

Università degli Studi di Trieste

---

Facoltà di Ingegneria  
Corso di laurea in Ingegneria Informatica  
Laurea specialistica



# **PROGETTO E REALIZZAZIONE DI UN APPLICATIVO “SOCIAL NETWORK” PER LA CONDIVISIONE DI EVENTI**

Candidato:  
Maurizio Pozzobon

Relatore:  
Prof. Maurizio Fermeglia

*Anno Accademico 2010/11*

*Dedico questo mio lavoro alla mia famiglia,  
che mi ha sostenuto e sopportato durante questi anni  
e a mia moglie Maria Rosa,  
che mi ha amato ed aiutato.*

*Ringrazio, inoltre, il mio relatore Fermeglia Maurizio,  
per avermi guidato durante il corso di questo progetto.*

*Maurizio*

# Indice

Introduzione.....	4
Capitolo 2 - Requisiti di InformaMi.....	7
Capitolo 3 - Tecnologie utilizzate.....	9
3.1 Java.....	9
3.2 Google App Engine.....	9
3.3 Play! Framework.....	10
3.3.1 GAE.....	10
3.3.2 Siena.....	10
3.3.3 CRUD.....	10
3.4 Facebook Connect.....	10
3.5 RestFB.....	11
3.6 Imgur.....	11
3.7 Google search.....	11
Capitolo 4 - Progettazione dell'applicazione.....	12
4.1 Progettazione database.....	13
4.1.1 Definizioni delle entità e relazioni.....	13
4.1.2 Analisi delle entità.....	19
4.1.3 Analisi delle relazioni e delle cardinalità.....	21
4.1.4 Progettazione fisica del database.....	25
4.2 Progettazione moduli funzionali.....	28
4.2.1 Application .....	29
4.2.2 Events.....	30
4.2.3 Places.....	31
4.2.4 Tags .....	31
4.2.5 Photos.....	32
4.2.6 Actions.....	32
4.2.7 Medals.....	32
4.3 Progettazione ed implementazione dell'interfaccia grafica.....	33
4.3.1 Delineazione degli elementi di base.....	33
4.3.2 Implementazione interfaccia grafica.....	36
Capitolo 5 - Implementazione moduli funzionali.....	40
5.1 Gestione login Facebook.....	40
5.2 Hosting immagini.....	44
5.3 Gestione ricerca per zona .....	47
Capitolo 6 - Gamificazione.....	51
Capitolo 7 - Test dell'applicazione.....	53
Conclusione.....	58
Bibliografia.....	60
Appendice A – Allegato codice.....	61

# Introduzione

Si intende creare un applicazione web in grado di integrarsi con Facebook, il più popolare dei social network, che permetta agli utenti di pubblicare eventi futuri ed interagire con essi. Questo viene fatto per rendere più semplice scoprire eventi interessanti nella propria città.

L'obiettivo finale è di realizzare e pubblicare un sito web in cui gli studenti universitari possano scoprire nuovi eventi e locali nella loro città. Questo sito deve essere di facile utilizzo e deve evidenziare gli eventi popolari, invece, quelli impopolari vanno nascosti. Il sito, inoltre, deve essere di interesse anche per coloro che pubblicizzano degli eventi, cioè devono poterlo considerare una valida alternativa ai volantini distribuiti all'ingresso della mensa.

L'idea per questa tesi è stata concepita per delle motivazioni puramente personali, volevo che i miei amici e parenti potessero capire cosa ho fatto come tesi. La maggior parte delle tesi universitarie sono fuori dalla portata dei non addetti ai lavori e quindi ho voluto realizzare qualcosa di utile per tutti. A quello scopo ho pensato: *“cos'è che 500 milioni di persone utilizzano quasi ogni giorno? Facebook”*. Non potendo creare Facebook, ho deciso di basarmi sul suo successo per migliorare un aspetto di Facebook che non funziona correttamente.

Facebook è utilizzato quotidianamente da molte persone per condividere pensieri, foto e situazioni della loro vita, ma risulta poco pratico per trovare informazioni riguardanti un aspetto fondamentale della vita universitaria.

Molti studenti universitari il mercoledì sera vanno a divertirsi al cosiddetto 'mercoledì universitario'. I proprietari dei locali cercando di attirare la loro attenzione inondano l'università e le strade della città con volantini promozionali. Questo si traduce con un inutile spreco di carta e in studenti presi d'assalto all'entrata della mensa o in altri luoghi frequentati per dare loro volantini che poco dopo butteranno. Inoltre, anche se lo studente fosse interessato alle informazioni che i locali vogliono pubblicizzare, in genere se ne dimenticherà prima di dover scegliere dove andare la sera. Questo accade perché i proprietari dei locali lo contattano quando la sua preoccupazione principale è cosa mangiare o che lezione avrà dopo. Non dove andare a ballare.

Inoltre, l'unico modo per gli studenti di venire informati di un evento tramite Facebook è quello di essere invitati da un proprio amico. Questo nella maggior parte dei casi non è adatto, in quanto risulta difficile che uno studente venga invitato in una data nella quale non aveva già altri programmi, oppure l'evento potrebbe essere ospitato in un locale troppo

lontano dall'abitazione dello studente, infine l'evento al quale si è invitati deve essere di proprio gradimento (non sempre si hanno gli stessi gusti dei propri amici). Per questo motivo scovare eventi su Facebook risulta poco pratico. La vera utilità degli eventi su Facebook è quando si vuole organizzare una festa di compleanno o comunque un evento privato.

Quali altre alternative hanno gli studenti che vogliono uscire una sera ma non sanno cosa fare?

- possono cercare in giro per l'università dei volantini che promuovano una festa il giorno che gli interessa
- possono parlare con i propri amici e vedere se salta fuori qualcosa di interessante

Nell'era del web 2.0 questi metodi mi sono sembrati un po' obsoleti e macchinosi.

Ispirato a siti come digg.com (che permette la condivisione di link) e stackoverflow.com (che è un'evoluzione dei forum per domande e risposte) ho deciso di creare una piattaforma per la condivisione di eventi. Questa deve permettere di trovare facilmente degli eventi di vario tipo (feste, concerti, esposizioni, rappresentazioni teatrali...) in una data ed in un luogo specifico. Inoltre, è necessario che evidenzii quali sono gli eventi più popolari.

Nel corso di questa tesi sarà necessario:

- Analizzare in dettaglio i requisiti dell'applicativo
- Decidere quali tecnologie utilizzare per sviluppare il sistema
- Progettare e realizzare l'applicativo
- Realizzare un'interfaccia grafica gradevole per il mercato scelto
- Pubblicare l'applicativo su un sito raggiungibile dal pubblico

Questo elaborato verrà strutturato come segue:

- Capitolo 2: verranno analizzati i requisiti per l'applicativo, ricavati tramite l'intervista di 10 studenti dell'università di Trieste
- Capitolo 3: verranno elencate le tecnologie utilizzate per sviluppare l'applicativo, per ognuna di esse verrà data una breve descrizione ed il motivo per il quale è stata scelta

- Capitolo 4: verrà presentata la progettazione dell'applicazione, questo capitolo è diviso in tre grossi sottocapitoli in cui si progettano singolarmente il database, i moduli funzionali e l'interfaccia
- Capitolo 5: presenta l'implementazione di tre moduli funzionali significativi dell'applicazione
- Capitolo 6: verrà spiegato il concetto di *gamificazione* e si delinea in che misura verrà utilizzato nella progettazione dell'applicativo
- Capitolo 7: sarà presentato il concetto di automatizzazione dei test del sistema, presentando due tipi diversi di test e degli esempi di test realizzati nel corso della tesi

Il vincolo principale per questo progetto è la necessità di renderlo utilizzabile da un ampio pubblico, quindi deve essere sviluppato e distribuito su una piattaforma pubblicamente accessibile e che possa gestire un elevato numero di visitatori. Un ulteriore vincolo è l'aspetto economico. Per realizzare questo progetto non si sono ricevuti finanziamenti di alcun tipo, per questo motivo si cercherà di spendere il meno possibile. Si è scelto inoltre, per motivi più ideologici che altro, di utilizzare tecnologie *opensource* durante lo sviluppo del progetto. Quindi, oltre alle tecnologie elencate nel Capitolo 3 per lo sviluppo del progetto di lavorerà su un sistema operativo Linux (Ubuntu 11.04) e con l'ambiente di sviluppo Eclipse Helios. Quest'ultima scelta come è stato detto è puramente a livello ideologico. Il sottoscritto preferisce, quando possibile, utilizzare tecnologie *opensource* in quanto gli danno la garanzia di non essere costretto a dover dipendere da un'unica compagnia per gli aggiornamenti degli strumenti che usa per lavorare, inoltre, questi strumenti sono spesso supportati da un'enorme comunità di programmatori felici di programmare per il piacere di farlo.

Si è deciso di intitolare il risultato di questo progetto *InformaMi* quindi nel corso dell'elaborato verrà spesso utilizzato questo nome per indicare l'applicativo web.

## Capitolo 2 - Requisiti di InformaMi

*In questo capitolo verranno elencati i requisiti necessari ad InformaMi per poter soddisfare le necessità degli studenti universitari. Per fare ciò sono stati informalmente intervistati 10 studenti.*

Si è quindi deciso di creare un servizio per la condivisione di eventi. Analizzando i possibili utilizzatori di questo servizio si è deciso di focalizzare l'attenzione sul mercato di ragazzi e ragazze di un'età compresa tra i 20 ed i 28 anni che frequentano un'università. Questa decisione è stata presa in quanto si ritiene che la vita dello studente universitario comporti diversi stimoli a frequentare locali regolarmente, inoltre, il passaparola tra gli studenti è molto elevato, quindi un servizio come InformaMi può diffondersi velocemente e conquistare questa nicchia.

Allo scopo di rendere il servizio il più vicino possibile alle necessità del mercato scelto sono stati informalmente intervistati 10 studenti. A questi è stato presentato un'idea di base delle funzionalità del servizio ed è stato chiesto il loro parere, sono stati chiesti consigli su come migliorarlo, funzionalità che vorrebbero vedere e funzionalità che preferirebbero non avesse.

Tra le funzionalità richieste le più frequenti sono state:

1. Integrazione con Facebook. L'onnipresenza di Facebook nella vita degli universitari è evidente dal fatto che l'integrazione con Facebook è la funzionalità più richiesta tra gli intervistati. Vogliono poter accedere ad InformaMi con il loro account Facebook, condividere gli eventi con i propri amici, taggare i propri amici sulle foto pubblicate negli eventi, pubblicare anche su Facebook le foto che pubblicano su InformaMi...
2. Ricerca per data. La ricerca per data è probabilmente la funzionalità più ovvia di questo tipo di applicazione web. Gli utenti vogliono poter visualizzare tutti e solo li eventi tenuti in una determinata data e possibilmente ora (sera piuttosto che mattina o pomeriggio).
3. Ricerca per luogo. Un'altra funzionalità fondamentale è la ricerca per luogo. È inutile venire informati di un evento molto popolare a Roma quando il visitatore abita a Trieste e sta cercando qualcosa da fare quella sera stessa.
4. Ricerca per tipo di evento. InformaMi deve permettere di pubblicizzare diversi tipi di eventi, dalle feste nei locali agli eventi teatrali, ma gli utenti devono poter visualizzare solo i tipi di eventi ai quali sono interessati.
5. Elevata categorizzazione degli eventi. Questa non è mai stata una richiesta diretta

degli intervistati, ma la si è potuta intuire dai diverse richieste fatte. Gli utenti non vogliono solo distinguere tra un evento musicale ed uno cinematografico, ma anche che genere di musica ci sarà, qual'è la probabile età media dei partecipanti, se il locale è costoso o meno, ecc...

6. Commentare gli eventi. Prima e dopo che si sia tenuto un evento gli utenti vogliono poter dare il proprio parere sull'evento.
7. Commentare i locali in cui sono tenuti gli eventi. Gli utenti vogliono poter commentare l'organizzazione di un evento passato ed i diversi aspetti del locale che l'ha ospitato.
8. Condivisione foto legate agli eventi. Gli intervistati desiderano poter pubblicare delle foto riguardanti un evento oltre alle foto ufficiali di colui che lo organizza.

Funzionalità da evitare:

1. Spam. Gli utenti sono stufo di quei siti che inviano email e notifiche su Facebook inutili e troppo spesso. Quindi hanno richiesto che InformaMi non lo facesse.
2. Messaggistica interna. Gli intervistati non ritengono necessario l'implementazione di un sistema di messaggistica interna agli utenti. Apprezzerebbero in cambio la possibilità di mandare email o messaggi su Facebook.



## Capitolo 3 - Tecnologie utilizzate

*In questo capitolo vengono presentate le tecnologie utilizzate durante la realizzazione di InformaMI, inoltre, viene spiegato il motivo per il quale sono state scelte.*

Analizzando gli obiettivi inizialmente predisposti per il progetto e le richieste risultate dalle interviste ai possibili clienti si sono poi scelte le tecnologie da utilizzare per sviluppare il progetto. Queste sono le seguenti:

### 3.1 Java

Come linguaggio di programmazione si è scelto di utilizzare Java in quanto è un linguaggio stabile, ampiamente utilizzato nel mondo e con svariate librerie a disposizione che possono facilitare lo sviluppo del progetto.

### 3.2 Google App Engine

Essendo ciò che si andrà a sviluppare un'applicazione web è fondamentale che questa venga poi effettivamente distribuita su un server web accessibile al pubblico.

Vi sono tre alternative per ottenere questo risultato:

- un server privato
- un server privato virtuale
- hosting sul cloud

Le prime due alternative non risultano economicamente giustificabili visto che il carico richiesto, almeno inizialmente, sarà molto ridotto.

L'utilizzo dei cloud permette di avere dei costi di hosting proporzionati all'effettivo utilizzo, per questo motivo è stata scelta questa alternativa.

All'inizio del 2008 Google ha cominciato a fornire un servizio di hosting cloud per applicazioni Java e Python. Le condizioni economiche erano molto vantaggiose e il servizio fornito di qualità. Si è ritenuto Google App Engine (il servizio hosting di Google) l'alternativa migliore per il nostro progetto. A giugno 2011 Google ha annunciato dei cambi alle condizioni economiche e questo può in futuro portare a dover cambiare fornitore. Questo avvenimento giustifica pienamente l'utilizzo del modulo Siena durante lo sviluppo dell'applicazione.

