UNIVERSITÀ DEGLI STUDI DI TRENTO
FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI


UNIVERSITY OF TRENTO - Italy

Corso di Laurea in INFORMATICA

---

Elaborato Finale

# WHAT'S UP WEB: ENABLING INTERGENERATIONAL SHARING THROUGH THE WEB

Relatore:

Prof.
Maurizio Marchese

Laureando:

Massimiliano Battan

Correlatore:

Cristhian Daniel Parra Trepowski

ANNO ACCADEMICO 2011 - 2012

# Indice

# Capitolo 1

# Introduzione

Negli ultimi anni lo straripante progresso tecnologico nel campo sociale, vedi l'affermazione mondiale di alcuni social network e la continua nascita di nuovi, ha dato luogo a nuove opportunità. Una di queste era la possibilità di mettere a disposizione le tecnologie dell'informazione agli anziani, una fascia generazionale che come sappiamo non ha avuto la possibilità di nascere e crescere in pieno sviluppo dell'informatica bensì si è trovata a doverla fronteggiare in modo brusco e avventato. E' risaputo che, invecchiando ogni persona deve fronteggiare parecchi cambiamenti sia fisici che psichici che portano l'individuo ad affrontare la vita e le proprie esperienze in modo diverso, sicuramente più limitato e meno coinvolgente. Le interazioni sociali con amici e parenti diminuiscono, spesso riducendosi a saltuarie chiamate e visite occasionali. In questo contesto si inserisce il progetto What's up[1], una piattaforma di comunicazione che favorisce la divulgazione intergenerazionale di contenuti (messaggi, foto, video) tramite l'utilizzo di applicazioni mobile. Basandosi su di una comune infrastruttura il progetto What's up è composto di due applicazioni principali:

- **What's up LifeShare** sviluppata per smartphone Android ed Apple, utilizzata dai giovani per condividere foto, catturate o presenti sul proprio smartphone, nonché messaggi.
- **What's up Display** sviluppata per iPad, utilizzata dagli anziani come display di tutti i messaggi e foto ricevute.

Il lavoro svolto all'interno di questo progetto che verrà descritto nel trattato nasce dall'esigenza di voler portare queste funzionalità sul web. Il world wide web offre ormai incredibili opportunità di sviluppo, grazie all'espansione di tecnologie atte al dynamic

[1] What's Up fa parte di uno dei progetti di Life Participation, un team di designers e sviluppatori che propongono soluzioni atte a migliorare il benessere emotivo dell'individuo sfruttando le più moderne tecnologie in campo mobile e web

web e alle eccezionali prestazioni e funzionalità dei recenti browser.
L'obbiettivo era quello di sviluppare un applicazione web in grado di fornire entrambi i principi alla base di What's up: Share & Display. Per quanto riguarda Share, è nata la volontà di voler creare un'applicazione web in grado di offrire ai giovani l'opportunità di condividere con i propri parenti anziani le foto presenti sul social network più famoso ed utilizzato al mondo[2]: *Facebook*[3].
L'applicazione web Display invece avrebbe dovuto offrire le medesime funzionalità del Display già sviluppato per IPad, cosi da permettere agli anziani la visualizzazione dei propri contenuti senza dover necessariamente disporre del tablet targato Apple. Il tutto per garantire un'alta portabilità all'intera piattaforma.
Nel secondo capitolo verranno presentate le tecnologie utilizzate durante lo sviluppo di What's Up WEB, mentre nel terzo i requisiti che hanno dato vita al progetto.
Il quarto capitolo si occupa di presentare l'architettura utilizzata per lo sviluppo dell'applicazione, infine il quinto capitolo analizza il risultato finale, mostrando inoltre le difficoltà affrontate e le soluzioni adottate.

[2]Analisi effettuata daAlexa è, azienda leader nell'analisi di siti web
[3]Facebook.com

# Capitolo 2

# Stato dell'arte

Partiamo nel descrivere le tecnologie utilizzate nello sviluppo dal principio, Internet. Come riportato da wikipedia, Internet viene così definito:

> *The Internet is a global system of interconnected computer networks that interchange data by packet switching using the standardized Internet Protocol Suite (TCP/IP).*

Internet attualmente rappresenta il mezzo di comunicazione di massa più diffuso al mondo nonché la più grande rete di computer attualmente esistente e come tale offre innumerevoli servizi, in continua evoluzione[1]. Uno di questi, alla base del mio progetto, è il World Wide Web (WEB), nato nel 1991 grazie alla prima versione effettivamente disponibile del protocollo HTTP(v 1.0), un sistema per la trasmissione d'informazione sul web tramite la lettura ipertestuale, non-sequenziale dei documenti. Il World Wide Web Consortium (W3C)[2], la community internazionale per lo sviluppo degli standard Web, in continua evoluzione per permetterne la crescita, utilizza questa definizione[3]:

Figura 2.1: Il primo logo del Web, progettato da Robert Cailliau

[1] http://it.wikipedia.org/wiki/Internet
[2] World Wide Web Consortium Website
[3] Definizione ricavata dal trattato sul web consultabile qui

> *The World Wide Web (WWW, or simply Web) is an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (URI).*

Come si legge, il Web è uno spazio di informazione, che permette all'utente che ne fa uso di navigare, tramite Browser, ed usufruire di un innumerevole quantità di contenuti e servizi.
Le prime tre specifiche per le tecnologie Web sono le seguenti:

- URLs (Uniform Resource Locator): un Url non è altro che una semplice sequenza di caratteri necessaria per identificare in modo univoco una risorsa presente in Internet. Attraverso un browser, chiunque a partire da un Url può richiedere l'accesso ad una determinata risorsa se presente.

- HTTP (HyperText Transfer Protocol): il protocollo d'Applicazione per la comunicazione su Web.

- HTML (HyperText Markup Language): il principale linguaggio markup per la visualizzazione di pagine web ed altre informazioni, fruibili attraverso un web Browser.

Una delle maggiori limitazioni iniziali del Web era quella di essere troppo statico; infatti l'utente poteva soltanto visualizzare ed esplorare ogni risorsa disponibile senza però avere in alcun modo la possibilità di interagire con essa. Un sito web di questo genere viene realizzato unicamente tramite HTML, in questo modo ogni richiesta fatta avrà come risultato sempre la stessa pagina, indipendentemente dallo stato dell'utente e dalle sue azioni. Per questo motivo ben presto sono state definite moltissime tecnologie al servizio della creazione di pagine web dinamiche, dando vita al concetto di Web 2.0[1].
Una pagina web Dinamica non viene costruita a priori, bensì viene plasmata al momento, in base alla scelta dell'utente, alle sue azioni oppure allo stato di un determinato database annesso.
Molte tecnologie sono state sviluppate per garantire questa dinamicità, sia lato client, sia lato server dando vita alla creazione di vere e proprie applicazioni Web.

## 2.1 Un'applicazione web

Una web Application è un vero e proprio software sviluppato per essere eseguito all'interno del browser stesso, più precisamente un'applicazione client-server che fornisce intrattenimenti e servizi all'utente sfruttando il web.
La struttura che meglio rispecchia un'applicazione web è quella composta da tre strati: presentation layer, application layer, storage layer.

### 2.1.1 Presentation technologies

Il Presentation Layer rappresenta l'insieme di tutte le tecnologie client-side per la realizzazione di applicazioni web interattive ed animate, in risposta a determinati input dell'utente, eventi temporali, cambi di stato del browser. Il contenuto dinamico viene direttamente generato sul computer locale in uso, e più precisamente dal browser. I linguaggi più utilizzati per questo fine sono i cosiddetti client-side scripting languages, come Javascript, Actionscript oppure Flash. Questi linguaggi, definiti linguaggi interpretati, non vengono compilati, piuttosto vengono inviati dal web Server al browser, il quale si occupa interamente della loro esecuzione grazie a degli interpreti implementati ad hoc. Nella realizzazione di What's Up Web è stato utilizzato JavaScript[4], un linguaggio di scripting orientato ad oggetti standardizzato tra il 1997 e 1999 dalla ECMA (European Computer Manufacturers Association ) supportato ormai da tutti i browser attualmente in uso.
Javascript da solo non basta però per generare pagine web dinamiche. DHTML ( dynamic HTML ) rappresenta l'insieme di tutte le tecnologie necessarie per realizzare applicazioni web interattive. Oltre ad un linguaggio client-side come Javascript, abbiamo bisogno di un linguaggio markup statico, l'HTML, per realizzare e visualizzare il risultato finale; inoltre è necessario un linguaggio per gestire e formattare l'aspetto di un documento scritto in HTML come il CSS (Cascading Style Sheets); infine un modello ad oggetti per la rappresentazione strutturale del documento, neutrale sia alla lingua in uso sia alla piattaforma utilizzata. In questo caso il modello di riferimento è il DOM ( Document Object Model )[5], standard ufficiale utilizzato da tutti i browser per la rappresentazione dell'interpretazione di una pagina HTML. Questo modello prevede di visualizzare un documento sotto forma di albero tramite cui si possono scorrere tutti gli elementi che lo formano, permettendo di aggiornarne struttura e contenuto in modo rapidissimo. DOM viene spesso contrapposto ad un altro set di API (Application

[4]Inizialmente sviluppato ed implementato nel browser Netscape Navigator 2.0, Sun Microsystems e Netscape diedero il nome Javascript al primo linguaggio di scripting cliend-side, sotto licenza Mozilla Foundation. Microsoft allo stesso modo implementò il proprio linguaggio, Jscript. Il linguaggio venne successivamente standardizzato con il nome di ECMAScript dall'ECMA

[5]Document Object Model Technical Report

Programming Interface ), nel caso in cui l'applicazione si basi su XML; in questo caso per permettere la lettura e l'elaborazione di file XML viene utilizzato SAX[6] ( Simple Api for XML ) che a differenza del modello visto in precedenza analizza i documenti linea per linea, senza seguire una struttura ad albero; il flusso dell'analisi è unidirezionale, quindi una volta analizzato un determinato elemento questo non può essere più rielaborato a meno che non venga riprocessato l'intero documento.
Attraverso queste tecnologie è molto semplice ottenere effetti grafici nonché cambiamenti nell'aspetto e nel contenuto sia dopo un interazione con l'utente sia dopo un istante preciso, difficilmente ottenibili altrimenti. Con il DHTML non è necessario ricaricare nessuna pagina, nessun documento, ne richiedere al web Server di rigenerare nuovo contenuto. Invece, viene soltanto modificato il documento correntemente in memoria, manipolando in locale gli elementi HTML, gli attributi o gli stili, garantendo una quasi istantaneità nei cambiamenti grafici della propria applicazione.
Spesso per velocizzare il caricamento di una pagina vengono utilizzati alcuni strumenti che permettono di separare il contenuto dinamico di una pagina HTML dalla sua presentazione grafica. Per esempio si può creare a priori la struttura ( template ) di una pagina Web che solo su richiesta verrà riempita e compilata con il contenuto dinamico. Per ottenere questo risultato si può fare uso del Web Templating, una semplice tecnica che permette di costruire pagine HTML statiche a mano, contenenti codice Javascript; alla prima necessità, il browser richiederà al web Server il template, caricandolo in memoria ed eventualmente salvandolo nella propria cache per utilizzi successivi; il codice javascript presente provvederà poi, lato client, a popolare la pagina HTML con i valori opportuni. Per sviluppare questa procedura mi sono servito di un linguaggio per template, EJS[7] ( Embedded javascript ). EJS combina dati e template per creare pagine HTML client-side, mantenendo sempre un'ottima pulizia ed organizzazione nel codice. Questo strumento permette di caricare template da più file separati e di effettuare caching. Un'ulteriore funzionalità è quella di permettere di caricare template parziali (EJS's partials): esiste la possibilità all'interno di un template di richiamarne altri, aumentando la libertà e l'organizzazione del programmatore. Infine, caratteristica che ho trovato molto utile, è presente un'ottima gestione degli errori segnalando sempre la riga in cui è occorso.

