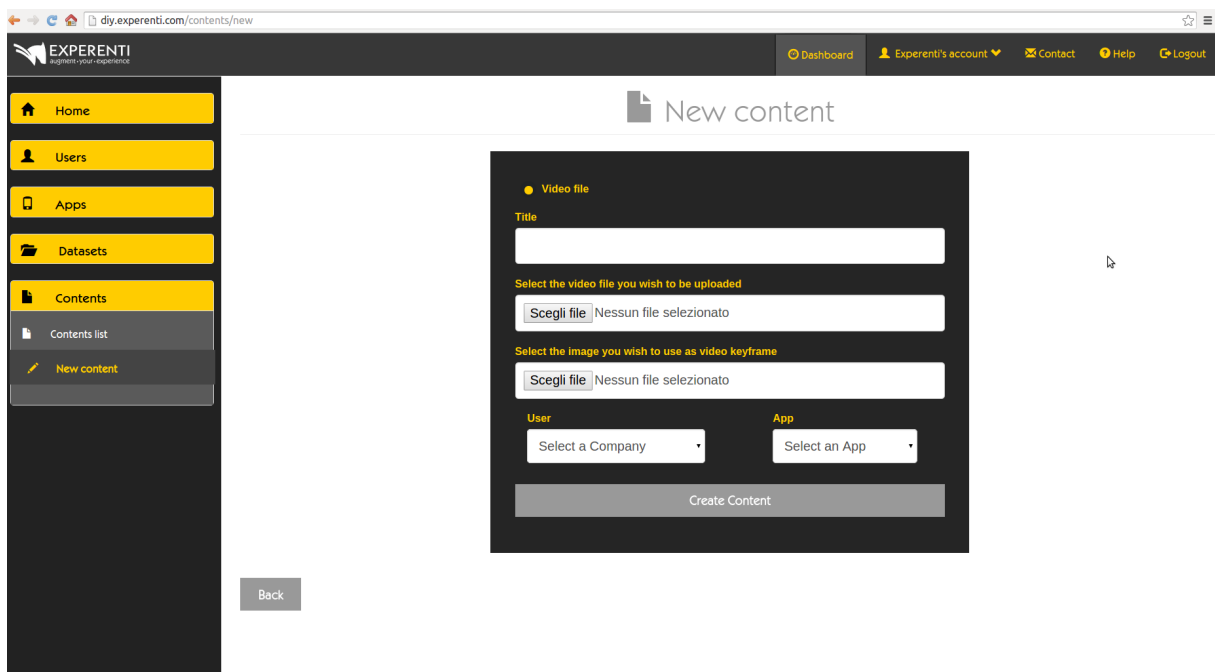


figura 4.7: Schermata relativa all'inserimento di un video in *Experenti Self Service*

Appendice A

La realtà aumentata

La realtà aumentata consiste nell'arricchimento di quanto viene percepito dalla sfera sensoriale delle persone con informazioni virtuali di svariate origini e forme. Tali informazioni, che solitamente non sarebbero percepibili attraverso i cinque sensi, vengono manipolate elettronicamente, per poi essere sovrapposte alla realtà attraverso l'utilizzo di particolari dispositivi quali *smartphone*, *tablet*, visori e *pc* dotati di *webcam*: questo è dovuto al fatto che un *software* di realtà aumentata ha bisogno di diversi sensori per ricavare informazioni dalla realtà in cui l'utente è immerso. Il punto di forza della realtà aumentata è il fatto che essa permette all'utilizzatore che accede alle informazioni extra di continuare ad avere la completa percezione degli altri suoi sensi, godendo quindi di un arricchimento "puro".



figura A.1: Esempio di realtà aumentata interno all'app Experienti

Le informazioni che “aumentano” la realtà possono essere presenti nella memoria del dispositivo utilizzato, oppure possono essere ricavate da internet in tempo reale, prima di essere rese disponibili attraverso il *display* (o gli altoparlanti, in una minoranza dei casi) del dispositivo utilizzato. Questa tecnologia ha raggiunto l’utilizzo da parte del grande pubblico nel 2009, grazie al notevole miglioramento delle tecnologie impiegate nel settore *mobile*, ma è presente in letteratura già a partire dagli anni Quaranta del Novecento. I contesti in cui è possibile l’applicazione di questa tecnologia sono davvero tantissimi e spaziano dall’arte all’archeologia, dall’edilizia al *gaming*, dall’istruzione alle traduzioni in *real-time* e molti altri. La rapidissima espansione che sta interessando la realtà aumentata e le risorse *hardware* ad essa legate farà sì che sempre più aspetti della nostra vita quotidiana si legheranno a questa tecnologia, permettendoci di accedere in qualunque momento ad una varietà pressoché illimitata di informazioni virtuali come se facessero parte della realtà di tutti i giorni.

Appendice B

Glossario

A

API: nell'ambito informatico, l'acronimo *API* sta per *Application Programming Interface*, ed indica principalmente un'insieme di procedure disponibili in *set* atte alla realizzazione di un determinato compito all'interno di un programma. Questo termine viene utilizzato anche per indicare le librerie *software* che un linguaggio di programmazione rende disponibili.