### **3.3 Play! Framework**

Play è un framework web che aiuta lo sviluppo di applicazioni web in Java. Il modo tradizionale di sviluppare applicazioni web in Java ed in altri linguaggi simili è alquanto laborioso, infatti, sono nati diversi linguaggi di programmazione per risolvere questo problema. Play è stato sviluppato per portare i benefici di questi più moderni linguaggi nel mondo Java, mantenendo così tutti i benefici del vasto ecosistema di librerie sviluppate attorno ad esso.

Play permette inoltre l'utilizzo di librerie opzionali, dette moduli, che forniscono funzionalità importanti, tra queste ci sono:

#### **3.3.1 GAE**

Questo modulo permette di utilizzare Google App Engine come hosting provider per le applicazioni sviluppate tramite Play.

#### **3.3.2 Siena**

Questo modulo serve da *wrapper* per l'API del motore database. Quindi permette di utilizzare le funzionalità del motore database nascondendone effettivamente i dettagli. Questo è molto utile nel caso si decida in futuro di spostare l'applicazione su un nuovo hosting provider (ad esempio il cloud di Amazon) e poter quindi sostituire l'attuale Datastore di Google con un database diverso, sia esso SQL o NoSQL.

#### **3.3.3 CRUD**

Questo modulo facilita la creazione del lato amministratore per l'applicazione, con semplici passaggi esso crea un'interfaccia che permette di creare, modificare ed eliminare le entità dal database.

### **3.4 Facebook Connect**

Facebook Connect è un API sviluppata da Facebook che permette a terze parti di accedere ai dati degli utenti Facebook. Nel applicativo in questione verrà utilizzato per non richiedere agli utenti di creare un account su InformaMi, ma permettergli di utilizzare il loro account Facebook.

Questo da un lato elimina i possibili problemi di sicurezza dovuti al dover memorizzare informazioni sensibili degli utenti. Dall'altro evita agli utenti di dover creare e ricordarsi una

nuova combinazione di nome utente e password.

### **3.5 RestFB**

RestFB è una libreria Java che maschera parte dell'API di Facebook per semplificare il suo utilizzo. In InformaMi verrà utilizzata esclusivamente per caricare i dati dell'utente loggato, ma le funzionalità che permette sono svariate e può risultare utile in un futuro sviluppo dell'applicativo.

### **3.6 Imgur**

Nonostante GAE sia risultato il miglior servizio di hosting, ha anch'esso dei limiti. In particolare, non permette di memorizzare file sul filesystem.

In alternativa ad un filesystem con permessi di scrittura GAE permette la memorizzazione dei file come blob nel Datastore. Questa funzionalità però è alquanto sperimentale e si sono riscontrati diversi problemi nel tentativo di implementarla. Per questo motivo si è deciso di affidarsi ad un servizio di hosting esterno.

Imgur permette la memorizzazione di immagini attraverso una semplice API. Visto che gli unici file che InformaMi ha il bisogno di caricare sono immagini (le immagini degli eventi/locali e le foto degli utenti) si è scelto di utilizzare questo servizio. Un altro possibile servizio era lo storage cloud di Amazon.

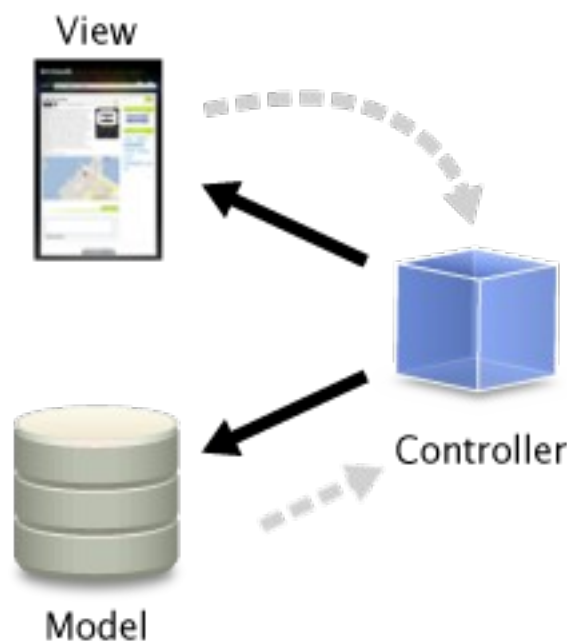
### **3.7 Google search**

Per permettere agli utenti di cercare un evento od un locale all'interno di InformaMi (oltre alla ricerca per luogo, tipo e data) si è deciso di avvalersi del motore di ricerca personalizzato di Google. Questo permette di restringere il campo dei risultati di Google ad un unico sito internet. Creando così un motore di ricerca interna molto avanzato in brevissimo tempo.

## Capitolo 4 - Progettazione dell'applicazione

*In questo capitolo vengono spiegati tutti i processi che si sono seguiti durante la progettazione dell'applicazione, in particolare per la progettazione del database, dei moduli funzionali e dell'interfaccia grafica.*

In base ai requisiti delineati al capitolo 2 e le tecnologie decise al capitolo 3 per la progettazione di InformaMi si è deciso di utilizzare l'architettura MVC (Model View Controller), dove si cerca di separare in tre elementi distinti i dati, la logica funzionale e l'interfaccia grafica. Uno dei vantaggi di questo modo di strutturazione è la possibilità di sostituire uno dei tre elementi senza modificare, o con poche modifiche, sugli altri. Ad esempio sarà possibile creare un applicativo da utilizzare sui dispositivi mobili (iPhone, Android...) senza modifiche al modello e le modifiche al controller probabilmente saranno legate all'aggiunta di funzionalità specifiche per gli smartphone.



*Figura 1: Architettura MVC*

Si può vedere in figura 1 come grazie alla separazione di vista e modello, e grazie all'interfacciamento da parte del controller, non è necessaria l'interazione diretta tra i due.

Avendo stabilito l'architettura di base del sistema si è proceduto per prima cosa alla progettazione del database che andrà a memorizzare i dati, successivamente si sono progettati i moduli funzionali ed in fine l'interfaccia grafica.

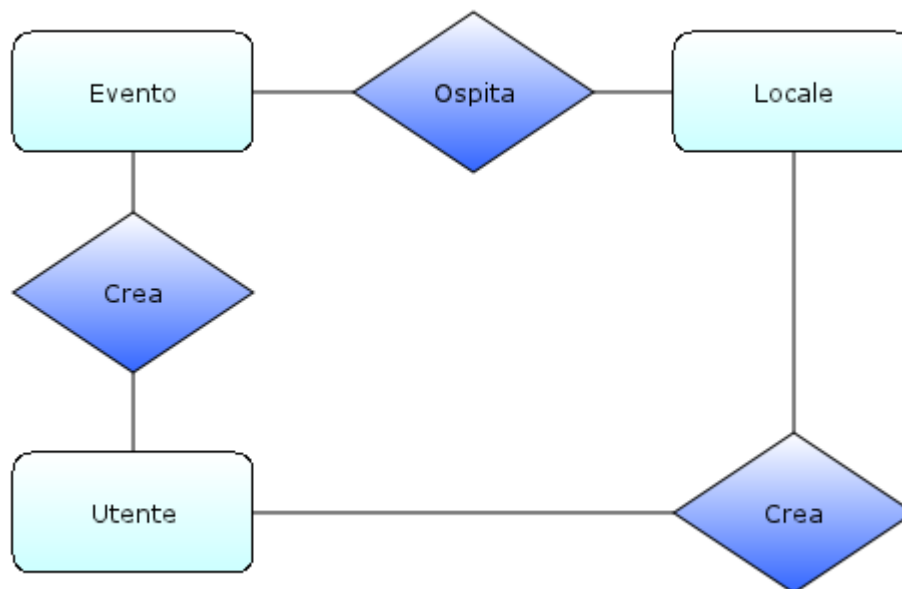
## 4.1 Progettazione database

### 4.1.1 Definizioni delle entità e relazioni

La progettazione del database è stata effettuata a passi, partendo da uno schema il più semplice possibile e poi aggiungendo i dettagli mano a mano.

Le tre entità fondamentali di InformaMi sono:

- Utente – è un visitatore registrato del sito, può creare gli eventi ed i locali
- Evento – è appunto l'evento da pubblicizzare, viene ospitato in un locale e creato da un utente
- Locale – ospita gli eventi, viene creato da un utente.



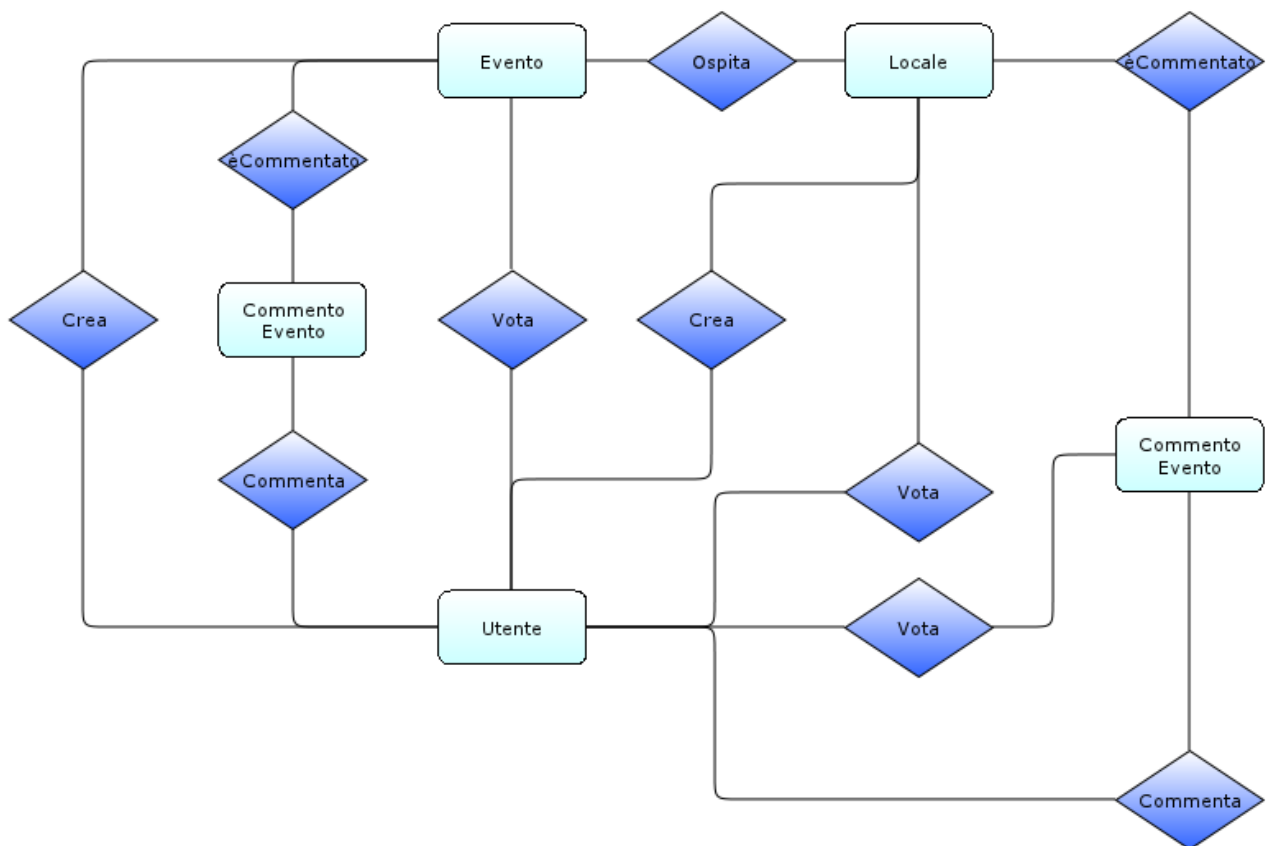
*Figura 2: Modello base del database*

Con le entità appena delineate è possibile creare un'applicazione che permette di condividere eventi, ma a livello molto base. Mancano tutte le capacità di interazione dei visitatori con gli eventi. A questo scopo bisogna quindi aggiungere delle relazioni di voto tra l'entità *Utente* e le entità *Evento* e *Locale*. Inoltre, bisogna creare le seguenti due nuove entità:

- Commento Evento – questa entità rappresenta il commento fatto da un *Utente* ad un *Evento*

- **Commento Locale** – questa entità rappresenta il commento fatto da un *Utente* ad un *Locale*

In fine, è possibile aggiungere due ulteriori relazioni di voto da parte dell'entità *Utente* nei confronti delle entità *Commento Evento* e *Commento Locale*. In questo modo, gli utenti non solo potranno votare gli eventi ed i locali interessanti, ma pure le proprie interazioni con questi, in maniera tale da neutralizzare il problema dei commenti spam oppure irrispettosi. Una piaga ben famosa dei siti di internet odierni.



*Figura 3: sono state aggiunte le relazioni ed entità che permettono l'interazione dei visitatori con i contenuti dei siti.*

Come abbiamo detto ora il modello del database permette di condividere eventi ed interagire con essi. A questo punto per aggiungere alcune delle funzionalità richieste dagli utenti è necessario introdurre due nuove entità, queste sono:

- **Etichetta** – questa entità rappresenta una caratterizzazione degli eventi. Viene creata da un *Utente* e può essere applicata a diversi eventi, così come ogni evento può avere svariate etichette

- Foto – questa entità rappresenta una foto scattata da un *Utente* ad un *Evento*. Questa entità ha due relazioni con l'entità *Utente*, la prima è il fatto di essere stata scattata e caricata da esso, la seconda può essere la presenza di un secondo *Utente* nella foto stessa.



Figura 4: Foto caricata da Mark Zuckerberg su Facebook dove è presente Barack Obama

In fine è stata aggiunta una relazione ricorsiva sull'*Utente*, questa è una relazione di amicizia, verrà utilizzata per tenere traccia di quali utenti sono amici tra loro e quindi informarli quando degli amici apprezzano gli stessi eventi. Così facendo potranno organizzarsi per andare insieme all'evento oppure incontrarsi là.