### 2.1.2 Librerie di supporto

**Jquery** Nel progetto sono state utilizzate alcune librerie e framework adatti allo scopo. Innanzitutto è stata inclusa la libreria JQuery 1.7[8]. Questa libreria Javascript offre innumerevoli funzionalità che semplificano i vari aspetti dello strato di presentazione:

[6]The SAX project official Website
[7]http://embeddedjs.com/
[8]Jquery official website

- Potente e veloce set di strumenti per il matching di elementi in un documento HTML.
- Get & Set degli attributi DOM di un elemento.
- Notevoli funzioni per l'attraversamento dei nodi DOM e la loro manipolazione.
- Get & Set delle proprietà di stile ( attributi CSS ) di ogni elemento.
- Metodi per la registrazione dei comportamenti che l'utente effettua quando interagisce con il browser, manipolando ulteriormente questi eventi registrati.
- Tecniche per l'aggiunta di animazioni standard o sofisticate all'interno di pagine web.
- Un insieme di utility per manipolare tipi ed oggetti Javascript ( iteratori, parser, controlli ).
- Un insieme completo di metodi per il supporto di richieste asincrone i.e. Ajax (asynchronous Javascript ). La tecnologia verrà presentata più avanti.

**Backbone** Per quanto riguarda la gestione della struttura dell'applicazione What's Up Web è stato utilizzato il framework Backbone[9]. Lavorando in applicazioni che comportano un grande utilizzo di Javascript, spesso si tende ad affidarsi troppo al DOM, legandovi e manipolandovi ogni dato. Il risultato non è altro che un applicazione composta da pile infinite di callback e selettori JQuery all'interno del DOM, cercando in tutti i modi di mantenere una sincronizzazione tra i dati sensibili dell'applicazione e l'interfaccia utente HTML.
Di conseguenza è preferibile scegliere delle soluzioni più strutturate in cui i dati vengano completamente slegati dal DOM, cosi da poter essere gestiti separatamente ed in modo più pratico e veloce.
In Backbone i dati sono rappresentati come modelli che possono essere creati, validati, salvati e distrutti in ogni momento; i Models si basano su vincoli di tipo chiave-valore, dando grande spazio alla gestione degli eventi. Esiste la possibilità di creare insieme ordinati di modelli, chiamati Collections e quella di gestire l'intera interfaccia grafica in View logiche, che possono essere aggiornate indipendentemente quando un Modello cambia, senza dover ricaricare la pagina. Le View quindi sono un'ottima soluzione per implementare il web templating nella propria applicazione web.
Oltre a queste caratteristiche Backbone fornisce infine metodi per l'instradamento delle pagine web attraverso gli oggetti Router.

[9] http://backbonejs.org

**Bootstrap from Twitter** Le tecnologie presentate fino ad ora consentono di creare e gestire ottime applicazioni web interattive. Per rendere l'esperienza il più coinvolgente e piacevole possibile, tutto ciò non basta. Avere una grafica ben curata, gradevole alla vista, è sempre consigliato ai fini di un buon risultato. Per questo motivo è stato scelto di usufruire di Bootstrap[10], un set di HTML, CSS, Javascript per la costruzione di componenti dell'interfaccia utente e per le interazioni. Bootstrap, progettato e sviluppato dagli stessi designers di Twitter, è costruito per supportare a pieno i nuovi elementi HTML5 e la sua sintassi, accrescere le varie componenti con l'utilizzo delle nuove funzionalità CSS3, fornire supporto per ogni browser nonché tablet e smartphone, design super reattivo per garantire un'esperienza consistente. Questo front-end toolkit fornisce inoltre vari plugin JQuery (caroselli, popup window, tooltips, scroll spy, ecc) e la possibilità di poterli personalizzare a piacere.

### 2.1.3 Application layer

Oltre all'insieme di tecnologie lato client per fornire DHTML, la creazione di pagine web dinamiche richiede un ulteriore strato logico: l'application layer. Con questa nomenclatura si vuole indicare l'insieme di tecnologie lato server che permettono di costruire contenuto dinamico a priori secondo un database sottostante, e l'eventuale sincronizzazione dei dati con il DBMS ( Database Management System ). Mostrare i post in un forum, effettuare una registrazione su un sito web, visualizzare i prodotti in un sito di e-commerce, sono tutti esempi di pagine web dinamiche manipolate dalla business logic risiedente sul web server.
Per adempiere a queste funzionalità esistono molti linguaggi come PHP, Perl, Asp, Jsp; nello sviluppo di What's Up Web non si è fatto utilizzo diretto di queste tecnologie bensì sono state sfruttate delle Api digitali già presenti all'interno della piattaforma What's Up. Le Api ( Application Programming Interface ) rappresentano un insieme di metodi e procedure messe a disposizione del programmatore per svolgere determinati compiti all'interno del proprio software. Attraverso quest'astrazione, il programmatore ha la possibilità di utilizzare un insieme di strumenti già fatti, senza dover preoccuparsi di doverli scrivere da se e senza doversi interessare del loro funzionamento. In questo senso, le web API rappresentano un insieme di richieste HTTP con le relative direttive per la struttura dei messaggi di risposta, molto vicine al concetto di Web Service.

**Mashup** La possibilità di creare applicazioni web utilizzando servizi e dati offerti da più fonti, diverse dal web server in uso, ha dato vita al concetto di Mashup. Grazie a questa tecnica, i web masters possono tramite le Api usufruire di servizi altrimenti

[10] http://twitter.github.com/bootstrap/

inaccessibili, cosi da creare applicazioni sempre più complesse e sofisticate.
Esistono vari tipi di Mashup[11]:

- *Business Mashup* che prevedono la combinazione di risorse proprie con altre fornite da web service esterni;
- *Consumer Mashup* che accostano risorse provenienti da differenti sorgenti organizzandole in una semplice interfaccia grafica all'interno del browser;
- *Data Mashup*, riunisce risorse simili tra loro per contenuto in un'unica rappresentazione, creando un nuovo servizio non presente precedentemente.

What's up WEB appartiene alla prima categoria, Business Mashup. Infatti, durante lo sviluppo sono state utilizzate due Api, una proprietaria, per la gestione dell'intera piattaforma What'Up, l'altra di Facebook, per realizzare l'applicazione SHARE. Facebook mette a disposizione le Graph Api, un set di strumenti per ottenere una semplice e consistente visione del grafo sociale alla base di Facebook. Attraverso le Api è possibile ottenere informazioni di base dell'utente, la lista dei propri amici, le proprie foto e/o albums, manipolare gli stream del proprio profilo e moltissime altre funzionalità che hanno reso il social network Facebook uno dei servizi maggiormente esportato ed integrato sul web.

### 2.1.4 Storage layer

L'ultimo strato che compone la struttura di un'applicazione web è quello relativo allo storage, la gestione consistente dei dati sensibili. Per questo fine si fa uso di sistemi di gestione dei database, Database Management System[12], che forniscono un insieme di tecnologie e software per la manipolazione dei dati ( salvataggio, aggiornamento, rimozione, backup, sincronizzazione). Nella realizzazione di What's Up WEB non vi è stato alcun utilizzo diretto del DBMS, in quanto la manipolazione dei dati attraverso l'uso effettivo di linguaggi sql o tecnologie atte allo scopo viene fatto ad un livello più basso della piattaforma What's Up. L'utilizzo delle Api ha garantito quindi un più alto livello di programmazione, evitando di riscrivere codice inutile.

## 2.2 Technologies for requests & responses

Uno degli svantaggi nell'utilizzare tecnologie basate su Javascript è quello riguardante l'accesso ai dati sensibili di un'applicazione, memorizzati su database remoti. Come

[11]From wikipedia: http://en.wikipedia.org/wiki/Mashup

[12]Un Database Management System composto da software che permettono di organizzare i dati, effettuare manipolazioni (software SQL, Structured Query Language), gestire la sicurezza e l'integrit del sistema.

è stato spiegato in precedenza, gli script vengono scaricati dal web Server ed eseguiti lato client, di conseguenza l'accesso al database non può essere fatto tramite Javascript stesso ma attraverso un linguaggio a parte che esegua la transazione e restituisca i dati in una variabile dello script. In una operazione di questo genere è necessario l'intero caricamento della pagina stessa, ogni volta che viene eseguita una transazione con il database.
Per superare questi limiti imposti nasce Ajax ( Asynchronous Javascript ), una delle tecnologie di sviluppo web alla base del concetto di web 2.0.

**Ajax** La programmazione Ajax permette di inviare e ricevere dati da un server in modo asincrono, senza dover ricaricare l'intera pagina. Un evento asincrono rappresenta un'azione eseguita al di fuori del principale flusso dell'applicazione, in modo da non interromperne la corrente esecuzione. In questo modo, un applicazione web può eseguire delle transazioni con database remoti senza minimamente interferire con la visualizzazione e il comportamento della pagina in uso all'utente. Questo permette di incrementare notevolmente le prestazioni dell'applicazione, nonché di separare le richieste fatte al server dai dati ricevuti in modo asincrono. L'utente non ha più l'obbligo di aspettare che le risposte del server siano completate prima di poter eseguire qualsiasi altra operazione all'interno della pagina.
Per recuperare i dati, le tecnologie Ajax prevedono l'utilizzo delle XMLHTTPRequests, un Api disponibile nei linguaggi web di scripting come Javascript per inviare richieste HTTP oppure HTTPS direttamente ad un web server o Api server. I metodi[13] di richiesta utilizzati in What's Up Web forniti dal protocollo HTTP sono[2]:

- GET: utilizzato per richiedere la rappresentazione di una risorsa specifica. Questa verrà restituita nel body.
- POST: utilizzato per inoltrare una determina risorsa al web server; la risorsa verrà specificata all'interno del body della richiesta.
- DELETE : viene usato per eliminare una risorsa.
- Altri metodi non utilizzati: HEAD, simile alla GET, ma senza body di risposta, viene utilizzato per ottenere meta data contenuti nell'header del pacchetto; OPTIONS, ritorna i metodi HTTP supportati dalla URL specificata; CONNECT, converte la richiesta corrente in un tunnel TCP/IP, utilizzato per comunicazioni SSL attraverso proxy HTTP; PATCH, utilizzato per effettuare modifiche ad una risorsa.

Il supporto alla tecnologia Ajax viene fornito dalla libreria JQuery che mette a disposizione specifici metodi per l'utilizzo di richieste asincrone.

[13]RFC 2616 for HTTP methods

La risposta di una richiesta Ajax è tipicamente formattata in XML ( Extensible Markup Language )[14] un linguaggio markup sempre più utilizzato nella trasmissione di dati e documenti sul Web, soggetto agli standard definiti dal W3C. L'XML è il formato usato per la maggiore nei Web Service e Web Api per la trasmissione dei messaggi tramite il protocollo HTTP.

**Json** Un'alternativa all'XML è fornita dal JSON ( Javascript Notation Object )[15], un altro formato utilizzato per lo scambio di dati in applicazioni web soprattutto insieme ad AJAX. A differenza dell'XML, JSON non prevede l'utilizzo di tag.
Nel progetto What's Up Web è stato preferito utilizzare la notazione JSON anziché XML, visto il maggior supporto di Javascript nel parsing ma soprattutto perché sia le Api della piattaforma What's Up, sia le Graph Api di Facebook, prevedono l'utilizzo di oggetti JSON come risposta alle richieste effettuate.

```
/*
 * example of JSON object
 */
{
    "id": 123,
    "type": "SMS",
    "message": "Ciao a tutti",
    "Author": {
        "name": "Massimiliano",
        "surname": "Battan"
    }
}