App mobile: è una variante delle applicazioni (in contesto informatico), la cui esecuzione avviene esclusivamente all'interno di dispositivi *mobile* quali *smartphone* e *tablet*.

Applicazione web: indica un'applicazione informatica accessibile, per mezzo di un *network*, attraverso il canale *web*.

Augmented Reality: vedi realtà aumentata.

B

Best-practices: generalmente, si intendono le esperienze più significative e/o quelle che hanno permesso di ottenere i migliori risultati.

C

Conflitto di interesse: indica una situazione che si verifica quando viene data una responsabilità ad una persona avente interessi di varia natura in conflitto con l'imparzialità richiesta dal ruolo.

Cross-platform: è la possibilità per un software di funzionare correttamente su più sistemi operativi diversi.

CRUD: nel contesto informatico, indica le operazioni *create*, *read*, *update* e *delete*, ovvero le quattro funzioni fondamentali nell'ambito dello *storage* persistente dei dati.

D

Database: indica una collezione di informazioni avente una struttura tale da renderle facilmente gestibili ed aggiornabili.

Dataset: nel contesto della realtà aumentata, si tratta di un archivio di *tag*.

DBMS: indica un *Database Management System*, ovvero un sistema atto alla gestione di basi di dati;

Design Pattern: è un modello di progettazione che descrive un problema ricorrente e la soluzione a tale problema.

Deploy: indica la messa in esercizio di un *software* con relativa consegna o rilascio al cliente.

E

ERP: sta per *enterprise resource planning*, ed indica un sistema informativo in grado di gestire molti dei processi di *business* rilevanti in ambito aziendale.

F

Framework: è uno strumento a supporto della progettazione di un *software* che semplifica il lavoro degli sviluppatori.

Front-end: indica una parte di un *software* atta all'interfacciamento con gli utenti che lo utilizzano ed all'acquisizione dei dati in ingresso.

G

Gantt: un diagramma di Gantt è uno strumento utile alla pianificazione dei tempi di realizzazione di un progetto ed al loro monitoraggio. In esso le attività seguono un ordine preciso.

GUI (Graphical User Interface): è l'interfaccia grafica di un'applicazione, ovvero un insieme di elementi e strutture che consente agli utenti di interagire con l'applicativo.

I

IDE: sta per *Integrated Development Enviroment* ed indica un ambiente di sviluppo integrato, ovvero un *software* utilizzato dai programmatori per sviluppare il codice sorgente di un programma.

M

Marketing esperenziale: è un tipo di *marketing* che si basa più sulle esperienze di consumo che sul valore d'uso dei prodotti, e che punta alla creazione di valore attraverso l'esperienza che permette all'utente di vivere.

MVC: sta per *Model-View-Controller*, un *pattern* architetturale molto utilizzato nel contesto dell'Ingegneria del *software*.

O

Open-source: indica un *software* i cui detentori dei diritti ne permettono l'apporto di modifiche da parte di altri sviluppatori indipendenti.

P

Plugin: nel settore informatico, indica un *software* che interagisce con un altro programma allo scopo di ampliare la gamma di funzionalità da esso offerte.

R

Realtà aumentata: indica l'arricchimento di quanto viene percepito dalla sfera sensoriale delle persone con informazioni virtuali di svariate origini e forme attraverso dispositivi quali, ad esempio, *smartphone* e *tablet*.

Reseller: è un individuo che acquista prodotti o servizi presso un fornitore per poi rivenderli.

Responsive design: è una tecnica di *web design* per la realizzazione di siti in grado di offrire una visualizzazione grafica corretta qualunque sia il dispositivo con il quale vengono visualizzati;

REST: sta per *REpresentational State Transfer*, ed è un tipo di architettura *software* pensato per i sistemi di ipertesto distribuiti, di cui l'esempio più famoso è il *World Wide Web*.

S

SaaS: sta per *Software as a service*, ed indica un servizio di *cloud computing* basato su un modello di distribuzione del *software* dove viene sviluppata e gestita un'applicazione *web*, che viene messa a disposizione dei propri clienti attraverso internet.

SDK: sta per *software development kit* e, nell'ambito informatico, indica un insieme di strumenti atti allo sviluppo ed alla documentazione di *software*.

Startup: è un termine che indica un'azienda nel primo periodo della sua vita.

U

URL: sta per *Uniform Resource Locator*, e corrisponde ad una serie di caratteri che identifica l'indirizzo di una risorsa in internet in modo univoco.

W

Web application: vedi “Applicazione *web*”.

Bibliografia

Libri

- Sam Ruby, Dave Thomas, David Heinemeier Hansson (2013), *Agile Web Development with Rails 4, First Edition*, Pragmatic Bookshelf

Siti web consultati

1. Rails Guides, <http://guides.rubyonrails.org/index.html>
2. Bootstrap overview, <http://getbootstrap.com/getting-started/>
3. Unity Manual, <http://docs.unity3d.com/Manual/index.html>
4. Vuforia Developer Portal, <https://developer.vuforia.com/>