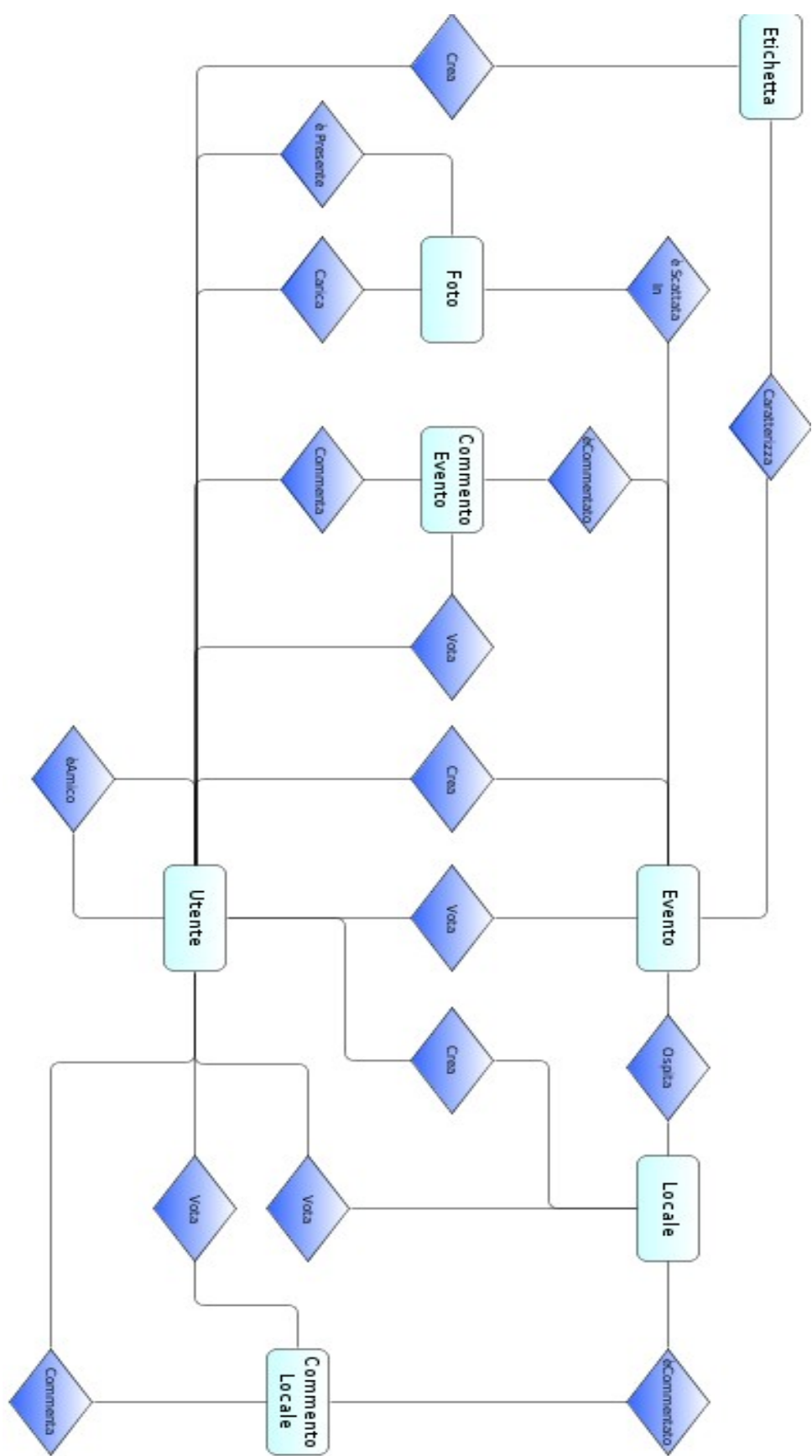


Figura 5



Per finire si possono aggiungere delle entità di supporto che non rappresentano direttamente delle funzionalità richieste dagli utenti intervistati, queste permettono però di tenere traccia ed evidenziare delle azioni compiute dagli utenti.

In questo modo è possibile premiarli ed incentivare l'utilizzo di InformaMi inserendo degli elementi di *gamificazione*. Inoltre, queste entità faciliteranno l'analisi dei comportamenti degli utenti (es. le funzionalità più utilizzate) in modo poi da poter migliorarlo adattandolo alle loro necessità.

Le due entità aggiunte sono:

- Tipo Medaglia – indica un tipo di medaglia che potrà essere assegnata agli utenti quando compiono delle attività degne di nota (es. condividere un evento votato positivamente da 100 altri utenti).
- Tipo Azione – indica un tipo di azione compiuta dagli utenti che può essere evidenziata, ma che non è abbastanza rara da meritare una medaglia. In combinazione con la relazione di amicizia introdotta al passo precedente permetterà di informare gli utenti delle azioni compiute dai propri amici.

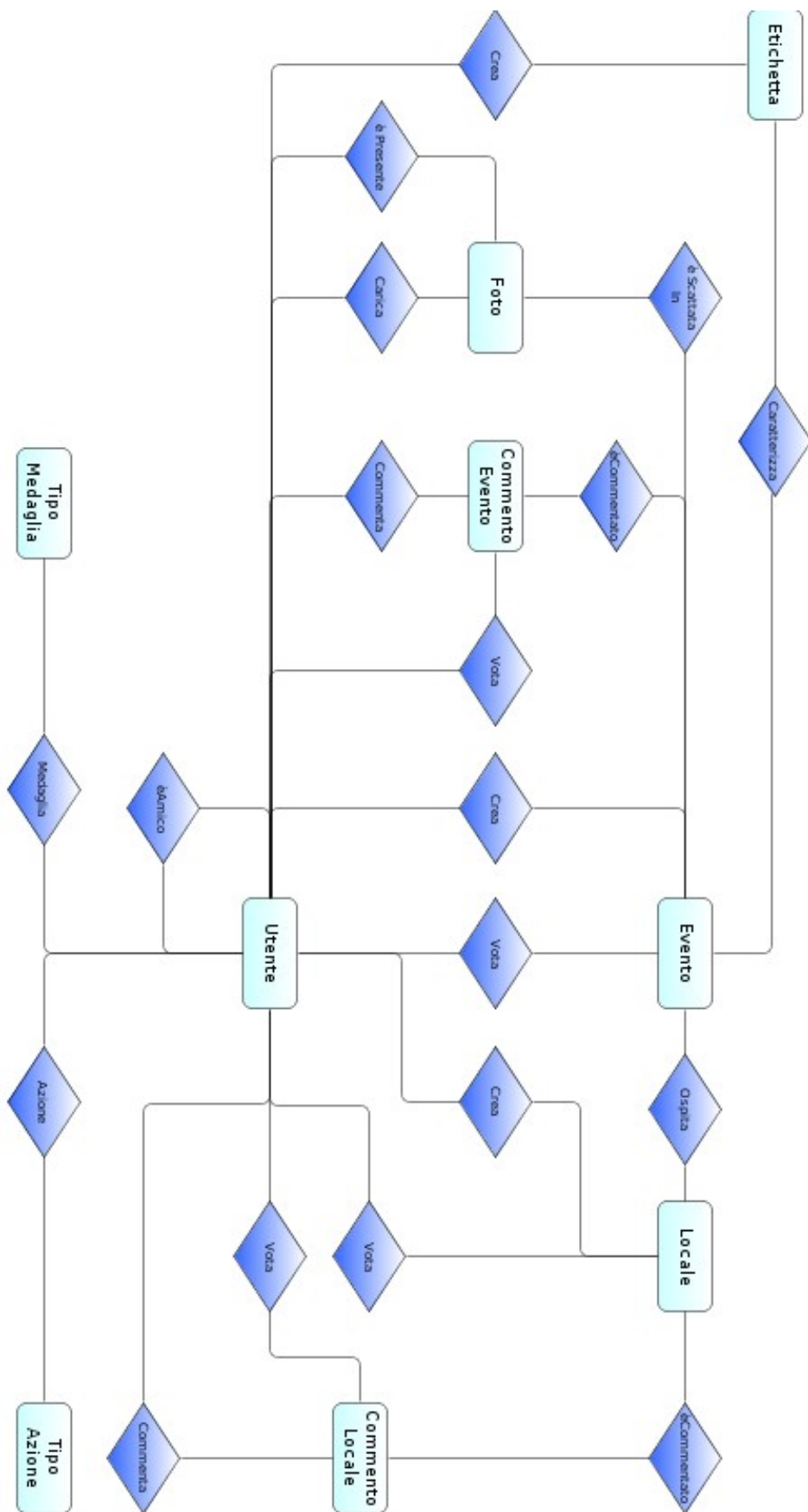


Figura 6: mostra il modello del database con riportate tutte le entità e relazioni necessarie alla realizzazione dell'applicazione web

### 4.1.2 Analisi delle entità

Di seguito verranno analizzate singolarmente tutte le entità e verranno evidenziati i campi che esse devono avere per poter memorizzare correttamente le informazioni necessarie nel database.

<b>Utente</b>	
ID	Identifica univocamente un <i>Utente</i> nel sistema
Nome	È il nome dell' <i>Utente</i> viene utilizzato come sostituto all'ID nell'interfaccia di InformaMi per rappresentare gli utenti.
Service	Una stringa di caratteri che indica quale applicativo l' <i>Utente</i> ha utilizzato per accedere al sistema.
WebId	L'ID dell'utente nell'applicativo esterno utilizzato per l'autenticazione. È utile per richiedere le sue informazioni al sistema (es. foto del profilo)
Email	Unica forma di contatto memorizzata nel sistema. Si è deciso di acquisire questo dato come salvaguardia in caso il <i>Service</i> per qualche motivo risulti non più disponibile. In questo modo sarà comunque possibile contattare gli utenti registrati ed in caso aiutarli a trasferire il loro account su un <i>Service</i> diverso.

<b>Evento</b>	
ID	Identifica univocamente un <i>Evento</i> nel sistema
Nome	È il titolo dell'evento che verrà visualizzato nella lista dei vari eventi disponibile o nella pagina dell'evento
Descrizione	È una descrizione dell'evento in cui verrà spiegato meglio di cosa si tratta. Può contenere informazioni su il genere musicale, ospiti speciali, il costo di ingresso...
Data Inizio	Questo campo memorizza la data e l'ora di inizio dell'evento
Data Fine	Questo campo memorizza la data e l'ora di fine dell'evento
Image Links	Memorizza gli URL relativi all'immagine caricata durante la creazione dell'evento

<b>Locale</b>	
ID	Identifica univocamente un <i>Locale</i> nel sistema
Nome	È il nome del locale che verrà visualizzato nella lista dei vari locali o nella pagina dedicata ad esso
Descrizione	È una descrizione del locale. Può contenere informazioni utili come una breve presentazione e gli orari di apertura
Indirizzo	È l'indirizzo del locale. Viene utilizzato per la ricerca di eventi per zona e per visualizzare le indicazioni su come arrivarci.
Image Links	Memorizza gli URL relativi all'immagine caricata durante la creazione del locale
NomePiccolo	È il nome del locale scritto interamente in minuscolo. Necessario per permettere l'auto-completamento quando si crea un nuovo evento.
Latitudine	Coordinata latitudinale del locale
Longitudine	Coordinata longitudinale del locale

<b>Commento Evento</b>	
ID	Identifica univocamente un <i>Commento Evento</i> nel sistema
Testo	Il testo del commento effettuato
Data	Data e ora di quando è stato effettuato il commento

<b>Commento Locale</b>	
ID	Identifica univocamente un <i>Commento Locale</i> nel sistema
Testo	Il testo del commento effettuato
Data	Data e ora di quando è stato effettuato il commento

<b>Etichetta</b>	
ID	Identifica univocamente un <i>Etichetta</i> nel sistema
Nome	Il nome dell'etichetta che caratterizzerà gli eventi

<b>Foto</b>	
ID	Identifica univocamente una <i>Foto</i> nel sistema
Image Links	Memorizza gli URL relativi all'immagine caricata durante la creazione dell'evento
Data	Data e ora di quando è stata caricata la foto

Tipo Medaglia	
ID	Identifica univocamente un <i>Tipo Medaglia</i> nel sistema
Nome	Il nome della medaglia
Valore	Il valore del <i>Tipo Medaglia</i>

Tipo Azione	
ID	Identifica univocamente un <i>Tipo Azione</i> nel sistema
Nome	Il nome dell'azione

### 4.1.3 Analisi delle relazioni e delle cardinalità

Di seguito vengono analizzate in dettaglio le relazioni del modello del database, specificando se necessario i campi che queste dovranno avere e la cardinalità delle relazioni.

#### **Crea**

È una relazione presente tra: *Utente e Locale*; *Utente ed Evento*; *Utente ed Etichetta*;

Rappresenta appunto il fatto che sia stato un determinato utente ad aver aggiunto al sistema un determinato locale (o evento, o etichetta). La cardinalità è uno a molti, in quanto un utente può creare più locali (o eventi, o etichette), ma un locale (o evento o etichetta) può essere creato da solo un utente.

#### **Ospita**

È una relazione tra *Locale ed Evento*.

Rappresenta il fatto che l'evento viene tenuto in un locale. La cardinalità è uno a molti.

#### **È Amico**

È una relazione ricorsiva sull'entità *Utente*.

Rappresenta il legame tra due utenti diversi, e quindi la voglia di essere informati sulle rispettive azioni compiute. La cardinalità è molti a molti, infatti, un utente può avere molti amici ed a sua volta molti utenti possono avere lui come amico.

La relazione *È Amico* non necessariamente deve essere reciproca. Ad esempio, Mario può essere amico di Luca, ma Luca non essere amico di Mario. In questo caso se Luca pubblica un evento, Mario ne verrà informato, ma se Mario pubblica un evento Luca non ne verrà informato.

### **Caratterizza**

È una relazione tra *Etichetta* ed *Evento*.

Indica che una certa etichetta rappresenta correttamente una caratteristica di un evento (es. etichetta “Musica” per un concerto). La cardinalità è molti a molti.

### **È Scattata in**

È una relazione tra *Foto* ed *Evento*.

Indica che quella particolare foto è stata scattata ad un evento, quindi può comparire nella pagina di quell'evento ed anche nella pagina del locale che ha ospitato quell'evento. La cardinalità è una a molti.

### **Carica**

È una relazione tra *Utente* e *Foto*.

Rappresenta il fatto che un determinato utente ha aggiunto la foto in questione ad InformaMi. La relazione è molti ad uno.

### **È Presente**

È una relazione tra *Utente* e *Foto*.

Indica il fatto che un utente è presente in una foto. Per questo motivo questa relazione ha due campi: *posizione X* e *posizione Y*. Questi rappresentano le coordinate del punto della foto in cui è presente l'utente. La cardinalità è molti a molti.