```

Figura 2.2: Esempio di un documento in formato JSON

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Examble of XML format
-->
<root>
    <resource>
        <id>123</id>
        <type>SMS</type>
        <message>Ciao a tutti</message>
        <Author>
            <name>Massimiliano</name>
            <surname>Battan</surname>
        </Author>
    </resource>
</root>

```

Figura 2.3: Esempio di un documento in formato XML

[14] http://www.w3.org/XML/
[15] http://www.json.org/

L'immagine seguente rappresenta un riassunto delle tecnologie utilizzate nello sviluppo di What's Up WEB, cercando di mostrare la loro interazione.

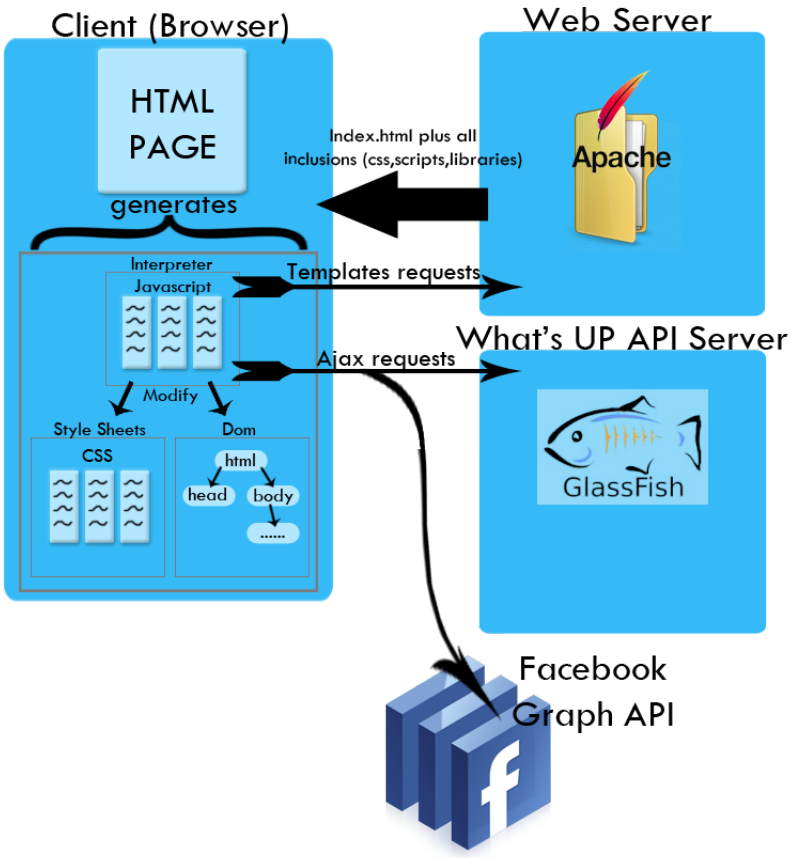


Figura 2.4: Schema delle tecnologie utilizzate in What's Up WEB

# Capitolo 3

# Requisiti

Lo sviluppo dell'applicazione What's Up Web ha richiesto un'attenta stesura dei requisiti per entrambe le funzionalità di Share e Display.

## 3.1 Share

Vista la vastissima diffusione dei social network tra i giovani, abbiamo deciso di discostarci dalla normale applicazione di share già realizzata per smartphones. In quel caso infatti, le immagini che potenzialmente potevano essere inviate erano quelle presenti sul dispositivo utilizzato oppure quelle scattate per lo scopo. L'applicazione web realizzata invece avrebbe dovuto permettere all'utente di condividere qualsiasi propria immagine presente su uno dei social network attualmente più in uso nella fascia di età giovanile, Facebook. In questo modo dopo aver caricato una foto su Facebook il giovane può decidere in qualsiasi momento di condividerla anche con la propria nonna/o senza doverla salvare per poi inviarla in un secondo momento.

### 3.1.1 Requisiti Funzionali

Per ottenere questo risultato è stato necessario realizzare un'applicazione Facebook embedded, utilizzabile quindi sia all'interno di una pagina Canvas fornita ad hoc da Facebook, sia all'esterno del social network utilizzando un URL precisa.
What's up Share deve permettere all'utente in modo semplice e soprattutto veloce di poter condividere, con le persone abilitate alla ricezione delle risorse, ogni immagine presente nel proprio account Facebook. Di conseguenza l'utente deve avere la possibilità di sfogliare le proprie foto, organizzate in album, mantenendo in questo modo l'ordine a cui egli stesso è abituato e che si aspetta di trovare.
Inoltre l'applicazione Share prevede l'implementazione della gestione degli inviti. Ogni utente che utilizza una delle applicazioni What's Up ha infatti l'opportunità di

invitare tramite mail un qualsiasi parente o amico in modo tale da permettere all'invitato , dopo aver accettato, di diventare un contribuente del proprio gruppo personale. Da quel momento in poi ogni contribuente potrà in qualsiasi momento inviare foto e messaggi all'utente (nel nostro caso l'anziano). Visto lo scenario, Web Share fornisce la possibilità all'utente di ricevere le notifiche degli inviti che lui stesso ha ricevuto, permettendo di accettare oppure rifiutare singolarmente ogni richiesta.

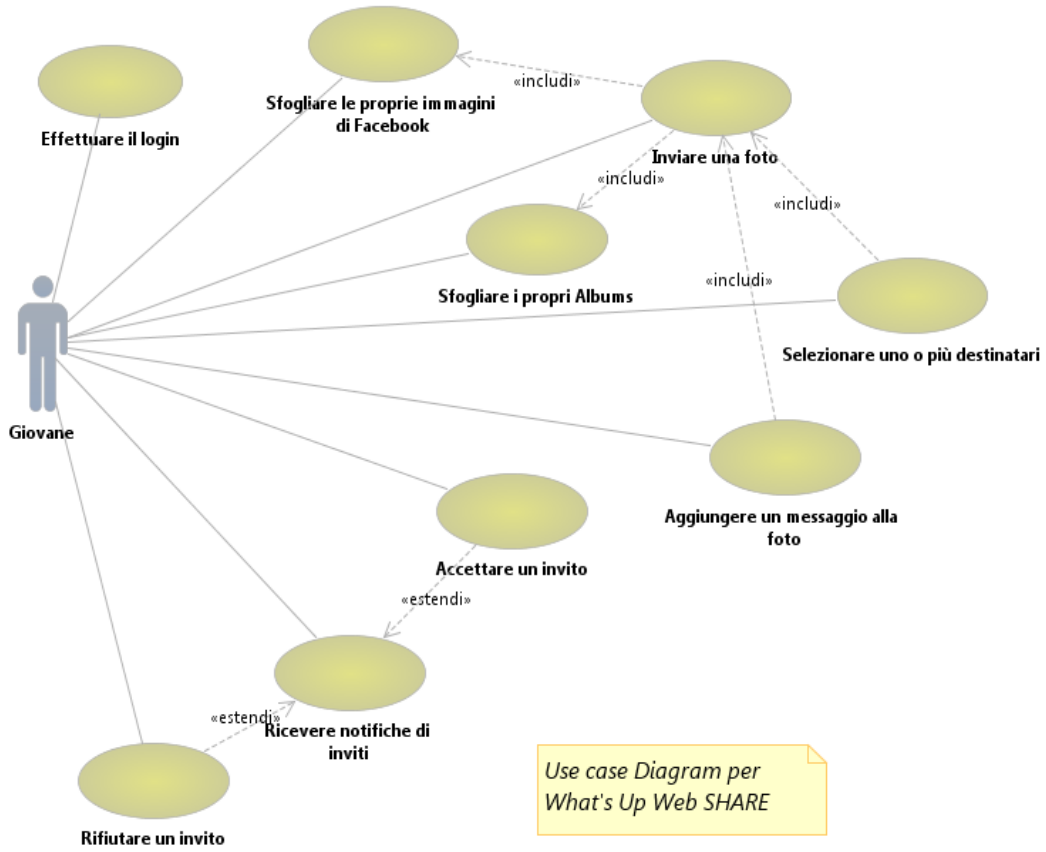


Figura 3.1

Come si può osservare dallo use case Diagram in figura 3.1, l'applicazione deve fornire all'utente le seguenti funzionalità:

- Effettuare il login utilizzando un account Facebook preesistente
- Sfogliare le proprie immagini. L'applicazione dovrebbe permettere subito dopo l'accesso di avere disponibili in prima pagina alcune foto (anche tutte) cosi da permettere in modo rapido l'inoltro di una di queste.

- Sfogliare i propri album. E' richiesta un'organizzazione ulteriore delle foto in Album, seguendo la struttura di Facebook cosi da non disorientare l'utente.

- Visualizzare eventuali inviti ricevuti.

- Accettare o rifiutare un invito.

- Dopo aver scelto un'immagine, l'utente può decidere di inviarla. A questo punto avrà la possibilità di selezionare, se presenti, gli eventuali destinatari, nonché di aggiungere un messaggio facoltativo; al termine della procedura l'utente dovrà essere informato dell'eventuale successo o insuccesso.

- Non è prevista alcuna funzione di logout poiché, nonostante possa essere utilizzata anche esternamente, What's Up SHARE nasce come applicazione Facebook embedded.

### 3.1.2 Requisiti non funzionali

Oltre ai requisiti sopracitati ve ne sono altri che devono rispettare la politica di What's Up e di Facebook.

- L'account utilizzato per il login nell'applicazione deve essere un utente già registrato su Facebook.

- L'utente deve permettere all'applicazione di ottenere le proprie informazioni di base ed il permesso di accedere alle proprie immagini.

- L'applicazione deve essere accessibile utilizzando entrambi i protocolli HTTP e HTTPS come richiesto dalla politica di Facebook.

- La graphic user interface (GUI) deve essere ben curata, utilizzando il framework Bootstrap (Twitter).

- L'applicazione web deve lavorare su un server apache 2.2.

- L'applicazione dovrà essere conforme ai principi dell'architettura REST.

L'ultimo punto verrà chiarito meglio nel capitolo 4.

## 3.2 Display

What's Up Web Display nasce dalla volontà di voler esportare le funzionalità già espresse dall'applicazione sviluppata per IPad, su web, garantendo un'altissima portabilità all'intera piattaforma.

Display deve garantire all'anziano di visualizzare ogni messaggio o immagine che egli ha ricevuto, dando la possibilità di navigare tra le risorse, in senso temporale. Inoltre deve essere integrata la possibilità di invitare un parente o amico a far parte del proprio gruppo, rendendolo un contribuente, nonché di visualizzare gli attuali contribuenti presenti. I requisiti funzionali dell'applicazione sono:

- Effettuare il login.
- Effettuare il logout. A differenza di Share, Display non è un'applicazione embedded di Facebook, per questo motivo l'utente deve avere la possibilità di effettuare la disconnessione.
- Visualizzare tutte le risorse ricevute, una alla volta. I tipi di risorsa che dovranno essere visualizzati sono: Immagine (con titolo opzionale), Messaggio, Immagine+Testo (con titolo opzionale).
- Visualizzare la persona che ha inviato la risorsa.
- Visualizzare le risorse più vecchie con un semplice click.
- Visualizzare le risorse più recenti con un semplice click.
- Visualizzare le persone abilitate all'invio di risorse.
- Invitare un nuovo utente a contribuire. In quel caso, utilizzando la piattaforma What's Up, l'utente in questione riceverà una mail in cui verrà invitato a scaricare od utilizzare una delle applicazioni di What's Up per la condivisione e di conseguenza avrà l'opportunità di accettare l'invito a diventare un contribuente del gruppo. Nel caso in cui l'utente invitato sia già presente nel sistema come fruitore dell'applicazione, non riceverà nessuna mail bensì un invito diretto all'interno dell'applicazione che egli utilizza.

### 3.2.1 Requisiti non funzionali

Oltre ai requisiti descritti, sono stati discussi alcuni requisiti non funzionali necessari a rispettare il target dell'applicazione, l'anziano. Grande attenzione viene data al layout, mantenendo una grafica gradevole ma il più possibile semplice ed immediata. Largo

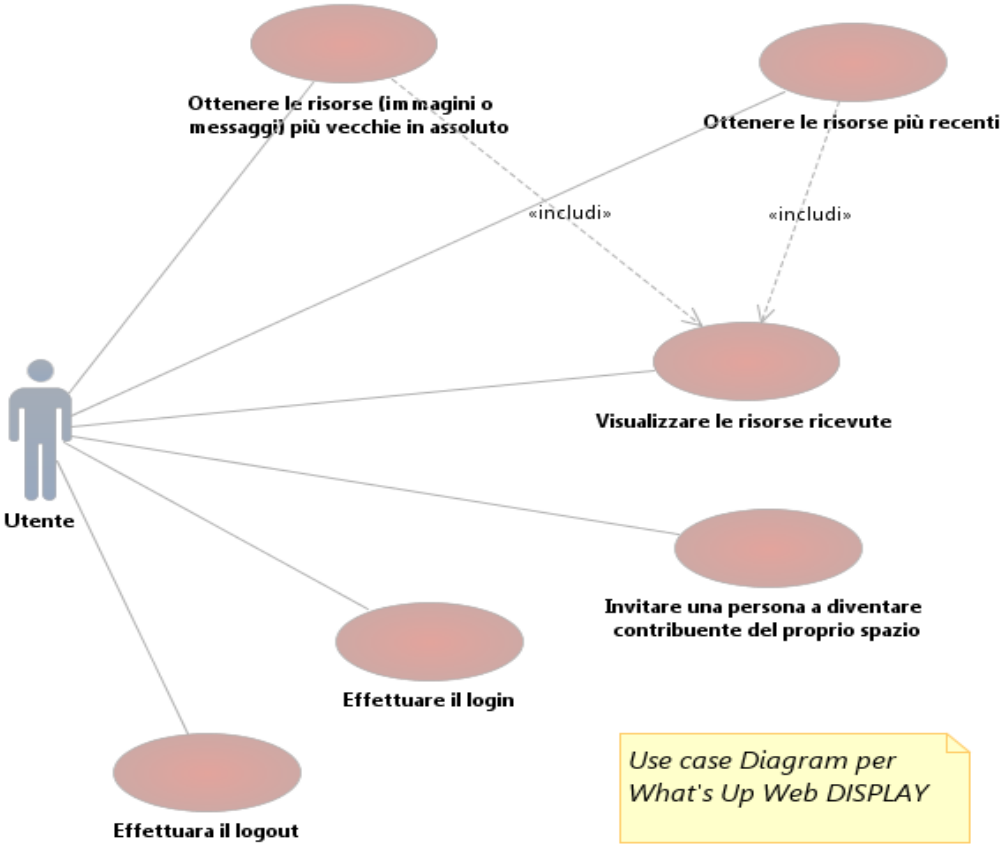


Figura 3.2

spazio va dato alla visualizzazione della risorsa, riservando solo una piccola parte dell'interfaccia alle funzionalità.
Visto il dato trattato, immagini, possiamo trovarci di fronte a notevoli ritardi nel caricamento delle risorse dovuti alla loro dimensione, che molto spesso può anche superare il Megabyte per ognuna. A questo si possono aggiungere i ritardi o quantomeno delay del server di What's Up nel caso in cui le risorse richieste siano innumerevoli. In ultimo, ma non per importanza, si aggiunge la sgradevolezza alla vista di un'immagine non subito disponibile, che si carica mentre la stiamo visualizzando ( aspetto per altro sicuramente poco chiaro ad un anziano ). Per questi svariati motivi uno dei requisiti su cui si è lavorato maggiormente è stato la velocità. L'applicazione infatti deve essere in grado di evitare al massimo qualsiasi tipo di caricamento visibile all'utente e di richiedere al server le risorse volta per volta se necessario, senza richiederne troppe per volta. Riassumiamo di seguito le caratteristiche:

- Creazione invisibile al primo utilizzo di uno spazio per l'utente, in cui successivamente verrà richiesto di invitare dei contribuenti.
- Grafica curata, semplice ed intuitiva, garantendo largo spazio alla visualizzazione delle immagini. E' richiesto l'utilizzo del framework Bootstrap (Twitter).
- Notevole velocità.
- Polling continuo per controllare l'eventuale presenza di nuovi messaggi. In quel caso l'utente dovrà essere avvertito della loro presenza e messo in condizioni di poter visualizzare i nuovi messaggi attraverso una semplice azione.
- Il login deve essere effettuato con un account Facebook già registrato.
- L'applicazione deve funzionare su un server Apache 2.2.

### 3.2.2 Un'applicazione per il Chrome Web Store

Una delle future modalità di fruizione dell'applicazione DISPLAY sarà tramite il Chrome Web Store[1], uno spazio dedicato agli sviluppatori per pubblicare le proprie applicazione Web o giochi, in cui gli utenti di Chrome possono facilmente scaricarle all'interno del proprio browser. I canoni che dovrebbe rispettare un'applicazione Web, definiti dallo stesso Chrome Store sono i seguenti[2]:

- **Focus preciso**: *A web app should do one thing, and one thing well*. Il contenuto dell'applicazione deve essere il più possibile mirato, in modo che l'utente, avendo l'obbiettivo ben chiaro in mente, lo può conseguire immediatamente evitando di includere funzionalità esterne e non precedentemente pubblicizzate.
- **Big Screen**: l'interfaccia utente deve prediligere l'utilizzo di grandi schermate in quanto viene tipicamente eseguita su un notebook o pc desktop. Sono da evitare l'utilizzo di menu, banner, link in quanto non stiamo sviluppando un sito web, bensì un' applicazione scelta da un utente per un preciso motivo: soddisfare l'esigenza del caso.
- **Ricca esperienza**: la web app deve avvicinarsi il più possibile ad una comune applicazione desktop in modo da esprimere al massimo le proprie funzionalità ed incontrare le aspettative dell'utente.
- **Bellezza**: Le innumerevoli tecnologie per lo sviluppo dell'interfaccia grafica supportate dai moderni browser permettono di costruire applicazioni sempre più coinvolgenti e piene di effetti grafici.

[1] https://chrome.google.com/webstore

[2] Thinking in web app, from Google Developer

- **Velocità**: la risposta ad azioni dell'utente deve essere immediata e la reattività sia reale che percepita deve essere pari ad un'applicazione desktop.

# Capitolo 4

# Architettura dell'applicazione

What's Up Web, come descritto nei capitoli precedenti, rappresenta un espansione delle funzionalità della piattaforma What's Up. Per questo motivo è bene chiarire l'architettura alla base dell'intero progetto, per poi entrare nello specifico riguardo la parte Web.

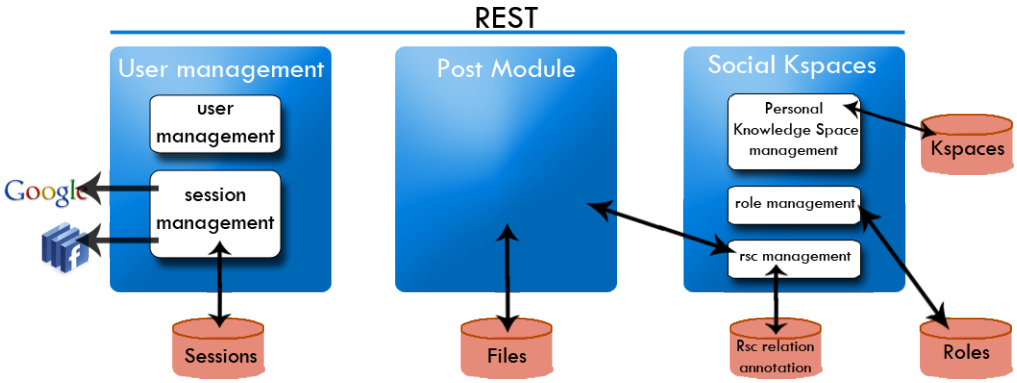


Figura 4.1: Architettura What's Up Platform

Come si può vedere dall'immagine 4.1, l'architettura di What's Up è formata da tre componenti principali:

- Lo **User Management** comprende la gestione degli utenti e le loro relative sessioni. Per connettersi ad uno dei servizi What's Up è richiesto tassativamente che l'utente sia in possesso di un account Google oppure Facebook: attraverso i servizi di Google Friend Connect e di Facebook è possibile creare una sessione

all'interno della piattaforma, necessaria per poter utilizzare qualsiasi funzionalità. Lo User Management è connesso con il relativo database per la gestione delle sessioni.

- **Modulo POST**: la componente incaricata di gestire le risorse inviate tramite POST al sistema, annessa al database per la gestione dei files.

- **Social Kspaces**: il cuore della piattaforma, descritto nel prossimo paragrafo.

## 4.1 Kspace Management

Alla base dell'architettura vi è il concetto di KS ( Knowledge Space ), un deposito virtuale per differenti tipi di risorse. Ad ogni utente che accede per la prima volta all'applicazione viene associato un Kspace di cui diventa il proprietario. Successivamente può provvedere ad invitare altri utenti a far parte del proprio Kspace (come Contributors). Un Kspace viene utilizzato per condividere risorse che attualmente possono essere di tipo SMS, IMAGE oppure IMAGE_SMS ( immagine con sms ), attraverso il modulo POST.
Il gestore, oltre ad amministrare i Kspace, si occupa di gestire i ruoli all'interno di ogni spazio nonché di mantenere sincronizzate le relazioni tra Kspace e risorse del modulo POST.
All'interno del progetto Web viene fatto uso di tutti e tre i moduli: gestore degli utenti per la connessione con Facebook e relativa gestione della sessione, modulo Post per inviare le risorse nell'applicazione SHARE, Kspace Management per tutte le altre operazioni presentate nei requisiti del capitolo 3.

## 4.2 Il design REST

Il concetto di REST ( Representational State Transfer ) esprime un insieme di criteri per il design di servizi e applicazioni Web. Il termine RESTful è molto simile a quello di object-oriented per un linguaggio. Un'applicazione può essere progettata in modo object-oriented ma ciò non fa si che la sua architettura sia object-oriented. In questo caso, l'architettura che meglio esprime il concetto di RESTful è la ROA ( Resource oriented Architecture ). Se da una parte, il design REST prevede l'utilizzo del metodo HTTP per incorporare le informazioni relative al metodo della richiesta, l'architettura ROA prevede l'incorporatura delle informazioni di scoping all'interno dell'URI ( Uniform Resource Identifier ). L'unione di questi due strumenti permette di avere a disposizione un potente strumento. Attraverso una semplice riga come questa infatti *(GET /reports/open-bugs HTTP/1.1)* è subito intuibile quali siano le intenzioni del

client.
Il precedente esempio mi può aiutare meglio a presentare l'architettura in esame. Alla base di tutto si trova il concetto di risorsa:

> *A resource is anything that's important enough to be referenced as a thing in itself.*[3]

Una risorsa rappresenta qualsiasi elemento coerente del sistema che può essere rappresentato tramite un indirizzo. Da qui il primo principio chiave, **addressability**, cioè l'esigenza che ogni risorsa possa essere indirizzabile tramite un URI, il più possibile auto descrittiva. Un client può in qualsiasi momento richiedere al server una risorsa tramite il protocollo HTTP ed i relativi metodi supportati ( GET, POST, DELETE, ), senza la necessità di incorporare all'interno del pacchetto HTTP ulteriori messaggi, quanto avviene invece in architetture RPC ( Remote Procedure Call )[1] utilizzando SOAP (simple object access protocol)[2]. Questo conferisce al sistema un'interfaccia uniforme.
Un altro principio alla base dell'architettura ROA è **statelessness**: ogni richiesta HTTP effettuata dal client verso il server deve accadere in completo isolamento, includendo tutte le informazioni affinché il server possa rispondere in modo completo. Il server è privo di qualsiasi stato e non tiene traccia di nessun evoluzione del client, cosicché ogni richiesta può essere soddisfatta in qualsiasi momento.
Quando un client richiede una risorsa tramite il metodo sopracitato, la risposta che egli otterrà non sarà l'effettiva risorsa locata sul server, bensì un oggetto specificatamente formattato in un determinato linguaggio che rappresenta lo stato attuale della risorsa richiesta; da qui il concetto alla base dei sistemi RESTful, **the representation**:

> *A representation is any useful information about the state of a resource.*[3]

E' evidente ora il significato di REST, trasferimento dello stato che rappresenta ( la risorsa ). Riassumendo i concetti alla base di un'applicazione RestFul sono:

- La risorsa.
- Il suo nome (URIs).
- La sua rappresentazione.
- I collegamenti tra le risorse.

[1] Con Remote Procedure Call si intende la modalità in cui una determinata procedura viene eseguita su un computer diverso da quello che ha richiesto la sua esecuzione.
[2] La documentazione SOAP è disponibile all'indirizzo http://www.w3.org/TR/soap/

Le proprietà riassunte dell'architettura ROA, che meglio esprimono il design REST, sono:

- Client-server approach
- Addressability
- Statelessness
- Uniform Interface

What's Up Web è a tutti gli effetti un' applicazione RESTful in quanto si avvale dell'utilizzo di Api web interamente basate sull'architettura ROA. Più nello specifico, quello che rende What's Up Web il client ideale per Web services RESTful è l'utilizzo dell'architettura Ajax. Il sistema prodotto infatti segue sempre questa procedura:

1. L'utente richiede, attraverso il browser, l'URI principale dell'applicazione.
2. Il server serve la richiesta fornendo una pagina web con degli script embedded.
3. Il browser inizialmente renderizza la pagina html ed esegue tramite il suo interprete gli script presenti.
4. Gli script eseguono delle richieste HTTP asincrone ai server Api di What's Up sia in modo autonomo, sia in risposta ad alcune azioni dell'utente. L'utente è quasi sempre all'oscuro di queste richieste.
5. Gli script si occupano di manipolare la risposta del server, modificando l'interfaccia utente tramite l'alterazione del DOM e il web Templating ( tecnologie descritte nel Capitolo 2).

Come si può vedere un'applicazione Ajax segue bene i canoni principali di un'architettura orientata alle risorse. Inoltre il client viene sempre eseguito all'interno del web browser, il cui ambiente rafforza e stimola l'utilizzo del design REST, piuttosto di SOAP ed altri protocolli che appesantiscono il browser.

## 4.3 Architettura del Presentation Layer

L'interfaccia utente dell'applicazione sfrutta il modello MVC ( Model-View-Controller ) come pattern architetturale. Alla base del design MVC vi è la separazione tra i dati sensibili dell'applicazione e la loro visualizzazione. Il modello prevede la suddivisione dell'applicazione in tre componenti principali, definendone la loro interazione:

- *Model*: Rappresenta i dati dell'applicazione

- *Controller*: Gestisce l'interazione dell'utente con il sistema, manipolando se necessario il Model.
- *View*: Ciò che l'utente vede, la rappresentazione grafica del Model.

Attraverso l'utilizzo del framework Backbone, What's Up Web segue questa tipologia di design per gestire i client script. Le tipologie di script sono quattro, ognuna con una caratteristica specifica e i propri metodi caratteristici. La funzione di controller viene eseguita da due tipologie di script diversi; gli script *Router* si occupano di gestire la navigazione all'interno delle pagine, rimanendo in ascolto sull'URL del web Server. Ad ogni Router viene assegnata una determinata pagina ( o più ); ogni volta che l'URL cambia, il Router incaricato per la pagina selezionata provvede al caricamento del determinato script *View*. Gli script View rappresentano il cuore dell'applicazione. In essi si provvede sia al caricamento dei template per la relativa pagina ed al loro rendering, sia alla gestione degli eventi che vi possono occorrere. All'interno della View inoltre vengono eseguite le richieste Ajax per i servizi What's Up, inoltrando la rappresentazione delle risorse ai template, per la loro visualizzazione nell'interfaccia grafica. Per manipolare i dati Backbone mette a disposizione gli oggetti *Model* e *Collection* (un insieme di Models ) che possono essere creati, validati, distrutti oppure salvati sul server, sempre mantenendo sincronizzata la View associata.

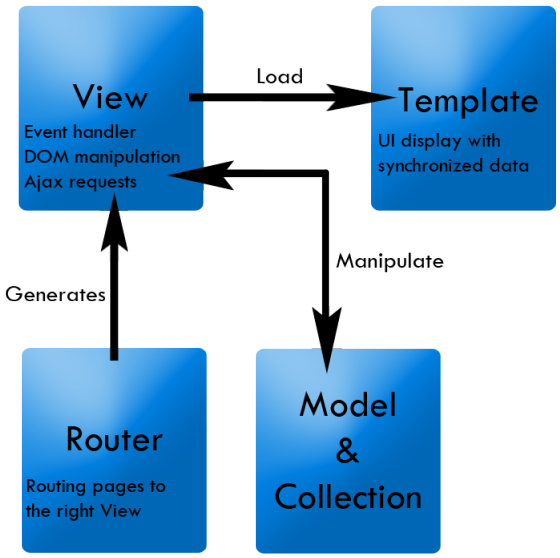


Figura 4.2: Design MVC di What's Up Web utilizzando Backbone

# Capitolo 5

# Sviluppo dell'applicazione ed esempi di utilizzo

In questo capitolo presenterò le soluzioni adottate ed i problemi riscontrati durante lo sviluppo dell'applicazione, per arrivare ad illustrare il risultato finale tramite degli esempi di utilizzo e frammenti di codice.
Come si può vedere dall'immagine sottostante, la home page di What's Up Web è formata da due semplici bottoni che rimandano alle due principali funzionalità della piattaforma : SHARE e DISPLAY.


Figura 5.1: Main page di What's Up Web

## 5.1 Connessione a Facebook

Come descritto nell'architettura del Capitolo 4 , per accedere ad entrambe le applicazioni l'utente deve essere in possesso di un account Facebook con il quale deve effettuare il login. Una volta loggati in Facebook, il sistema deve recuperare l'*access*

*token*[1], richiesto per creare una sessione all'interno del sistema What's Up. Per ottenere questo risultato è stato necessario innanzitutto creare due diverse applicazioni Facebook all'interno del programma Facebook Developershttps://developers.facebook.com/, a cui vengono associati i relativi Application ID. Ho dovuto utilizzare due Application ID invece di uno perché SHARE e DISPLAY appartengono a due differenti categorie di applicazioni Facebook; SHARE rappresenta una vera e propria applicazione Facebook embedded ( anche se può essere utilizzata all'esterno del Canvas[2] di Facebook ), mentre DISPLAY sfrutta solo le funzionalità del Social Network per effettuare l'accesso al sistema What's Up tramite Facebook. Una volta registrate le due applicazioni all'intero di Facebook per ottenere gli app ID, mi sono servito dell'SDK Javascript per effettuare le chiamate ai server Api di Facebook. Per caricare l'SDK è bastato includere all'interno della pagina HTML lo script fornito dallo stesso Facebook tramite:

```
<script type="text/javascript" src="https://connect.facebook.
    net/en_US/all.js"></script>
```

ed inserire all'interno del body HTML il seguente tag

```
<div id="fb-root"></div>
```

ai fini di caricare in modo corretto l'sdk, ed agganciare eventuali finestre popup al documento. Una volta configurato l'SDK, ho dovuto utilizzare le Api Facebook per inizializzare l'SDK con il relativo application ID per poi passare all'effettivo login dell'utente.
La gestione del login prevede tre casi:

1. L'utente è già loggato in Facebook e ha già autorizzato l'applicazione. In questo caso posso ottenere subito l'access token ed inizializzare la mia applicazione.

2. L'utente è già loggato in Facebook ma non ha autorizzato l'applicazione. In questo caso tramite un popup di Facebook il sistema chiede all'utente di accettare i permessi.

3. L'utente non è loggato in Facebook. Verrà generata una finestra popup per il login, al termine del quale si torna al punto 1 oppure 2.

Per quanto riguarda i permessi dell'applicazione, SHARE richiede che l'utente dia il consenso all'accesso delle proprie foto e album all'interno di Facebook mentre DISPLAY richiede solo l'accesso alle informazioni di base ( nome, email, ).

[1]Un access token di solito è una stringa di caratteri generati del servizio in questione, tramite cui viene identificata la sessione dell'utente.

[2]Facebook utilizza questa nomenclautura per definire lo spazio entro il quale viene inserita l'applicazione.

```
1
2 var fbLoginScopes = {
3 scope : 'email,publish_stream,user_photos'
4 };
```

Riassumendo, affinché l'utente possa effettivamente loggare all'intero dell'applicazione, Facebook deve prima autenticare l'utente, poi autenticare il sito web per assicurarsi che l'utente stia dando le proprie informazioni ad un sito effettivamente registrato su Facebook, in ultimo l'utente deve esplicitamente autorizzare il sito ad avere accesso alle sue informazioni.
E' bene sottolineare che tutte le Api di Facebook agiscono in modo asincrono, permettendo al flusso dell'applicazione di poter comunque scorrere durante l'utilizzo di quest'ultime. Di conseguenza va sempre fatta molta attenzione a quando vanno compiute determinate azioni. Chiamare il metodo per la creazione di una sessione What's Up all'esterno ( se pur successivamente ) della chiamata login di Facebook per esempio, non assicura certamente che questa andrà a buon fine. E' molto probabile che durante l'esecuzione di un tipico metodo *createSession* la procedura di login di Facebook non abbia ancora restituito l'access token necessario, generando l'impossibilità di creare una sessione.

### 5.1.1 Immagini e album di Facebook in SHARE

Entrambe le applicazioni utilizzano Facebook per il login, ma soltanto SHARE fa uso delle Graph Api[3] per accedere alle foto dell'utente.
Le Graph Api permettono di avere accesso al grafo sociale di Facebook, rappresentandone gli oggetti. Ad ogni oggetto, che sia un utente, un album, una foto, viene associato un ID unico con il quale vi si può accedere. Il pattern tipico di una Graph Api utilizzando Javascript é il seguente:

```
1 FB.api("url for the resource/s",{options},{function(response)
    });
```

Per accedere agli album di un utente per esempio, l'url utilizzata é *'me/albums'* dove a *me* é automaticamente associato dall'SDK l'ID dell'utente che sta utilizzando l'applicazione, mentre per accedere alle foto del medesimo si fa uso di *'ID_album/photos'*.
L'ultimo argomento della procedura é una funzione il cui parametro a sua volta contiene il risultato della richiesta. Essa viene eseguita solo al termine della chiamata, quindi, per quanto detto nel paragrafo precedente, la manipolazione dei dati in risposta deve essere effettuata all'interno di quella funzione, e non all'esterno . Una funzione di questo tipo é solitamente definita *callback*.

[3]References per le Graph Api

Durante l'utilizzo di queste Api ho riscontrato un paio di problemi nella manipolazione degli Album; durante l'analisi dei json di risposta infatti, mi sono accorto di alcune difformità e incongruenze. Spesso il json conteneva Album invisibili, cioé album che l'utente stesso non sa di possedere e che non vengono visualizzati all'interno della navigazione album di Facebook stesso. Alcune volte poi, il json non forniva nessun campo cover_img, utilizzato di consueto per ottenere l'immagine cover dell'album ( problema risolto assegnando alla prima immagine dell'album l'onere di essere anche l'immagine cover) . Inoltre, in alcuni casi di test, alcuni account presentavano delle tipologie di album ( questa volta effettivamente presenti in Facebook ) con caratteristiche inconsuete: nel json degli album in questione risultava un numero maggiore di 0 di foto, nonché un preciso ID per l'immagine utilizzata come cover; nonostante ciò, l'interrogazione ( tramite l'Api già menzionata 'ID_album/photos' ) restituiva un json vuoto. Dopo un'attenta analisi dei vari album, sono giunto alla conclusione che alcune tipologie di album contengono immagini non effettivamente presenti sui server Facebook, bensì situate su server esterni: questo é il caso di album generati da applicazioni esterne, come per esempio Instagram[4]. In questo caso ho fatto in modo che SHARE evitasse direttamente l'analisi di album creati da altre applicazioni.

## 5.2 Generazione del contenuto dinamico tramite Backbone

In questo paragrafo voglio mostrare il procedimento con cui viene costruita una pagina all'interno dell'applicazione web con l'utilizzo del framework Backbone; questa procedura sarà sempre la stessa per ogni contenuto sia di SHARE che di DISPLAY.

1. Nella pagina principale di riferimento, in questo caso index.html, viene inserito un elemento inizialmente vuoto, al cui interno verrà caricato il futuro template.

```
<div id="wu-page" ></div>
```

2. Viene costruito un script di routing come il seguente:

```
WU.Routers.Home = Backbone.Router.extend({
  routes : {
    "/home" : "displayHome"
  },
  displayHome : function() {
    var homeView = new WU.Views.Home({
```

[4]Instagram è un'software gratuito che permette di scattare foto dal proprio smartphone per poi poterle condividere all'interno dell'applicazione e opzionalmente anche su vari social network.http://instagram.com/

```
7       el : $('#wu-page')
8     });
9   }
10 });
```

In questo caso, ogni volta che l'utente accede all'url *'domainurl/home'* verrà richiamata la funzione *displayHome()*. Questa funzione a sua volta crea un oggetto View nel quale viene fatto un binding tra l'elemento *el*, caratteristico dell'oggetto View, e l'elemento del DOM ottenuto tramite Jquery. In questo modo il contenuto dinamico generato da View verrà appeso all'interno di *el*.

3. All'interno dello script View viene fatto un ovverride del metodo *render()*, specificando l'azione da compiere. Nell'esempio seguente, viene inizialmente caricato il template all'interno del browser. Successivamente viene chiamata la funzione render sul template. Questo metodo accetta una lista di elementi chiave-valore utilizzati dal template per costruire il contenuto dinamico. In questo caso viene legato all'oggetto *albums* un JSON di album precedentemente caricati da Facebook. All'interno del template EJS sarà possibile richiamare *albums* per accedere al JSON.