### **Commenta**

È una relazione tra: *Utente* e *Commento Evento*; *Utente* e *Commento Locale*.

Indica che un commento è stato creato da un utente. La cardinalità è molti a uno.

### **È Commentato**

È una relazione tra: *Commento Evento* ed *Evento*; *Commento Locale* e *Locale*.

Lega un commento all'entità commentata. La cardinalità è uno a molti.

## **Vota**

È una relazione tra: *Utente* ed *Evento*; *Utente* e *Locale*; *Utente* e *Commento Evento*; *Utente* e *Commento Locale*;

Indica il giudizio tra un utente e l'elemento votato. Questo può essere positivo o negativo, quindi necessita di un campo booleano per memorizzare questo fatto. La cardinalità è molti a molti, ma può esistere solo un unico voto tra un utente ed un evento (o locale, o commento evento, o commento locale).

## **Medaglia**

È una relazione tra *Utente* e *Tipo Medaglia*.

Rappresenta il fatto che un determinato utente ha fatto un'azione degna di nota. Necessita di un campo per memorizzare la data in cui è stata conseguita questa medaglia. La cardinalità è molti a molti.

## **Azione**

È una relazione tra *Utente* e *Tipo Azione*.

Questa relazione memorizza delle azioni importanti compiute dagli utenti, ma non abbastanza per essere assegnate delle medaglie. In alcuni casi l'azione può riguardare un oggetto del sistema e quindi è necessario un campo per associare l'azione all'oggetto. Si è deciso di utilizzare l'URL dell'oggetto come identificativo. Un altro campo necessario è quello per memorizzare la data e l'ora in cui è stata compiuta l'azione.





#### 4.1.4 Progettazione fisica del database

Prima di procedere con la progettazione del database bisogna evidenziare alcune limitazioni importanti della piattaforma che abbiamo deciso di utilizzare. Per prima cosa Google App Engine non permette l'utilizzo di un motore database relazionale (MS SQL, MySQL...), in secondo luogo non permette l'accesso in scrittura al filesystem. Per questo motivo non è possibile caricare direttamente sul server le immagini degli eventi/locali/utenti.

Al contrario dei database relazionali classici questo sarà implementato ad oggetti. Ciò vuol dire che per ogni entità si creerà una classe Java, i campi saranno delle variabili ed ogni classe può inoltre avere delle funzioni integrate in essa, questo è molto utile se si vogliono aggiungere dei trigger alle azioni di inserimento, cancellazione o aggiornamento delle entità.

Il database verrà implementato utilizzando Siena, che è una libreria che si frappone tra l'applicativo ed il database sottostante, questo allo scopo di rendere l'applicativo il più possibile indipendente dal motore database in uso. L'obiettivo di Siena è quello di permettere, se necessario, di sostituire in futuro il motore database con uno diverso richiedendo il più basso possibile numero di modifiche. È particolarmente interessante il fatto che Siena permetta di utilizzare sia motori database NoSQL (come l'attuale Datastore di Google) che motori SQL (come MySQL o MS SQL), questo aspetto risulterà fondamentale se si deciderà di trasferire InformaMi su un servizio di hosting diverso da Google App Engine.

Una differenza importante tra l'implementazione tramite oggetti e tramite tabelle è che, nel primo caso, le relazioni tra due entità non verranno realizzate mediante l'uso delle chiavi esterne, invece, verrà incluso l'oggetto stesso che si vuole referenziare nell'entità.

Per semplicità si può vedere ad esempio la relazione *è scattata in* tra *Evento* e *Foto*. In un database relazionale si sarebbero create due tabelle diverse, entrambe con un *id* identificativo. Inoltre, alla tabella *Foto* si sarebbe aggiunto un campo *idEvento* in cui includere l'*id* dell'*Evento* referenziato. In questo modo è possibile ottenere tutte le foto collegate ad un evento effettuando una ricerca sulla tabella *Foto* per quelle con un particolare *idEvento*, d'altra parte è possibile risalire all'evento dove è stata scattata una foto ricercando sulla tabella *Evento* quella riga il cui *id* è pari all'*idEvento* della foto in questione.

Nel database ad oggetti implementato con Siena invece di creare delle tabelle verranno create delle classi Java. Nel caso dell'esempio in esame l'entità *Evento* avrà una variabile di tipo `Query<Foto>`, questa restituisce tutte le *Foto* che referenziano l'evento in questione. Per referenziare un *Evento* la classe *Foto* non deve fare altro che avere una variabile di tipo *Evento* ed inizializzarlo all'evento che si vuole referenziare. In questo modo se a partire da

una foto si vuole risalire all'evento dove è stata scattata non è necessario fare un'ulteriore query, questo è già presente all'interno dell'entità *Foto*.

Questo è ovviamente una semplificazione concettuale per il programmatore, l'implementazione effettiva usa delle query automatiche su degli id memorizzati nel database. Se si volesse però fare a meno di questa semplificazione, ed utilizzare il database più come un classico motore relazionale lo si potrebbe fare. Basterebbe memorizzare delle liste con gli id degli oggetti che referenziano e dall'altro lato memorizzare l'id dell'entità da referenziare. Questo porterebbe però a dover garantire programmaticamente l'integrità referenziale di tutte le entità. Cioè garantire che in nessuna lista ci sia l'id di un'entità che non esiste. I database relazionali permettono di avere questa garanzia facilmente una volta fornite delle regole. Purtroppo il motore database messo a disposizione di Google App Engine non fornisce questa possibilità.

Dato quindi il funzionamento del motore database, o meglio il suo funzionamento attraverso la libreria Siena, il passaggio dal modello Entità-Relazione del capitolo precedente, ad un modello fisico del database è stato molto veloce. Le entità sono state trasformate in classi ed ad esse sono stati aggiunte delle variabili di tipo *Entità* o di tipo *Query* per implementare le relazioni. Le relazioni che richiedevano dei campi aggiuntivi (ad esempio *èPresente*) sono anch'esse state trasformate in classi ed i due estremi delle relazioni sono stati rappresentati mediante delle variabili di tipo *Entità*.

Si può vedere nel grafico seguente il modello fisico finale del sistema dove ogni riquadro rappresenta una classe, con le varie variabili, ed i collegamenti tra loro indicano le relazioni tra i campi *Query* ed *Entità*.

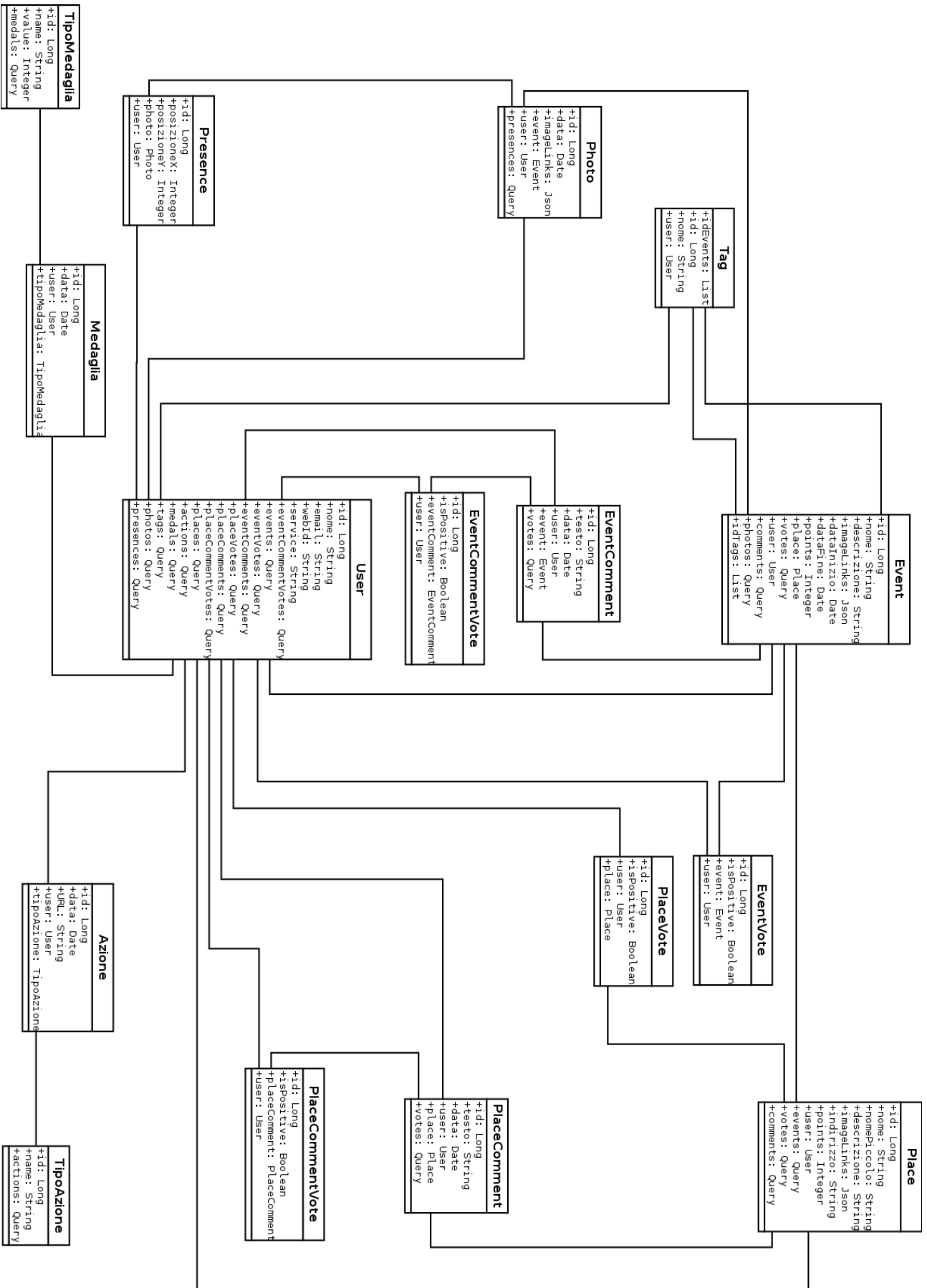


Figura 8: modello fisico del database

## 4.2 Progettazione moduli funzionali

Come detto in precedenza i moduli funzionali verranno divisi in diversi *controllers* che si occuperanno di fare da tramite tra il modello del database e l'interfaccia utente. Questi moduli funzionali sono:

- Application
- Events
- Places
- Tags
- Photos

Il primo *controller* gestisce funzioni generiche, come l'autenticazione utente e la ricerca, gli altri gestiscono funzioni legate ai modelli dai quali prendono il nome.

Vi saranno due moduli funzionali che non possono essere implementati come *controllers*, invece saranno inseriti direttamente all'interno dei modelli ai quali appartengono essendo essi dei *trigger* da attivare ogni volta che viene creata un entità. Questi due *trigger* sono:

- Actions
- Medals

## 4.2.1 Application

Deve gestire alcune funzionalità generiche, queste sono:

- Login utente ed eventuale registrazione automatica

Carica i dati dell'utente, che intende autenticarsi, attraverso il client di Facebook fornito da *RestFB*, successivamente cerca nel database se è presente un utente con quell'indirizzo email. Se è presente carica l'id dell'utente ed inserisce questa informazione nei dati della sessione. Se non è presente ne crea uno e lo inserisce nel database, successivamente inserisce l'id dell'utente appena creato nei dati della sessione.

- Logout utente

Rimuove dai dati della sessione l'id dell'utente.

- Autenticazione tramite Facebook

Questa funzione viene chiamata dall'API di Facebook durante l'autenticazione dell'utente, si aspetta come parametro un codice di autenticazione. Con questo codice l'applicazione esegue una richiesta all'API di Facebook per ottenere un codice di autorizzazione. Questo se ottiene una risposta da Facebook che contiene il codice di autorizzazione aggiunge questo dato alla sessione, chiama la funzione di login ed in fine visualizza la homepage.

- Ricerca nel sito

Questa funzione è molto semplice, si occupa semplicemente di visualizzare una pagina passando ad essa i parametri di ricerca. La pagina in questione contiene un codice Javascript che contatta i server di Google, carica e visualizza i risultati della ricerca.

## 4.2.2 Events

Deve gestire le funzionalità degli eventi, queste sono:

- Visualizzazione lista eventi

Carica tutti gli eventi non ancora scaduti ed ordinati per popolarità. Successivamente li visualizza. Se questa funzione riceve come parametro un indirizzo, prima di caricare gli eventi otterrà le coordinate geografiche dell'indirizzo, quindi caricherà solo gli eventi non scaduti che sono vicini a quell'indirizzo.

- Visualizzazione singolo evento

Carica tutte le informazioni riguardanti un singolo evento (locale, etichette, utente che ha creato l'evento, commenti...) poi lo visualizza.

- Creazione evento

Riceve i dati che l'utente ha voluto inserire sull'evento. Converte i dati nel formato che il modello si aspetta (ad esempio la data inizio e fine in *java.util.Date*), crea le etichette che non esistono ancora nel database, carica l'immagine dell'utente sul servizio di hosting e prepara il JSON con gli URL che la riguardano. In fine usa tutti questi dati per creare un nuovo evento e lo inserisce nel database.

- Aggiunta di un voto all'evento

Verifica che l'utente in questione non abbia già votato quell'evento. Se non l'aveva votato crea un nuovo voto e lo inserisce nel database. Se l'aveva votato ed il giudizio era lo stesso che in questo caso, ignora questo nuovo voto. Se l'aveva votato, ma il giudizio è opposto rispetto quello nel database, elimina il voto precedente, in questo modo risulta come se l'utente non avesse mai votato questo evento.

- Aggiunta di un commento all'evento

Semplicemente inserisce un nuovo commento per quell'evento da parte dell'utente.

### 4.2.3 Places

Deve gestire le funzionalità dei locali. Queste sono molto simili a quelle degli eventi, le due funzionalità che si differenziano sono:

- Creazione locale

Riceve i dati che l'utente ha voluto inserire sul locale. Converte i dati nel formato che il modello si aspetta (ad esempio la data inizio e fine in `java.util.Date`), carica l'immagine dell'utente sul servizio di hosting e prepara il JSON con gli URL che la riguardano, ottiene le coordinate geografiche corrispondenti all'indirizzo inserito dall'utente. In fine crea un nuovo locale e lo inserisce nel database.