```
1 render : function(listaAlbum) {
2       var html = new EJS({
3          url : './scripts/ejs/homeEJS.ejs'
4       }).render({
5          albums : listaAlbum.toJSON(),
6       });
7       $(this.el).html(html);
8     }
```

4. Una volta caricato e popolato il template, viene utilizzato il metodo Jquery *.html()* per inserire all'interno dell'elemento *el* (nel caso specifico all'interno del *div wu-page* ) il frammento HTML generato dal template.

## 5.3 Invio di una foto

Seguendo l'activity Diagram della figura 5.2, descrivo ora i principali passaggi nell'invio di una risorsa tramite l'applicazione SHARE.

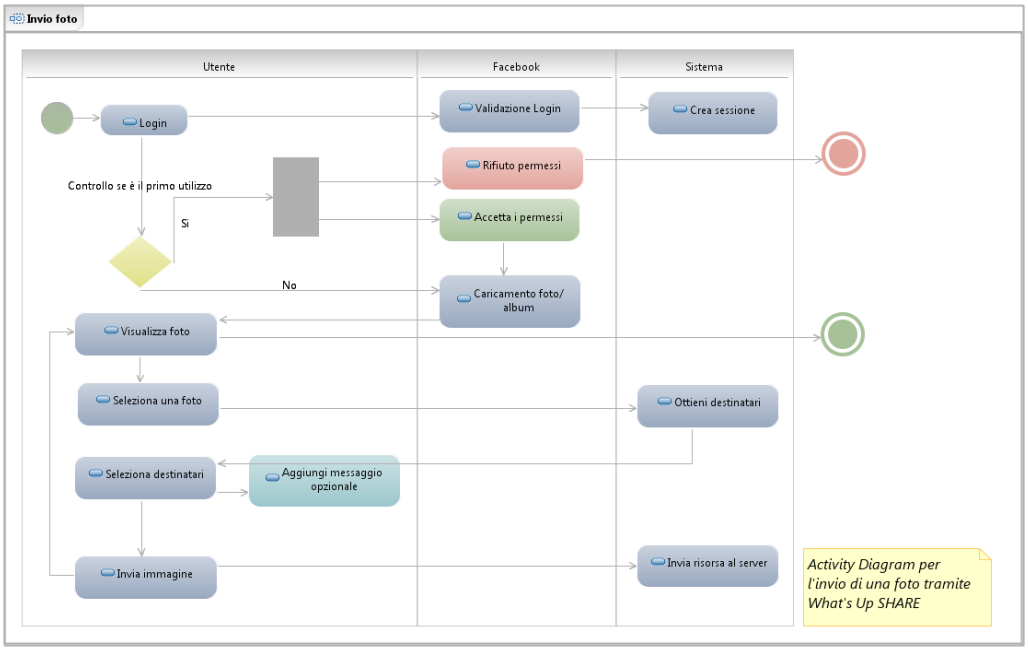


Figura 5.2

Una volta effettuato il login con Facebook, il sistema provvede al caricamento delle foto del proprio account. Per permettere una visualizzazione rapida ma pur sempre completa, ho deciso di suddividere la pagina in due parti, la prima contenente un'insieme scrollabile di immagini, più precisamente, un massimo di dieci immagini per ogni album. La seconda parte invece contiene l'insieme degli album dell'utente, ognuno rappresentato dalla relativa immagine cover; tramite un click sull'immagine, l'utente può visualizzare tutte le immagini contenute nell'album.
Ad ogni immagine é associato un bottone che permette di iniziare la procedura di invio dell'immagine. Il risultato di quanto descritto é rappresentato nello screenshot di figura 5.3. Ogni bottone non é altro che un link alla pagina di gestione degli invii, a cui viene passato l'ID della foto a cui é associato in questo modo:

```
1 <a href="domain url/modal/ID"></a>
```

Il router associato provvederà alla generazione dell'oggetto View per la relativa pagina modal, passandovi l'ID dell'immagine Facebook. La finestra per l'invio altro

non che un elemento *div* nascosto all'interno della pagina principale, attraverso l'uso di Bootstrap. Alla prima pressione del tasto *invia*, la View provvederà a caricare il template relativo, generando il contenuto e visualizzando la finestra all'interno dell'elemento precedentemente in *'hide'*. Le successive richieste della finestra di invio riutilizzeranno la View precedentemente creata, cambiando eventualmente il contenuto dinamico del template associato. Questa soluzione é stata attuata per il seguente motivo: la gestione degli eventi all'interno della finestra modale ( selezione dei destinatari, invio dell'immagine, cancellazione della procedura) é interamente gestita dalla View. L'eventuale creazione di nuove istanze View per ogni finestra avrebbe fatto si che ogni evento fosse gestito il numero di volte pari alle View presenti. Per esempio, dopo aver aperto e chiuso tre volte la finestra di invio, il successivo invio di un immagine sarebbe stato eseguito quattro volte! Un'altra possibile soluzione sarebbe stata quella di utilizzare i metodi di Backbone per rimuovere ogni volta l'intera View dal DOM così da eliminare anche i relativi binding agli eventi. In questo modo però ogni volta che viene creata una finestra per l'invio, é necessario rieffettuare il download del template dal server, operazione inutile anche se non dispendiosa vista l'esigua grandezza dei files .ejs ( in media 4-5 Kilobyte).

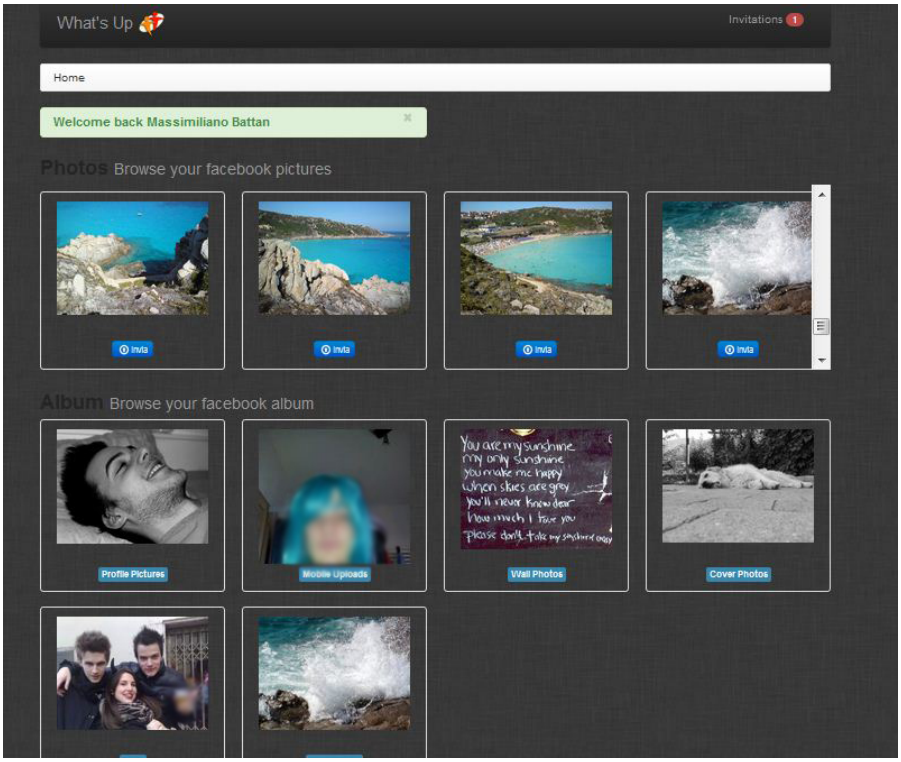


Figura 5.3: Main page di What's Up SHARE

Pari passo con la creazione della finestra di invio, il sistema provvede ad effettuare una XMLHTTPRequest[5] alle Api di What's Up, utilizzando il metodo *.ajax()* fornito da Jquery , per ottenere i possibili destinatari dell'immagine. I destinatari in questo caso sono tutti i possessori di un Kspace di cui l'utente che utilizza l'applicazione é Contributor. L'utente può selezionare più destinatari alla volta tramite i relativi checkbox, mentre se non sono presenti possibili destinatari, verrà visualizzato un messaggio di avviso. Opzionalmente l'utente può inserire un messaggio da allegare all'immagine ed infine provvedere all'effettivo invio della risorsa.

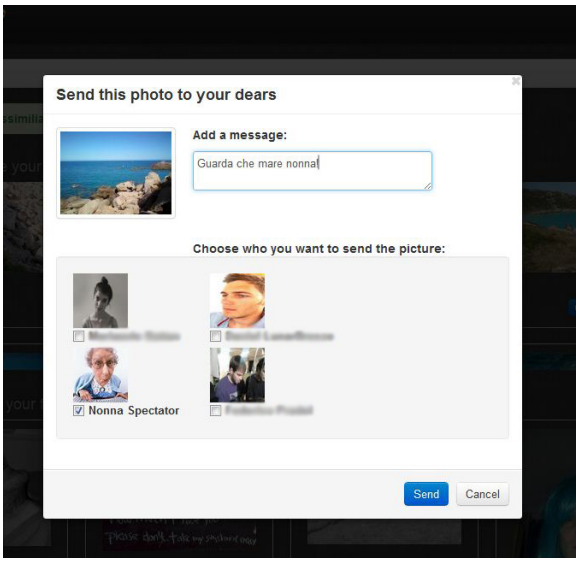


Figura 5.4: finestra modale per l'invio di una foto

L'invio della risorsa, come definito nelle reference delle Api, deve essere effettuato tramite il formato *multipart*. Durante la costruzione di un pacchetto HTTP, l'header del messaggio necessita sempre il parametro *content-type*, che definisce la tipologia di contenuto che conterrà il body della richiesta. La maggior parte delle richieste sia GET che POST effettuate dall'applicazione utilizzano come content-type *Application/Json* in quanto il body della richiesta é sempre formato da un oggetto json. Nel caso dell'invio vero e proprio di un file invece, le specifiche definite dall'RFC per quanto riguardano gli standart W3 prevedono l'utilizzo del formato multipart. Questo formato prevede la creazione di un body strutturato in questo modo:

```
1 Content-Type: multipart/mixed; boundary=4343982308
2
3 --4343982308
4 Content-Type: text/plain
```

[5]XMLHTTPRequest è stata introdotta nel paragrafo 2.2

```
5
6 Messaggio qui
7  --4343982308
8  Content-Type: application/octet-stream
9  Content-Transfer-Encoding: base64
10
11 PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA
12
13 --4343982308--
```

Come si può vedere, il body è suddiviso in parti, ognuna delle quali specifica il tipo del suo contenuto ( messaggio,file encodec in base64, ecc ). Le parti sono racchiuse all'interno del body tramite l'utilizzo di un boundary, valore necessario per delimitare ogni parte e per indicare la fine del messaggio. Il boundary non deve ovviamente essere presente in alcuna parte e tipicamente viene generato in modo random. In realtà SHARE non provvede ad effettuare un vero e proprio upload sul server dell'immagine di Facebook, bensi invia semplicemente l'url univoca per quella risorsa, quindi l'utilizzo del formato multipart non sarebbe necessario. Per permettere a tutte le applicazioni What's up Share di utilizzare la stessa Api, l'unica soluzione era quella di utilizzare un formato multipart sia per l'invio di file in base64 ( come nel caso dell'applicazione smartphone in cui viene inviata una vera e propria immagine scattata dal proprio telefono ) sia per l'invio di URL, come nel caso di What's Up Web.
Dopo aver provato diverse soluzioni senza successo per effettuare richieste multipart utilizzando lo stesso metodo *.ajax()* di Jquery , ho deciso di utilizzare in nuovo livello di Api sviluppato dal W3 Consortium, *XMLHTTPRequest Level 2*[6]. Questa nuova Api definisce l'interfaccia *FormData*, che permette di appendere messaggi multipart a richieste XHR.
Il risultato ottenuto e la soluzione descritta sono rappresentati dalla figura 5.5 e dal codice 5.1 rispettivamente. Una volta inviato il file, l'utente sarà avvisato se la procedura è andata a buon fine oppure no, tramite una finestra animata.

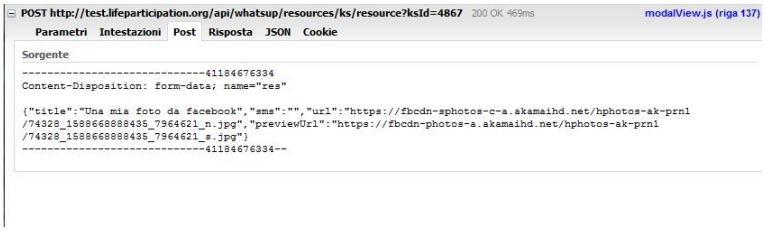


Figura 5.5: Esempio di un body in multipart Format durante l'invio di un'immagine

[6]References fornite da W3C sono disponibili all'indirizzo http://www.w3.org/TR/XMLHttpRequest/

**Listing 5.1: Sending Resource with POST and multipart format**