- Elenco locali filtrati per nome

Carica i nomi e l'indirizzo di tutti i locali il cui nome comincia per una determinata sequenza di caratteri, poi restituisce questi dati sotto forma di JSON. Questa funzione è utilizzata dalla pagina per la creazione di eventi per l'auto-completamento del nome del locale.

### 4.2.4 Tags

Deve gestire le funzionalità delle etichette

- Elenco eventi per etichetta

Carica tutti gli eventi che sono stati classificati con una determinata etichetta ordinati per popolarità. Successivamente li visualizza. Se questa funzione riceve come parametro un indirizzo, prima di caricare gli eventi otterrà le coordinate geografiche dell'indirizzo, quindi caricherà solo gli eventi che sono vicini a quell'indirizzo

- Elenco etichette filtrate per nome

Carica i nomi di tutte le etichette il cui nome comincia per una determinata sequenza di caratteri, poi restituisce questi dati sotto forma di JSON. Questa funzione è utilizzata dalla pagina per la creazione di eventi per l'auto-completamento delle etichette.

### 4.2.5 Photos

Deve gestire le funzionalità delle foto

- Aggiungi foto ad evento

Ottiene come parametri un'immagine e l'evento al quale legarla. Carica l'immagine al servizio di hosting, prepara il JSON con gli URL dell'immagine e crea l'entità nel database.

- Visualizza foto evento

Carica l'elenco di tutte le foto legate ad un evento nell'ordine in cui sono state aggiunte al sistema. Successivamente visualizza una pagina con le miniature di queste immagini, dove è possibile visualizzarle in grande.

- Aggiungi presenza a foto

Riceve come parametri l'id di una foto, le coordinate  $x$  ed  $y$  di dove l'utente ha cliccato nella foto. Crea una presenza con questi dati e la inserisce nel database.

### 4.2.6 Actions

Sovrascrivendo la funzione *insert()* dei modelli è facile eseguire delle operazioni ogni volta che viene aggiunta un'entità al sistema. In questo modo è possibile creare una serie di *Azioni* per tenere traccia del comportamento degli utenti.

Alcune azioni che possono essere registrate sono: creazione evento; commento evento; creazione locale; commento locale; caricamento foto; creazione etichetta...

Il motivo per tenere traccia di queste azioni ed il tipo di azione da registrare verranno analizzati più nel dettaglio nel capitolo riguardante la *gamificazione*.

### 4.2.7 Medals

Allo stesso modo che nel punto precedente, sovrascrivendo la funzione *insert()* del modello delle azioni è possibile intercettare la creazione di un'azione. In questo modo è possibile verificare se questa azione comporta l'assegnazione di una medaglia ad un utente. Per fare ciò bisognerà analizzare le possibili situazioni che possono comportare l'assegnazione di una medaglia, scartare quelle già assegnate, verificare se le rimanenti situazioni sono verificate, in caso positivo creare una medaglia per l'utente ed inserirla nel database. In fine, notificare l'utente delle nuove medaglie ottenute. Le azioni che comporteranno l'assegnazione di medaglie verranno analizzate nel capitolo dedicato alla *gamificazione*.



## 4.3 Progettazione ed implementazione dell'interfaccia grafica

### 4.3.1 Delineazione degli elementi di base

Per la progettazione dell'interfaccia grafica si è proceduto a passi. Per prima cosa si sono realizzati dei mockup di alcune schermate fondamentali. In questo modo è stato possibile definire un'idea di base di come dovrà essere l'interfaccia utente.

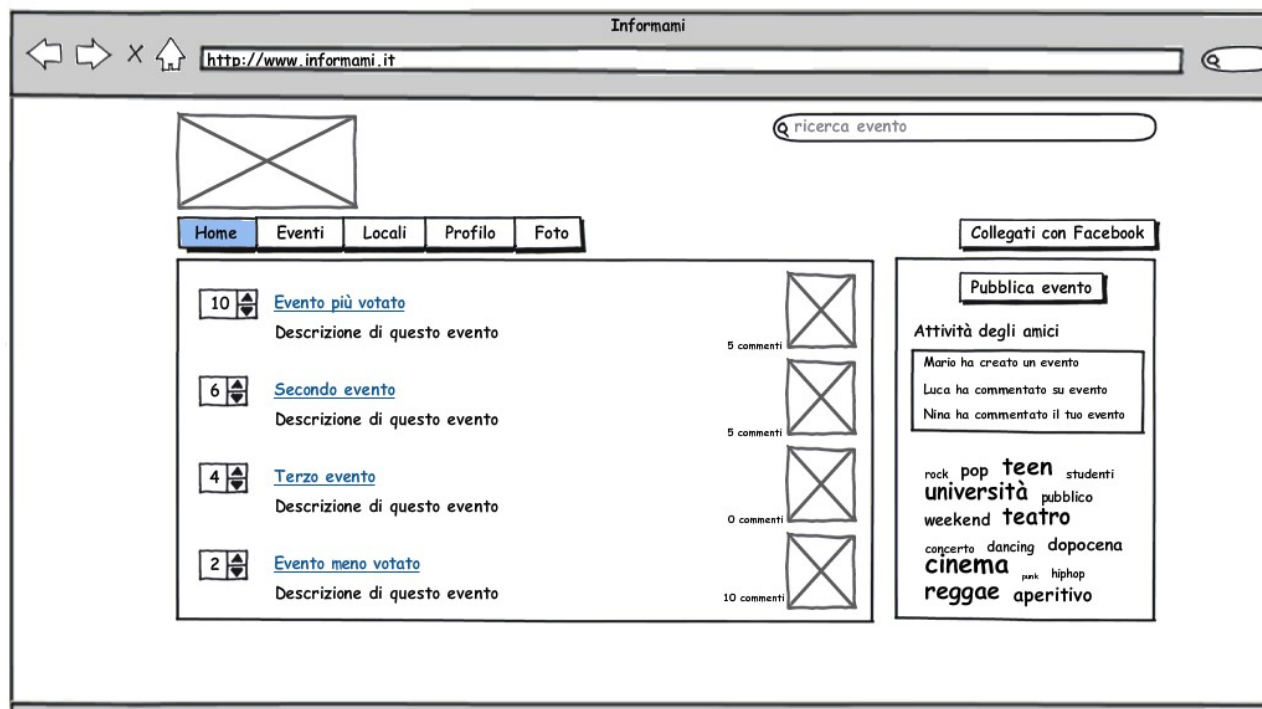


Figura 9: mockup homepage

In figura 9 sono presenti tutti gli elementi fondamentali del sistema. In alto è presente il logo dell'applicazione, i link di navigazione, un campo per la ricerca ed un bottone per autenticarsi tramite Facebook. La pagina è poi divisa in due colonne:

- nella colonna sinistra vi è l'elenco degli eventi ordinati per popolarità. Questi hanno un titolo, una breve descrizione e l'immagine dell'evento.
- Nella colonna destra ci sono informazioni riguardo le azioni degli amici, una *tag cloud* con le etichette più popolari per gli eventi ed un link per pubblicare un nuovo evento

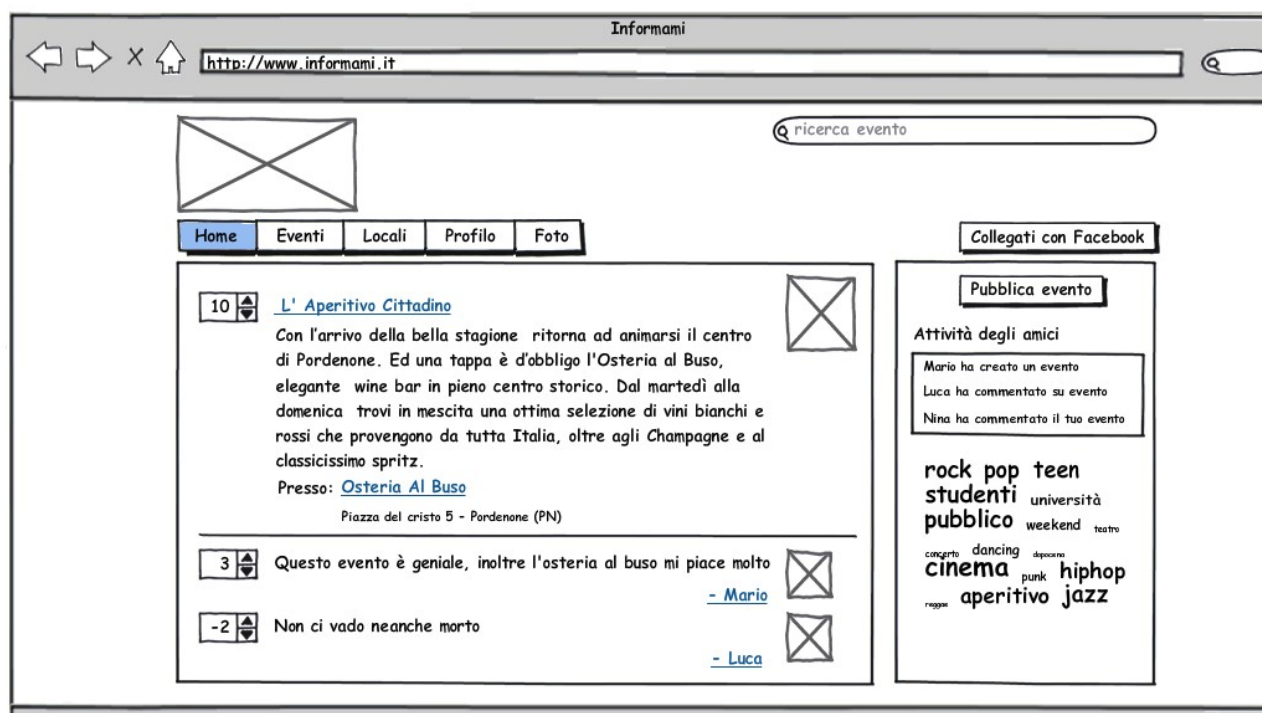


Figura 10: mockup della schermata di un evento

Solo la colonna sinistra è cambiata rispetto alla figura precedente. Qui sono presenti alcuni elementi già visti (il punteggio, le frecce per votare, il titolo e l'immagine) è inoltre presente la descrizione completa, un link al locale che ospita questo evento (con indirizzo) ed i commenti degli altri utenti. Come si può vedere anche i commenti hanno un punteggio e sono ordinati in base ad esso, inoltre, è presente la foto ed il nome dell'autore del commento.

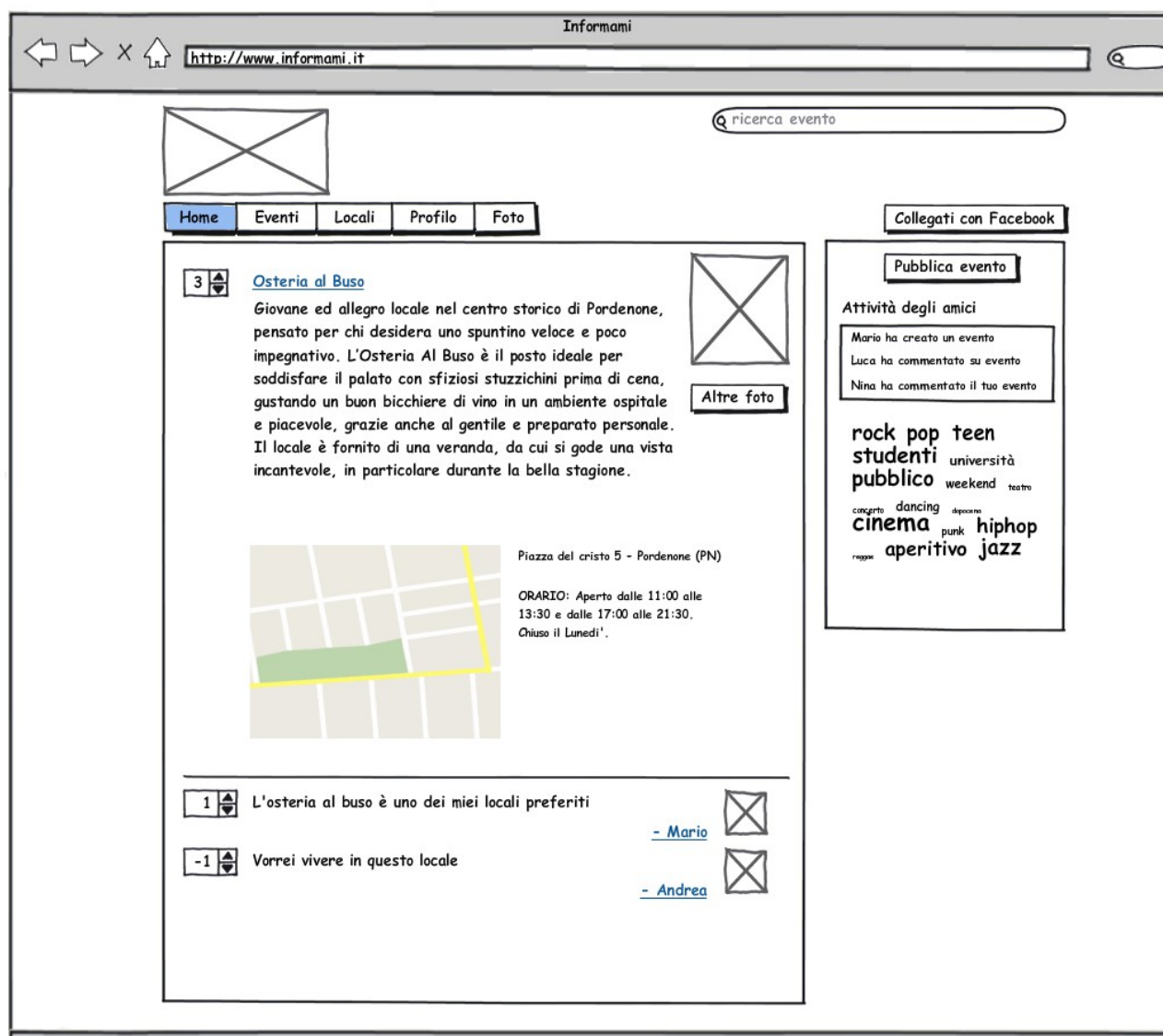


Figura 11: rappresentata il mockup di una schermata di un locale

Questa è molto simile al mockup di un evento, la principale differenza è la presenza di una mappa per meglio rappresentare l'indirizzo del locale.