```
1         var baseURL = '/api/whatsup/resources/ks/resource';
2         var self = this;
3        var params = $.param({
4           ksId: receivers
5        },true);
6        var photo = this.model.toJSON();
7        var xhr = new XMLHttpRequest();
8        var fd = new FormData();
9        var request = {
10          Auth : localStorage.sessionId
11       };
12       var resource = {
13          title : 'Una mia foto da facebook',
14          sms : ($('#messageArea').val()),
15          url : photo.source,
16          previewUrl : photo.picture
17       };
18       fd.append('res',JSON.stringify(resource));
19       xhr.open( 'POST', baseURL+'?'+params, true );
20       xhr.setRequestHeader('Auth',localStorage.sessionId);
21       xhr.send( fd );
22       xhr.onreadystatechange = function(){
23          if (this.readyState == 4 ){
24     if (this.status == 200){
25                   $('#modalAftersend').modal('show');
26                   $('#ok').show();
27                  $('#modalAftersend').delay(3000).fadeOut
                        (1000,function(){
28                       $(this).modal('hide');
29                   });
30               }else {
31                   $('#modalAftersend').modal('show');
32                   $('#fail').show();
33                   $('#modalAftersend').delay(3000).fadeOut
                        (1000,function(){
34       $(this).modal('hide');
35     });
36               }
37          }
38 }
```

## 5.4 Gestione degli inviti tramite Share

Attraverso la procedura di invito, il possessore di un Kspace può permettere a chiunque di diventare un contribuente del proprio spazio; se l'invito viene accettato egli potrà iniziare fin da subito a ricevere gli aggiornamenti dalla persona invitata.
Grazie al contrassegno in alto a destra visibile nella figura 5.3, l'utente può vedere il numero degli eventuali inviti che ha ricevuto. Alla pressione dell'etichetta, viene visualizzata una finestra, dello stesso stile di quella per l'invio, in cui sono rappresentati gli eventuali inviti. In qualsiasi momento l'utente può provvedere ad accettare o rifiutare ognuna delle richieste, oppure semplicemente chiudere la finestra senza prendere alcuna decisione.
Ad ogni bottone, sia di accettazione che rifiuto, è associata la secret Key della relativa richiesta. Tutti i bottoni di rifiuto vengono legati ad una procedura di rifiuto, stessa cosa vale per i bottoni di accettazione. Nel caso in cui l'utente accettasse uno o più inviti, diventando così un Contributor dello spazio dell'invitante, quest'ultimo sarà subito disponibile tra i possibili destinatari nella procedura di invio.

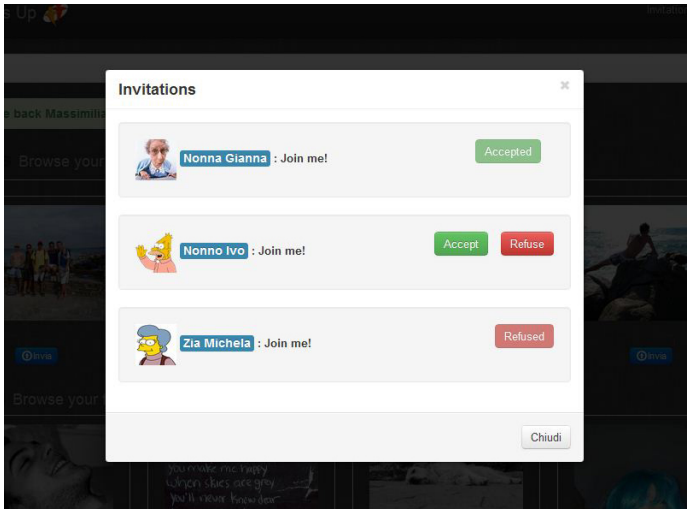


Figura 5.6: finestra modale per la gestione degli inviti

## 5.5 Display delle risorse

L'applicazione per il display delle risorse prevede, come per SHARE, il login tramite Facebook ed il relativo trattamento dei permessi. Una volta creata la sessione però, è necessario che il sistema faccia un controllo ulteriore sull'utente. Infatti, se si tratta del primo utilizzo del DISPLAY, e più in generale, di una qualsiasi applicazione della

piattaforma What's Up, non saranno sicuramente presenti risorse da visualizzare. Per di più, all'utente in questione non sarà associato nessun Kspace quindi è necessario crearne uno: in questo modo, senza che ne sia consapevole, l'utente viene predisposto fin da subito all'utilizzo dell'applicazione. Dopo aver creato il Kspace di cui l'utente diventa proprietario, l'unica azione possibile ed utile è quella di invitare dei Contributors, tramite l'operazione descritta nel paragrafo 5.6.
Nel caso in cui l'utente abbia già un proprio Kspace, l'applicazione provvede da subito alla visualizzazione delle eventuali risorse.

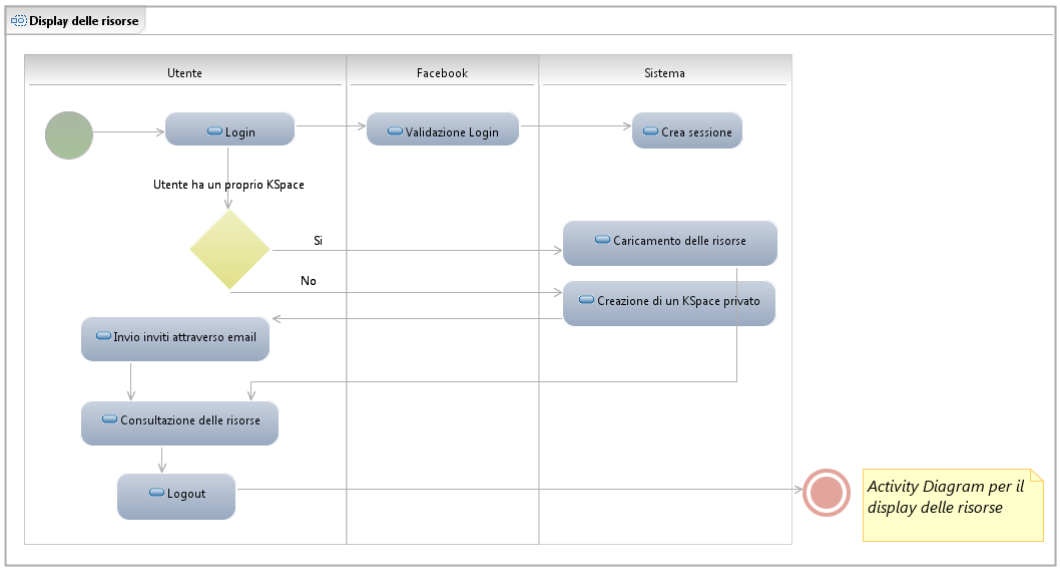


Figura 5.7

### 5.5.1 Architettura del carosello

Per la rappresentazione delle risorse ho utilizzato un carosello fornito da un plugin Javascript di Bootstrap, modificandolo interamente secondo le mie esigenze e definendo una specifica architettura per la realizzazione. Il plugin infatti fornisce solo una struttura base in cui un insieme predefinito di immagini può essere scorso verso sinistra oppure verso destra. Nella realizzazione del DISPLAY mi sono scostato da questa caratteristica ed ho preferito sviluppare un carosello dinamico, in grado di poter visualizzare contenuti caricati al momento e non predefiniti. Tutto ciò per due motivi:

1. Non effettuare richieste inutili alle api di What's Up che potrebbero solo rallentare l'applicazione. Invece di effettuare un'unica richiesta ed ottenere tutte le risorse del Kspace in esame, ho preferito seguire uno stile *ON DEMAND*.

2. Le Api What's Up non prevedono un metodo per ottenere il numero di risorse presenti nel Kspace, quindi ottenerle tutte sarebbe stato poco efficiente. In quel caso avrei dovuto chiedere un numero fisso di risorse: se la risposta contiene esattamente il numero di risorse richieste, forse ve ne sono altre presenti, altrimenti ho finito. Questo processo mi è sembrato poco pratico e sicuramente non garantisce ottime prestazioni.

Per ottenere questa funzionalità ho realizzato una rete di template strutturata in questo modo: Un template più esterno provvede a creare la struttura HTML del carosello, inglobandola all'interno della pagina principale (index.html). Al suo interno, gli elementi del carosello sono caricati attraverso un altro template, che si occupa di renderizzare, ad ogni request sul server Api, le risorse ottenute. Questo template inoltre si occupa di gestire la tipologia della risorsa: il display infatti deve fornire la possibilità di visualizzare immagini, messaggi, oppure immagini con messaggi, quindi non può essere utilizzata la stessa struttura per risorse diverse. Nel caso di messaggi il template provvede a caricare una pagina di sfondo, in stile carta, su cui viene stampato il messaggio con una buona formattazione, mentre nel caso di immagini o immagini con messaggio, questa viene inserita all'intero di una cornice con il messaggio stampato su un elemento *div* in stile semi trasparente. Le due tipologie sono visibili dalle immagini seguenti.

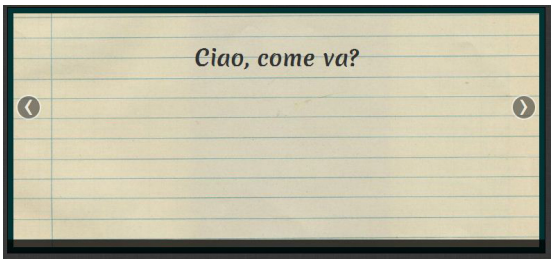


Figura 5.8: Design messaggio


Figura 5.9: Design immagine con messaggio

La caratteristica più importante che deve essere rispettata dal carosello è la temporalità: le risorse infatti devono essere presentate all'utente in ordine temporale, dalla più recente alla più vecchia. L'Api utilizzata per ottenere le risorse fornisce due opzioni atte allo scopo: ottenere una risorsa prima o dopo una determinata data e ordinare le risorse in modo ascendente o discendente secondo la data. Detto ciò descriverò ora l'architettura alla base del carosello.
Il carosello si basa sul concetto di *finestra*: una finestra esprime un insieme di risorse che può essere caricato in qualsiasi momento nel template per essere visualizzato all'interno del carosello. Di default, la grandezza della finestra che ho scelto è quattro,

ma può essere cambiata a piacere grazie ad uno script che contiene le costanti dell'applicazione. Ogni richiesta alle Api è fatta con l'intento di riempire una determinata finestra che potrà essere richiesta a breve. Le finestre sono in tutto cinque, e possono coincidere a seconda della situazione:

1. **Finestra corrente**: è la finestra correntemente caricata nel template.
2. **Finestra successiva**: è la finestra temporalmente successiva a quella attualmente visualizzata.
3. **Finestra precedente**: è la finestra temporalmente precedente a quella attualmente visualizzata.
4. **Prima finestra**: è la finestra che contiene le risorse più vecchie in assoluto.
5. **Ultima finestra**: è la finestra che contiene le risorse più recenti.

Il passaggio da una finestra all'altra avviene in quattro modi:

1. Tramite il tasto *Newest messages*: in questo caso il sistema provvederà a prendere l'ultima finestra e farla diventare la finestra corrente. Dopo di che viene aggiornato il template.
2. Tramite il tasto *Oldest messages*: in questo caso il sistema provvederà a prendere la prima finestra e farla diventare la finestra corrente. Dopo di che viene aggiornato il template.
3. Tramite i tasti *avanti e indietro*: il sistema provvederà a caricare nella finestra corrente la finestra successiva o precedente rispettivamente, ricalcolando eventualmente le altre.

Il principio di quest'architettura è quello di fare in modo che la prossima finestra che verrà richiesta sia già disponibile nel sistema. All'avvio infatti vengono caricate ultima finestra e finestra corrente, che coincidono, oltre a finestra precedente e prima finestra; finestra successiva è ovviamente vuota. Se l'utente esaurisce la finestra corrente scorrendo a sinistra, il sistema non deve chiedere nuove risorse ma provvede a caricare la finestra precedente già pronta. Durante la transazione, l'applicazione effettua all'insaputa dell'utente ulteriori richieste per ottenere le finestre mancanti, in modo che siano sempre disponibili le risorse a cui l'utente può accedere nell'imminente futuro. L'architettura viene riassunta dall'immagine 5.10.

Il design descritto fino a questo punto, insieme al caching delle immagini del paragrafo 5.5.2, garantisce all'intera applicazione una notevole velocità, evitando il più possibile ogni tipo di caricamento visibile all'utente e permettendo di ottenere in modo rapido le risorse più sensibili ( più recenti e più vecchie ).

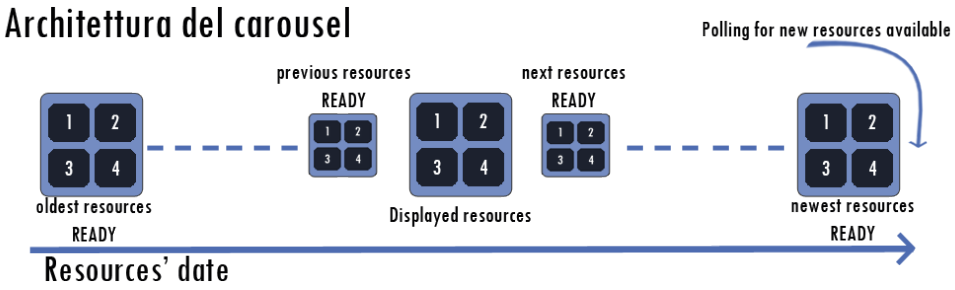


Figura 5.10

### 5.5.2 Caching delle immagini

Oltre alle richieste On Demand verso il server Api, ho implementato all'interno del carosello una cache personale utilizzata dall'applicazione per pre-caricare le immagini della finestra corrente. In questo modo, ho cercato di evitare all'utente la visione del caricamento di una risorsa dovuta al download dell'immagine, nel caso questo non fosse istantaneo. Il caching delle immagini viene fatto soltanto su risorse di tipo immagine o immagine con messaggio per ovvi motivi, e viene eseguito soltanto sulla finestra corrente.
L'implementazione prevede inizialmente la memorizzazione eseguita in uno script, quindi all'interno del browser, di uno stack di elementi HTML di tipo *img*, settando opportunamente il campo *src*: in questo modo il browser provvede ad effettuare il download dell'immagine, per poterla inserire all'interno dell'elemento. La chiamata a questa funzione di caching viene effettuata all'interno del template che si occupa di creare ad ogni occorrenza il contenuto del carosello. Dopo aver caricato il template ed averlo inserito all'interno del carosello, si procede a riempirlo con gli elementi img contenuti nello stack. In background il browser provvederà a scaricare, se non lo ha ancora fatto, le immagini. In questo modo la navigazione dell'utente all'interno della finestra corrente è fluida e senza intoppi, perché, download speed[7] permettendo, le immagini sono tutte pronte all'occorrenza.