Queste schermate sono poi state presentate ad alcuni studenti universitari. La reazione è stata molto positiva, tutti sono stati in grado di intuire l'utilizzo o significato degli elementi presenti nel sito (una volta spiegato a grandi linee lo scopo di InformaMi). Benché non sia comune ideare bene l'interfaccia grafica al primo tentativo, è probabile che questo successo sia dovuto al fatto di aver riutilizzato elementi presenti in diversi siti comunemente utilizzati dal target di utenti scelto.

### **4.3.2 Implementazione interfaccia grafica**

Basandosi sui mockup realizzati al punto precedente si è proceduto ad implementare l'interfaccia grafica. Per fare ciò si è deciso di partire da un tema sviluppato da *digitalnature* ed in linea con lo stile che si voleva dare al sito. Il tema si chiama *Mystique*.

Questo poi è stato modificato per adattarsi meglio alle necessità del progetto. Si può vedere nelle seguenti schermate il risultato finale.



## GLI SGAMI DI DOUBLE G

6 DAYS AGO Posted by [Maurizio Pozzobon](#)

1

DOUBLE G torna ad infuocare i giovedì sera al ROUND MIDNIGHT! Un giro di 360 gradi nel mondo della m...

[» Leggi tutto](#)



## WHITE WINDOW

6 DAYS AGO Posted by [Maurizio Pozzobon](#)

1

I White Widdow sono un gruppo australiano molto giovane dal punto di vista di anni di attività, sono...

[» Leggi tutto](#)



## TURBONEGRA + LOCAL HEROES

6 DAYS AGO Posted by [Maurizio Pozzobon](#)

0

Cosa succede quando un wild bunch di debosciate ragazze californiane dedite a sesso droga e rock and...

[» Leggi tutto](#)



## RED PASSION: IL GIOVEDI DEL REX

6 DAYS AGO Posted by [Maurizio Pozzobon](#)

0

L'aperitivo lungo del Rex con gli apprezzati cocktails di Manuel e le selezioni musicali di Paolo Val...

[» Leggi tutto](#)



## Evento prova

1 MONTH AGO Posted by [Maurizio Pozzobon](#)

-1

Questo è un evento di prova

[» Leggi tutto](#)



## Secondo evento di prova

1 MONTH AGO Posted by [Maurizio Pozzobon](#)

-1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec volutpat tincidunt lectus, commodo po...

[» Leggi tutto](#)

Search

GO

### MAURIZIO POZZOBON

Crea evento

Crea locale

### VICINO A...

Cerca

### TAG CLOUD

[aperitivo](#) [ballare](#)

[concerti](#)

[coverband](#) [dopocena](#)

[eatdrink](#)

[musicadalvivo](#) [prova](#)

[rock](#)

Mystique theme by [digitalnature](#)



RSS FEEDS

XHTML 1.1

TOP





## GLI SGAMI DI DOUBLE G

6 DAYS AGO

Posted by [Maurizio Pozzobon](#)

1

DOUBLE G torna ad infuocare i giovedì sera al ROUND MIDNIGHT! Un giro di 360 gradi nel mondo della musica e delle "good vibrations", passando dal ROCK al FUNK, per arrivare all'HIP HOP ed al resto della Black Music, per poi finire con l'elettronica più underground, come RAGGA JUNGLE e DUBSTEP.



[Altre foto](#)

### Indirizzo

Via Ginnastica 39 - Trieste (ts)



[ballare dopocena](#)

Search

GO

### MAURIZIO POZZOBON

[Crea evento](#)

[Crea locale](#)

### VICINO A...

Cerca

### TAG CLOUD

[aperitivo](#) [ballare](#)

[concerti](#)

[coverband](#) [dopocena](#)

[eatdrink](#)

[musicadalvivo](#) [prova](#)

[rock](#)

### COMMENTS (0)

[Aggiungi commento](#)

Mystique theme by [digitalnature](#)



RSS FEEDS

XHTML 1.1

TOP



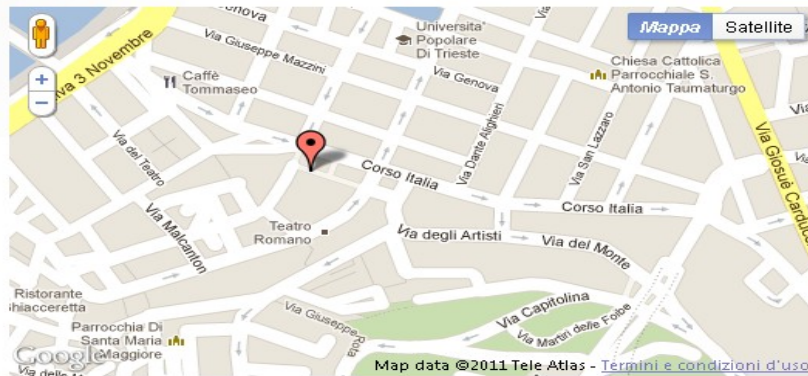
## Rex Café Gourmet

Posted by **Maurizio Pozzobon**

0

Il Rex Café Gourmet si distingue per lo stile moderno ed elegante: è un luogo ideale per le colazioni o come sala da the, ma diventa anche raffinato ristorante con una cucina dalle sfumature regionali. Il menu, appena rinnovato, vanta una vasta scelta di carpacci e tartare di carne e pesce, primi piatti e snack. All'ora dell'aperitivo si trasforma in meta mondana con gustosi cocktail, pestati e stuzzichini. Il Rex ha la vocazione del wine bar: dalla sua elegantissima cantina sotto vetro sfodera vini prestigiosi (italiani e francesi al centro dell'attenzione), una fornita rhumeria e degustazioni a tema, anche di champagne e Franciacorta. Al Rex l'offerta gastronomica si sposa con quella culturale: il locale organizza concerti, mostre fotografiche, serate live e originalissimi party. Il locale gode anche di un piccolo dehors, con sgabelli e divanetti glamour, sotto i portici a due passi da piazza Unità d'Italia. ORARIO: dal lunedì al giovedì 7-24, venerdì e sabato 7-2 di notte, domenica e festivi 9-22.

Galleria Protti 1 - Trieste (ts)



Search

GO

### MAURIZIO POZZOBON

Crea evento

Crea locale

### VICINO A...

Cerca

### TAG CLOUD

[aperitivo](#) [ballare](#)

[concerti](#)

[coverband](#) [dopocena](#)

[eatdrink](#)

[musicadalvivo](#) [prova](#)

[rock](#)

### COMMENTI (0)

Aggiungi commento

Mystique theme by [digitalnature](#)



RSS FEEDS

XHTML 1.1

TOP

## Capitolo 5 - Implementazione moduli funzionali

*In questo capitolo viene mostrata l'implementazione finale di tre dei moduli funzionali presentati nel capitolo 4.2, verrà prima chiarito meglio la loro logica tramite dei diagrammi di flusso e poi verrà mostrato il codice sorgente finale.*

### **5.1 Gestione login Facebook**

La gestione del login degli utenti viene fatta integrandosi con l'API di Facebook il diagramma di flusso dell'interazione tra InformaMi e Facebook è il seguente.



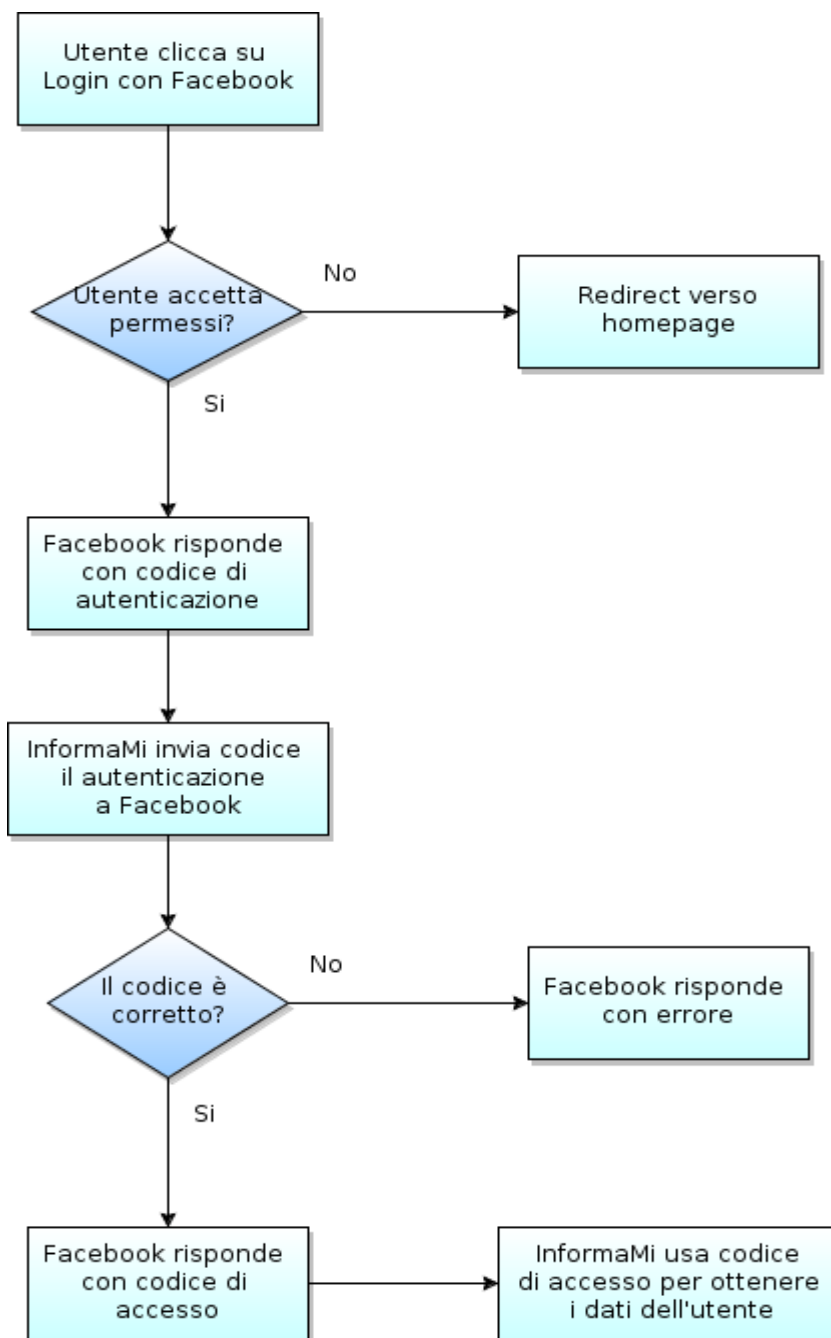


Figura 12: diagramma di flusso per la gestione del login

Questo procedimento viene implementato nel modo seguente. Per prima cosa InformaMi crea un link che se cliccato dall'utente lo porterà su Facebook. Il link deve contenere 3 dati importanti: l'id che identifica InformaMi, l'URL al quale inviare l'utente una volta che Facebook l'ha autenticato, i permessi di cui l'applicazione ha bisogno che l'utente gli conceda (in questo caso sono: l'email, data di nascita, luogo in cui si trova in questo

momento, poter pubblicare a nome dell'utente). Questo viene fatto tramite il seguente codice:

```
private static final String client_id = "202159310993";
private static final String redirect_uri = "http://www.informami.it/fbauth";

public static String getLoginRedirectURL() {
    return "https://graph.facebook.com/oauth/authorize?client_id=" +
        client_id + "&display=page&redirect_uri=" +
        redirect_uri + "&scope=publish_stream,email,user_location,user_birthday";
}
```

Dopo il click dell'utente, se ha accettato i permessi, verrà reindirizzato al URL indicato in precedenza, questo viene gestito da una funzione che deve effettuare un'ulteriore chiamata a Facebook, questa volta richiedendo un codice di autorizzazione. Se la chiamata va a buon fine, la risposta deve contenere due parametri *access\_code* ed *expires*, questi sono appunto il codice di autorizzazione necessario per richiedere i dati dell'utente, ed il numero di secondi per il quale quel codice è valido.

Una volta ottenuti questi due valori vengono memorizzati nei dati della sessione, in modo da poterli utilizzare in seguito, dopodiché viene visualizzata la homepage. Queste funzioni vengono eseguite dai seguente codici:

```
private static final String secret = "4*****4";

public static String getAuthURL(String authCode) {
    return "https://graph.facebook.com/oauth/access_token?client_id=" +
        client_id + "&redirect_uri=" +
        redirect_uri + "&client_secret="+secret+"&code="+authCode;
}

private static String readURL(URL url) throws IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    InputStream is = url.openStream();
    int r;
    while ((r = is.read()) != -1) {
        baos.write(r);
    }
    return new String(baos.toByteArray());
}
```

```

public static void fbauth(String code){
    if (code!=null) {
        String authURL = lib.Facebook.getAuthURL(code);
        try {
            URL url = new URL(authURL);
            String result = readURL(url);
            String accessToken = null;
            Integer expires = null;
            String[] pairs = result.split("&");
            for (String pair : pairs) {
                String[] kv = pair.split("=");
                if (kv.length != 2) {
                    throw new RuntimeException("Unexpected auth response");
                } else {
                    if (kv[0].equals("access_token")) {
                        accessToken = kv[1];
                    }
                    if (kv[0].equals("expires")) {
                        expires = Integer.valueOf(kv[1]);
                    }
                }
            }
            if (accessToken != null && expires != null) {
                facebookClient = new DefaultFacebookClient(accessToken);
                session.put("accessToken", accessToken);
                session.put("expires", expires+System.currentTimeMillis()/1000);
                DoLogin("Facebook");
                renderTemplate("Events/index.html");
            } else {
                Logger.error("Access token and expires not found");
                renderTemplate("Events/index.html");
            }
        } catch (IOException e) {
            Logger.error(e.toString());
            renderTemplate("Events/index.html");
        }
    }
}

```

## 5.2 *Hosting immagini*

La gestione dell'hosting delle immagine viene fatta integrandosi con l'API di Imgur, un servizio di hosting per immagini che è gratuito fino a 50 immagini all'ora. Il funzionamento di base di questa API è molto facile, è sufficiente inviare l'immagine ad un certo indirizzo e la risposta sarà una serie di URL riguardanti l'immagine. Questi sono:

- URL dell'immagine originale
- URL dell'immagine tagliata e scalata per risultare un piccolo quadrato
- URL dell'immagine scalata da usare come anteprima
- URL da chiamare per cancellare quell'immagine dai server di Imgur

Imgur si aspetta tra i dati della chiamata l'immagine da caricare codificata in *base64* e la chiave identificativa dell'applicazione. La risposta può essere in formato XML o JSON, si è deciso di utilizzare il formato JSON.

Di seguito è riportato il diagramma di flusso delle operazioni eseguite quando un utente cerca di caricare un'immagine. Questo è lo stesso sia nel caso che l'immagine venga caricata durante la creazione di un evento/locale oppure durante il caricamento di una foto.

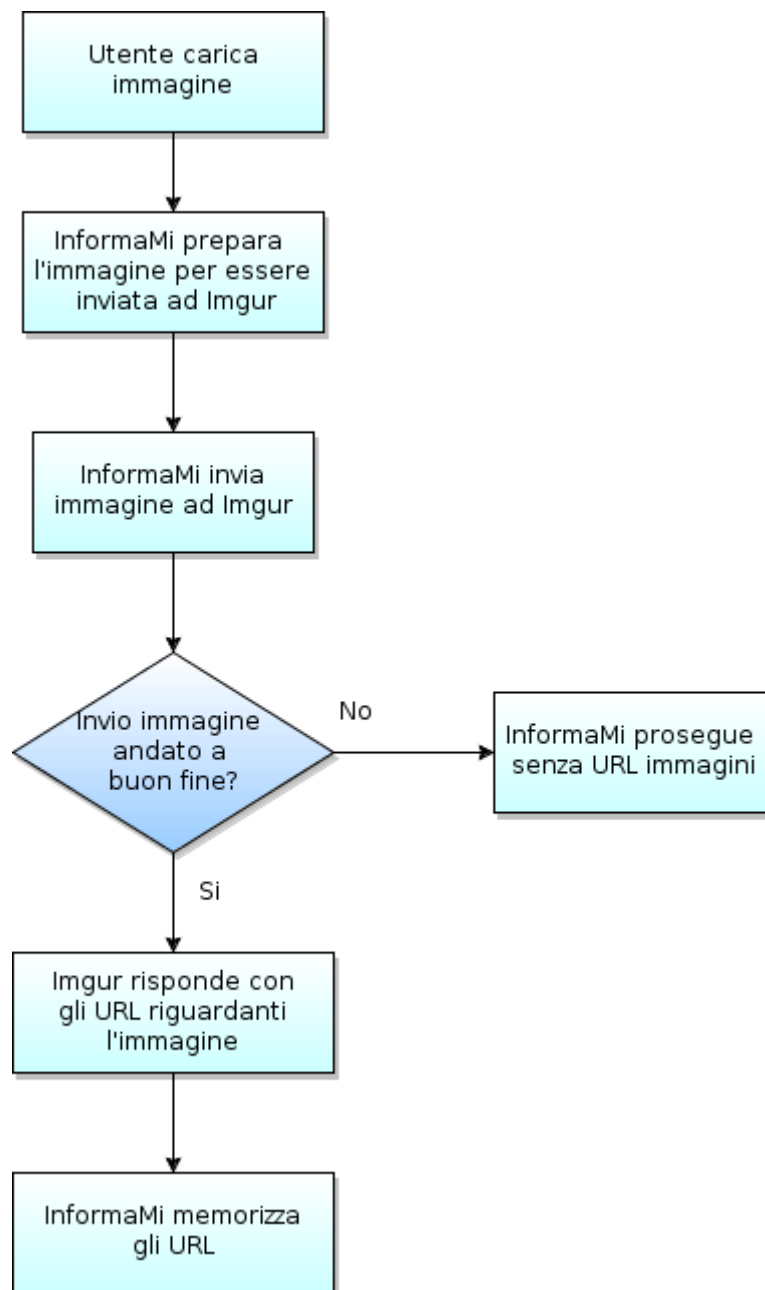


Figura 13: diagramma di flusso per il caricamento delle immagini

Di seguito è riportata la funzione che esegue le operazioni sopra descritte. Si può vedere come per prima cosa prepari i dati da inviare (quindi codifica l'immagine in *base64* ed appende la chiave di identificazione), poi invia questi dati all'URL dell'API di Imgur, successivamente legge la risposta, se questa è un JSON nel formato che si aspetta, carica gli URL altrimenti restituisce un JSON nullo.

```

private static final String imgurKey="O*****c";

public static Json uploadImage(Upload imagine) throws Exception {

    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    baos.write(imagine.asBytes());

    String data = URLEncoder.encode("image", "UTF-8") + "=" +
        URLEncoder.encode(Base64.encodeBytes(baos.toByteArray()), "UTF-8");
    data += "&" + URLEncoder.encode("key", "UTF-8") + "=" +
        URLEncoder.encode(imgurKey, "UTF-8");

    URL url = new URL("http://api.imgur.com/2/upload.json");

    URLConnection conn = url.openConnection();
    conn.setDoOutput(true);
    OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream());
    wr.write(data);
    wr.flush();

    BufferedReader br =new BufferedReader(
        new InputStreamReader(conn.getInputStream()));
    String inputLine;
    String a="";
    while ((inputLine = br.readLine()) != null)
        a=a+inputLine;
    br.close();

    Json json=null;
    try {
        JSONObject j = new JSONObject(a);
        j = j.getJSONObject("upload").getJSONObject("links");
        json = map().put("original", j.getString("original"))
            .put("small_square", j.getString("small_square"))
            .put("large_thumbnail", j.getString("large_thumbnail"))
            .put("delete_page", j.getString("delete_page"));
    } catch (Exception e) {
        e.printStackTrace();
    }
    return json;
}

```

### 5.3 Gestione ricerca per zona

Per la ricerca per zona si è deciso di valorizzare la semplicità di realizzazione rispetto alla precisione e velocità di esecuzione. La procedura per la ricerca di eventi vicino ad un indirizzo è delineata dal seguente diagramma di flusso.

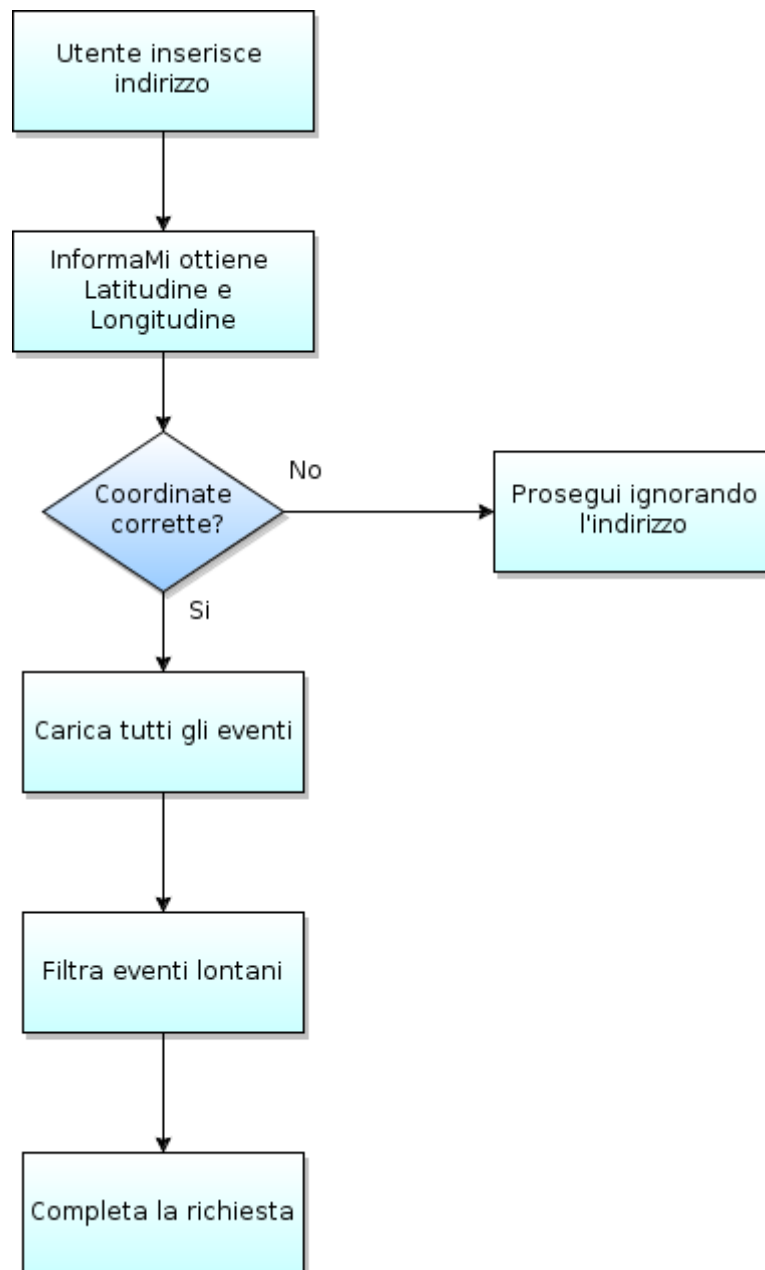


Figura 14: diagramma di flusso per la ricerca per zona

La semplificazione viene effettuata nella procedura di filtraggio degli eventi. Infatti, invece di verificare se gli eventi sono tenuti entro un raggio dall'indirizzo indicato, si utilizzerà una delimitazione rettangolare. Questo permette di filtrare gli eventi con quattro *if* che verificano che non vi siano più o meno 0,2 gradi di differenza in latitudine e longitudine tra l'indirizzo indicato e gli eventi visualizzati. 0,2 gradi corrispondono a 20 km circa latitudinalmente ed a 15 km longitudinalmente (alle nostre latitudini). Quindi all'utente verranno mostrati tutti gli eventi in un rettangolo di 40 km di base e 30 km di altezza centrato all'indirizzo indicato.

Una semplificazione dal punto di vista logico, ma che penalizza il sistema nelle prestazioni è il fatto che per filtrare gli eventi, prima si caricano tutti gli eventi, per ognuno di questi si carica il locale che li ospita, e poi si esegue il controllo sulle coordinate del locale. Se col tempo questa operazione diventerà troppo lenta si potrà duplicare le coordinate geografiche anche sull'evento, quindi effettuare il filtraggio direttamente sulla ricerca che carica gli eventi, riducendo nettamente i tempi di esecuzione.

```
public static void index(String address) {
    User user=CommonFunctions.getLoggedUser(session);
    Double lat=null,lng=null;
    if(address!=null){
        try{
            Map<String, Double> location = CommonFunctions.geocode(address);
            lat=location.get("latitude");
            lng=location.get("longitude");
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
    List<Event> events= Event.getAllByPoints(lat,lng);
    for (Event event : events) {
        event.user = User.findById(event.user.id);
    }
    List<Tag> tagcloud = Tag.getTagCloud();
    render(user,events,tagcloud,address);
}
```

Le coordinate geografiche corrispondenti ad un indirizzo vengono ottenute tramite l'API di Google, che risponde con un JSON con diverse informazioni, tra le quali appunto latitudine e longitudine dell'indirizzo cercato.



```

public static Map<String,Double> geocode(String address)
    throws IOException, JSONException{
    URL url = new URL(
        "http://maps.googleapis.com/maps/api/geocode/json?address=" +
        URLEncoder.encode(address,"UTF-8")+"&sensor=false");

    URLConnection conn = url.openConnection();

    BufferedReader br =new BufferedReader(
        new InputStreamReader(conn.getInputStream()));

    String inputLine;
    String a="";
    while ((inputLine = br.readLine()) != null)
        a=a+inputLine;
    br.close();
    JSONObject j =new JSONObject(a);
    Map<String,Double> map = new HashMap<String, Double>();

    j=j.getJSONArray("results").getJSONObject(0);
    j=j.getJSONObject("geometry");
    j=j.getJSONObject("location");
    map.put("latitude", j.getDouble("lat"));
    map.put("longitude", j.getDouble("lng"));

    return map;
}

```

La seguente funzione restituisce tutti gli eventi nel sistema, permette di filtrarli in base al fatto che siano scaduti o meno ed in base alla prossimità delle coordinate geografiche.

```

public static List<Event> getAll(boolean notExpireds,Double latitude,
    Double longitude) {
    List<Event> l;
    if(notExpireds)
        l= all().filter("dataFine>", new Date()).fetch();
    else
        l= all().fetch();
    if((latitude!=null)&&(longitude!=null)){
        List<Event> tempList = new ArrayList<Event>();
        for (Event event : l) {
            event.place=Place.findById(event.place.id);
            if(event.place.isNear(latitude,longitude))
                tempList.add(event);
        }
        l=tempList;
    }

    return l;
}

```

Come è stato detto in precedenza, il filtraggio viene effettuato su un rettangolo di 0,4 gradi per lato.

```
public boolean isNear(Double latitude2, Double longitude2) {  
    Double diff = 0.2;  
  
    if((latitude<latitude2+diff)&&(latitude>latitude2-diff)  
        &&(longitude<longitude2+diff)&&(longitude>longitude2-diff))  
        return true;  
    else  
        return false;  
}
```

## Capitolo 6 - Gamificazione

*In questo capitolo viene spiegato il concetto di gamificazione e il modo in cui viene applicato ad InformaMi.*

Gamificazione è la pratica di prendere degli spunti dal funzionamento dei giochi ed applicarli a qualcosa che non sia un gioco. L'idea alla base di ciò è che questo aumenta il coinvolgimento degli utenti con l'applicativo in questione.

I meccanismi che verranno utilizzati per aumentare il coinvolgimento degli utenti sono i seguenti:

- Riconoscimenti
  - Vengono riconosciute azioni compiute dagli utenti per invogliarli a continuare a farle, come premio verranno fornite delle medaglie di diverso valore (bronzo, argento, oro), questi premi sono solo simbolici, ma in futuro grazie alla collaborazione con i gestori dei locali, si potrebbero aggiungere degli sconti sulle consumazioni o all'ingresso.

Azioni da premiare saranno:

- Pubblicità: se un utente si sforza a promuovere un evento o il sito in generale
- Coinvolgimento: un utente che utilizza molto il sito (diverse visite al giorno, più giorni, completamento del profilo, lasciare molti commenti...)
- Senso civico: se un utente segnala degli eventi/commenti o in genere comportamenti non adatti di altri utenti
- PR: un utente che segnala molti eventi
- Opinione: un utente che effettua molti di voti negativi e positivi
- Popolarità: un utente che segnala eventi molto popolari

Ogni volta che l'utente compierà 10, 100, 1000 volte azioni appartenenti alle categorie prima elencate verrà consegnata una medaglia.