```
1 function cacheImage(resource) {
2     var cacheImage = { el : document.createElement('img'),
3                        res : resource.id
4                      };
5     cacheImage.el.src = resource.url;
6     WD.cache.push(cacheImage);
7 }
```

[7]La velocità di connessione in download disponibile al momento dell'utilizzo.

Ogni volta che viene ricaricato e ridisegnato il template, viene eseguito il seguente ciclo per inserire le immagini pre-caricate all'interno del carosello:

**Listing 5.2: Search for cached images and append them to the document**

```
1  for (var i = 0 ; i < this.currentWindow.length; i++){
2          //if the image is not an sms I have cached it and I
               can ask for it and append to the DOM
3          if (this.currentWindow[i].type!="SMS"){
4             var element = self.currentWindow[i];
5             //loop on the cached image to find the right one
6             $.each(WD.cache,function(index,cachedImg){
7                if(element.id==cachedImg.res){
8                   //append it to the DOM
9                   $('#'+self.currentWindow[i].id).append(WD.
                       cache[index].el);
10                  //remove from the cache (it will be ready
                       for the current Window)
11                  WD.cache.splice(index,1);
12                  //stop the loop
13                  return false;
14               }
15            });
16
17         }
18      }
```

### 5.5.3 Gestione delle nuove risorse

Un ulteriore funzionalità offerta dal Display consiste nel ricevere, durante l'utilizzo, eventuali nuove risorse inviate all'utente dai sui Contributors, sia attraverso l'applicazione web Share, sia attraverso le applicazioni per smartphone.
All'interno dell'applicazione, ad intervalli regolari ( modificabili tramite script delle costanti ), viene eseguita una funzione di aggiornamento, per controllare se ci sono nuove risorse disponibili. Grazie all'ultima finestra, in qualsiasi momento è possibile ricavare la data dell'ultima risorsa presente, richiedendo al server tramite Api eventuali risorse successive. Nel caso siano presenti nuove risorse, se l'utente si trova già sull'ultima finestra queste verranno automaticamente visualizzate nel carosello, sostituendo opportunamente la finestra. Nel caso invece in cui l'utente non si trovi nell'ultima finestra, comparirà un bottone lampeggiante per alcuni secondi ad indicare la presenza di nuovi messaggi ricevuti; alla pressione, il carosello si sposterà automaticamente alla visualizzazione dei nuovi messaggi.

La piattaforma What's Up prevede che una risorsa inviata ad un Kspace contenga il campo *seen* impostato a false per default; una volta visualizzata dal proprietario del Kspace, sarà cura del programmatore impostare la risorsa come *vista*. La scelta che ho fatto è stata quella di impostare una nuova risorsa seen=true quando l'utente si sposta a sinistra oppure a destra del carosello: in quel caso sono sicuro che il messaggio appena ricevuto è stato effettivamente visto.

## 5.6 Inviti

Nel Display è presenta la possibilità di invitare chiunque a diventare un Contributor del proprio spazio. Per fare ciò l'utente deve utilizzare l'apposito tasto *Invite Friend* e fornire una mail valida all'interno della finestra modale che apparirà: dopo aver effettuato l'invio, il destinatario riceverà una mail in cui verrà avvisato dell'invito. Nel caso in cui l'utente non abbia mai usato nessun'applicazione della piattaforma What's Up, verrà sollecitato al download di una di queste. In caso contrario, potrà accettare l'invito tramite quanto descritto nella sezione 5.4 e procedere fin da subito all'invio di risorse all'utente.

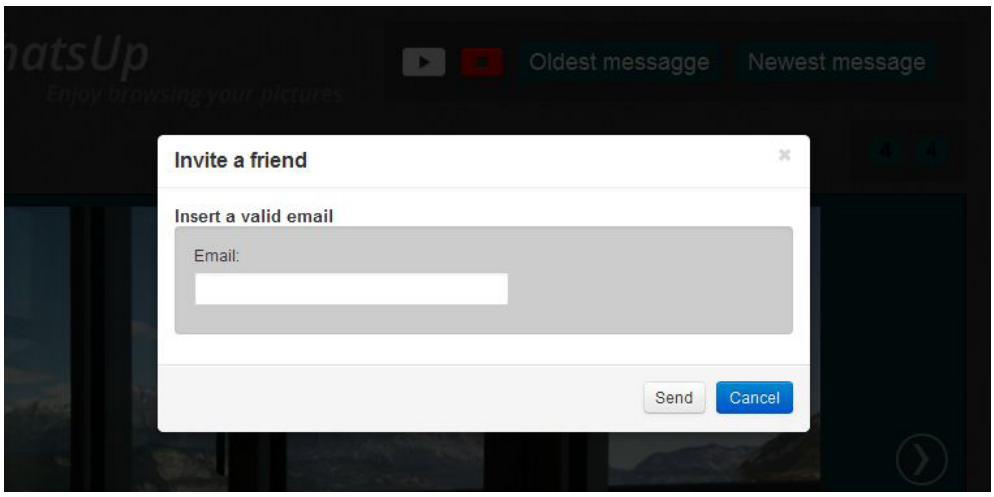


Figura 5.11: Finestra modale per l'invio di un invito

Il risultato finale dell'applicazione Display è rappresentato dell'immagine seguente.

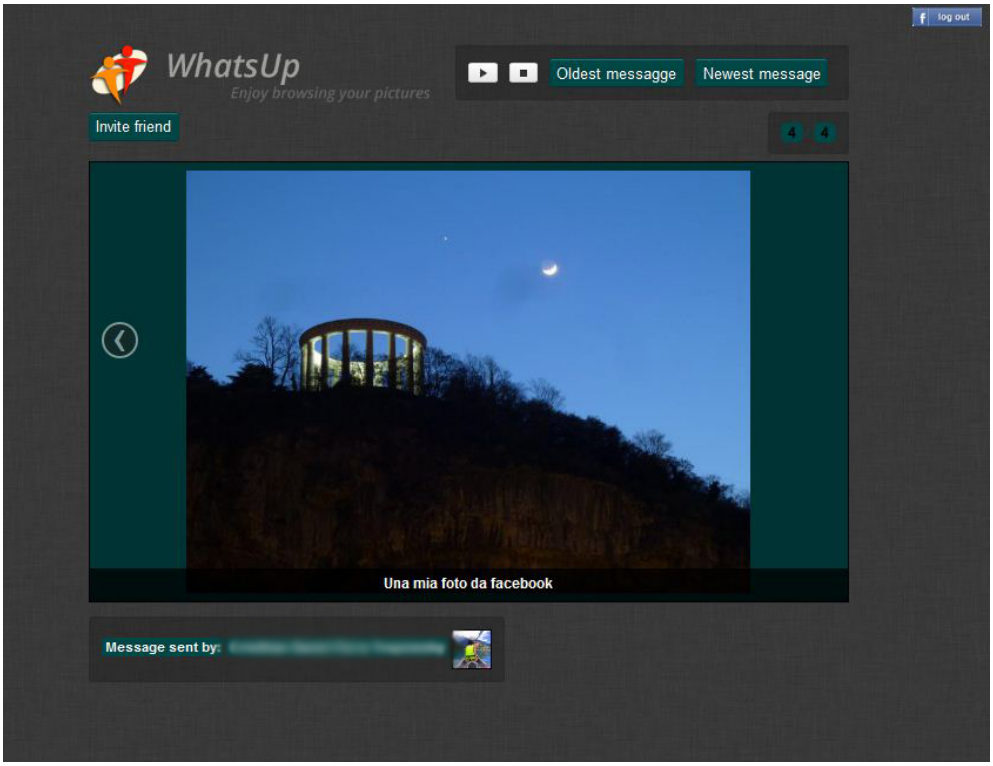


Figura 5.12: Main page dell'applicazione Display

# Capitolo 6

# Conclusioni

Il risultato ottenuto ha come obbiettivo principale quello di garantire il porting su web di tutte le funzionalità della piattaforma What's Up. I requisiti mostrati nel Capitolo 3 sono stati ampiamente rispettati e portati a termine ma il lavoro non finisce qui. Ulteriori caratteristiche verranno aggiunte in seguito o sono attualmente in fase di elaborazione, in quanto il feedback degli utenti e le idee degli sviluppatori di certo non mancano. La presenza di un tutorial prevista per il primo utilizzo dell'utente, la possibilità di rilasciare feedback negativi o positivi ad una risorsa, l'opportunità di invitare nel proprio Kspace gli amici di Facebook, sono solo alcune delle funzionalità che verranno presto implementate nell'applicazione web discussa fino ad ora.

What's Up Web è in questo momento in fase di test, disponibile all'indirizzo *http://test.lifeparticipation.org/whatsup/web*; utilizzata all'interno del team Life Participation, presto sarà disponibile sui server Production per essere pubblicata in via definitiva, contemporaneamente all'inserimento di What's Up Web Display all'interno del Chrome Web Store.

# Ringraziamenti

All'età di dieci anni già avevo scelto la mia strada: volevo saper fare tutto con il computer. Nella mia ingenuità sicuramente una grande aspirazione, troppo generale in fin dei conti, ma a distanza di anni, sono felice di aver fatto un vero primo passo verso quell'obbiettivo. Il mondo informatico è immenso, irraggiungibile nel complesso, ma rimane sempre il più bello che io conosca e quello su cui sto costruendo la mia vita.

Ogni passo non va mai fatto da solo, per questo mi sento di dover ringraziare coloro che mi hanno aiutato nel portare a termine questo traguardo. Prima di tutto ringrazio i miei genitori, Claudio e Jolanda, senza cui sicuramente non sarei arrivato a questo punto e le mie sorelle Lisa e Mariasole per essermi sempre accanto: questo è per voi.

Ringrazio il professor Marchese, che mi ha dato l'opportunità di svolgere il tirocinio prima, e il progetto per la tesi poi. Grazie a Marcos Baez per avermi integrato all'interno del gruppo di lavoro di Life Partecipation ed avermi dato la possibilità di sviluppare per loro questo progetto. Un sentito ringraziamento va a Cristhian Parra, colui che mi ha seguito durante tutto lo sviluppo, disponibile in ogni momento, e a tutti gli altri componenti del team tra cui Ivan e Michele per l'aiuto nel momento di bisogno.

Un grazie agli amici colleghi e a quelli non, senza di voi non sarei quello che sono.

Infine un grazie a te, anche se le cose non vanno sempre come vorremmo, tu mi sei stata vicino finchè hai potuto, dandomi la forza che tante volte mi mancava.

Adesso è il momento di iniziare a fare un altro passo, in bocca al lupo a me!

.

# Bibliografia

[1] Wikipedia. Web 2.0. `http://en.wikipedia.org/wiki/Web_2.0`, 2011.

[2] Keith W. Ross James F. Kurose. *Computer Networking A Top-Down Approach 4th edition*. Addison Wesley, 2005.

[3] Sam Ruby Leonard Richardson. *RESTful Web Services*. Oreilly, 2007.