- Collaborazione di comunità
  - La votazione dei migliori eventi per renderli facilmente trovabili da altri porta alla creazione di una comunità interna ad InformaMi e quindi a rafforzare il senso di appartenenza per gli utenti
- Status
  - Il numero e tipo di medaglie aumenteranno lo status degli utenti nei confronti dei novizi.

Queste meccaniche verranno evidenziate tramite le seguenti funzioni:

- Elenco attività
  - Verranno presentate le attività degli amici dell'utente in un feed di notizie
- Profilo utente
  - Ogni utente avrà un profilo sul sito che riassume le sue attività (punteggi, medaglie, eventi, commenti...)

In futuro si potrebbe anche creare una classifica degli utenti ordinati in base al valore delle loro medaglie per promuovere maggiormente i migliori utenti e creare della competizione positiva.

## Capitolo 7 - Test dell'applicazione

*In questo capitolo presentato il concetto di automatizzazione dei test ed il motivo per il quale è bene farlo. Inoltre, vengono mostrati dei test realizzati per l'applicazione.*

Per assicurarsi che l'applicazione sia funzionante al meglio, e che in futuro non vengano introdotti errori in funzionalità già implementate correttamente, è buona pratica creare dei test automatizzati delle funzionalità ed eseguirli ad ogni modifica dell'applicativo.

In questo progetto sono stati creati due tipi di test:

- Unit test – questi verificano il funzionamento di singole unità di codice
- Selenium test – questi verificano il funzionamento dell'applicazione dal punto di vista dell'utente

Verranno presentati i test necessari per garantire il corretto funzionamento dell'aggiunta di locali. Per poter aggiungere un locale è necessario:

- poter creare un utente e successivamente ottenere i suoi dati
- poter convertire un indirizzo in coordinate geografiche
- poter creare un locale e successivamente ottenere i suoi dati

Sono quindi tre unità funzionali distinte che bisognerà testare. In fine si realizzerà un *selenium test* per verificare che queste unità si interfaccino correttamente tra loro.

Prima di ogni test è importante partire da uno stato noto, per questo motivo viene cancellato il contenuto dell'intero database.

```
@Test
public void createUser() {
    try {
        new User("mauriziopz@gmail.com",
                "Maurizio Pozzobon", "1", "facebook", null).insert();
    } catch (Exception e) {}
    User user = User.findById("mauriziopz@gmail.com");
    user = User.findById(user.id);
    assertNotNull(user);
    assertEquals("mauriziopz@gmail.com", user.email);
    assertEquals("Maurizio Pozzobon", user.nome);
    assertEquals("1", user.webId);
    assertEquals(null, user.passwordHash);
}
```

Il primo test crea un utente con email e nome noti. Successivamente carica un utente tramite email, poi lo ricarica tramite id (in modo da testare entrambe le funzioni di caricamento utente). In fine, verifica che l'utente sia stato caricato e che i dati siano corretti.

```
@Test
public void testGeocode() {
    try {
        loc = CommonFunctions.geocode("Rivignano, Italia");
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (JSONException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    assertNotNull(loc.get("latitude"));
    assertNotNull(loc.get("longitude"));
    assertEquals((Double)13.0425, loc.get("longitude"));
    assertEquals((Double)45.8798518, loc.get("latitude"));
}
```

Il secondo test verifica che il funzionamento della conversione tra indirizzo e coordinate geografiche funzioni. Prova a convertire un indirizzo noto e poi verifica che le coordinate coincidano con quelle che si aspetta.

```

@Test
public void createPlace() {
    createUser();
    testGeocode();
    Json json = null;
    json = map().put("original", "http://original")
        .put("small_square", "http://small")
        .put("large_thumbnail", "http://large")
        .put("delete_page", "http://delete");

    User user = User.findByEmail("mauriziopz@gmail.com");
    assertNotNull(user);
    Place place=null;

    try {
        place = new Place(user,"Titolo locale",
            "descrizione del locale creato tramite test",json,
            "Rivignano, Italia",loc.get("latitude"),
            loc.get("longitude"));
    } catch (Exception e) {
        assertEquals(0,1);
    }
    place.insert();
    user = User.findByEmail("mauriziopz@gmail.com");
    List<Place> p = user.places.fetch();
    assertNotNull(p);
    assertFalse(p.isEmpty());
    for (Place pl : p) {
        assertEquals("Titolo locale",pl.nome);
        assertEquals("descrizione del locale creato tramite test",
            pl.descrizione);
        assertNotNull(pl.imageLinks);

        assertEquals("http://original",
            pl.imageLinks.get("original").asString());
        assertEquals("http://small",
            pl.imageLinks.get("small_square").asString());
        assertEquals("http://large",
            pl.imageLinks.get("large_thumbnail").asString());
        assertEquals("http://delete",
            pl.imageLinks.get("delete_page").asString());

        assertEquals("Rivignano, Italia",pl.indirizzo);
        assertNotNull(pl.latitude);
        assertNotNull(pl.longitude);
    }
}

```

Il terzo test esegue i primi due per ottenere i dati che essi generano (come è stato detto in precedenza, prima di ogni test il database viene cancellato). Successivamente prepara un JSON con gli URL dell'immagine. Ottiene l'utente creato in precedenza, crea un locale con dei dati predefiniti e poi lo inserisce nel sistema.

A questo punto ricarica l'utente dal sistema (per avere una versione aggiornata), ottiene da esso l'elenco dei locali che ha creato, esegue un ciclo su tutti i locali verificando che i dati siano quelli dell'unico locale che dovrebbe essere presente nel sistema.

```
void testCreatePlace() throws Exception {
    selenium.open("/events/index")
    selenium.clickAndWait("css=img[alt=\"Login with facebook\"]")
    assertTrue(selenium.isTextPresent(
        "Maurizio Pozzobon"))
    verifyTrue(selenium.isTextPresent(
        "Crea evento"))
    verifyTrue(selenium.isTextPresent(
        "Crea locale"))
    selenium.clickAndWait(
        "//div[@id='sidebar']/ul/li[2]/div/ul/li[2]/a/span")
    selenium.type("name=nome", "Selenium Locale")
    selenium.type("name=descrizione",
        "Descrizione di un locale creato tramite selenium")
    selenium.type("id=indirizzo", "udine, viale venezia")
    selenium.clickAndWait("css=input[type=\"submit\"]")
    selenium.clickAndWait("//ul[@id='navigation']/li[2]/a/span")
    verifyEquals("Selenium Locale",
        selenium.getText("link=Selenium Locale"))
    selenium.clickAndWait("link=Selenium Locale")
    verifyEquals("Selenium Locale",
        selenium.getText("link=Selenium Locale"))
    verifyTrue(selenium.isTextPresent(
        "Descrizione di un locale creato tramite selenium"))
    verifyTrue(selenium.isTextPresent("udine, viale venezia"))
    selenium.clickAndWait("//ul[@id='navigation']/li[4]/a/span")
}
```

L'ultimo test è quello che verifica direttamente il funzionamento per l'interfaccia grafica. Bisogna quindi delineare dei comandi che eseguirà il web browser.

Per la buona riuscita di questo test è necessario prima effettuare l'accesso a Facebook con il browser che si intende utilizzare ed aver già concesso i permessi di accesso ad InformaMi. Il test per prima cosa apre la homepage di InformaMi, poi clicca sul link per effettuare il login con Facebook. Verifica che l'utente sia effettivamente loggato. Verifica la presenza dei link



per creare gli eventi ed i locali. Successivamente clicca su quello per creare un locale. Inserisce un nome, una descrizione e l'indirizzo del locale. Poi lo crea.

Apri la pagina con l'elenco dei locali e verifica che sia presente. Clicca sul titolo del locale e verifica che tutti i dati corrispondano a quello del locale inserito. In fine clicca sul link per sloggarsi.

## Conclusione

Gli obiettivi che si erano delineati all'inizio della tesi per InformaMi sono stati quasi completamente raggiunti. Si è creato un'applicazione web facile da utilizzare e che permette ai suoi visitatori di scoprire i migliori eventi vicini a casa loro. È accessibile a chiunque, e chi ha un account su Facebook, può pubblicare, commentare e giudicare gli eventi ed i locali che li ospitano. Inoltre, gli utenti possono caricare le foto che hanno scattato agli eventi e possono *taggarsi* in esse. Vengono gestite le amicizie tra gli utenti ed essi vengono informati quando amici pubblicano eventi oppure interagiscono con eventi già pubblicati. In fine, gli eventi sono ricercabili per data, zona e caratteristiche.

L'unico obiettivo non raggiunto è quello di sostituire il volantinaggio agli occhi degli organizzatori degli eventi. Effettivamente questo era un obiettivo troppo grande per il tempo disponibile. È necessario un forte sforzo di marketing per poterlo raggiungere. Si spera che con il tempo, e con l'aumento di popolarità di InformaMi tra gli studenti, si riesca a raggiungere anche questo obiettivo.

Per completare questi obiettivi si è sviluppato un'applicazione con approssimativamente 3000 linee di codice, 26 classi, 13 viste e 2 classi di test. Queste verranno allegate nell'appendice A, inoltre sono stati caricati su <https://github.com/MaurizioPz/InformaMi> per essere più facilmente accessibili.

L'applicazione realizzata, con delle modifiche al modello del database, può essere utilizzata per scopi diversi. Il più generale è quello della semplice condivisione di link interessanti, ma potrebbe facilmente prestarsi per creare un sito di domande e risposte oppure ancora per la condivisione di ricette stagionali/regionali. Come però è stato delineato nell'introduzione si è deciso di specializzarlo per la condivisione di eventi, questo permette di aggiungere delle funzionalità particolarmente importanti per il suo scopo, come la ricerca per luogo e la condivisione di foto.

In futuro InformaMi può essere ampliato maggiormente, permettendo l'integrazione con altri social network (ad esempio Google+ e Twitter) e permettendo di pubblicizzare su Facebook gli eventi pubblicati su InformaMi. Il principale obiettivo per il futuro però, è quello di aumentare la sua popolarità, quindi è necessario pubblicizzarlo tra gli studenti universitari di tutta Italia, non solo a Trieste, ed anche tra i proprietari dei locali.

Dal punto di vista tecnico si può migliorare InformaMi aumentando il numero di test automatici, per verificare tutte le funzionalità, non solo quelle più importanti, ed implementando una ricerca interna in modo da sostituire quella di Google. Altre modifiche

possono essere quelle di aumentare la possibilità di comparire tra i risultati dei motori di ricerca mediante delle ottimizzazioni (ad esempio i titoli delle pagine, il testo dei link, l'aspetto degli URL delle pagine...).

Un'ultima modifica tecnologica ad InformaMi da fare quando risulterà necessario, è l'ottimizzazione per migliorare la velocità di risposta ed aumentare il numero utenti contemporanei che si possono servire. Come è stato detto in precedenza aumentando la ridondanza dei dati nel database è possibile semplificare e velocizzare la ricerca di eventi per zona e di altre funzionalità. Questa modifica non è stata fatta nel corso della tesi in quanto si è ritenuto non necessario concentrarsi su quell'aspetto, dato che il carico attuale sul sistema è molto ridotto.

# Bibliografia

- <http://www.playframework.org/documentation/> - documentazione di Play! Framework
- <http://code.google.com/appengine/> - documentazione dell'API di Google App Engine
- <http://restfb.com/> - documentazione della libreria RestFB
- <https://developers.facebook.com> – documentazione dell'API di Facebook
- <http://www.sienaproject.com/> - documentazione della libreria Siena
- <http://download.oracle.com/javase/6/docs/api/> - documentazione per Java
- <http://eclipse.org/> - IDE utilizzato per lo sviluppo
- [http://api.imgur.com/resources\\_anon](http://api.imgur.com/resources_anon) - documentazione API per il servizio di hosting delle immagini
- <http://www.google.com/cse/> - servizio di motore di ricerca personalizzato
- <http://code.google.com/apis/maps/documentation/geocoding/> - documentazione API per la conversione di un indirizzo in coordinate geografiche
- <http://balsamiq.com/> - strumento per creare i mockup dell'interfaccia grafica
- <http://stackoverflow.com/> - ispirazione iniziale ed utile per risolvere problemi e dubbi vari
- <http://digg.com/> - ispirazione iniziale
- <http://digitalnature.eu/themes/mystique/> tema grafico utilizzato come base per l'interfaccia di InformaMi
- <https://github.com/MaurizioPz/InformaMi> - repository per il sorgente di InformaMi

## Appendice A – Allegato codice

*Models/EventCommentVote.java*

```
package models;

import siena.Column;
import siena.Id;
import siena.Model;

public class EventCommentVote extends Model {
    @Id
    public Long id;
    public Boolean isPositive;

    //Relazioni
    @Column("user")
    public User user;
    @Column("eventComment")
    public EventComment eventComment;
    /**
     * @param user
     * @param eventComment
     * @param isPositive è un commento positivo?
     * @throws Exception
     */
    public EventCommentVote(User user, EventComment eventComment,
        Boolean isPositive) throws Exception {
        if(user!=null)
            this.user = user;
        else
            throw new Exception("User can't be null");
        if(eventComment!=null)
            this.eventComment = eventComment;
        else
            throw new Exception("Event Comment can't be null");
        this.isPositive = isPositive;
    }
}
```

---

*Models/PlaceCommentVote.java*

```
package models;

import siena.Column;
import siena.Id;
import siena.Model;
```

```

public class PlaceCommentVote extends Model {
    @Id
    public Long id;
    public Boolean isPositive;

    //Relazioni
    @Column("user")
    public User user;
    @Column("placeComment")
    public PlaceComment placeComment;
    /**
     * @param user utente che ha votato
     * @param placeComment commentoLocale votato
     * @param isPositive è un voto positivo?
     * @throws Exception
     */
    public PlaceCommentVote(User user, PlaceComment placeComment,
Boolean isPositive) throws Exception {
        if(user!=null)
            this.user = user;
        else
            throw new Exception("User can't be null");
        if(placeComment!=null)
            this.placeComment = placeComment;
        else
            throw new Exception("Place Comment can't be null");
        this.isPositive = isPositive;
    }
}

```

---

*Models/TipoAzione.java*

```

package models;

import siena.Filter;
import siena.Id;
import siena.Model;
import siena.Query;

public class TipoAzione extends Model{

    @Id
    public Long id;

    public String name;

    //Relazioni
    @Filter("actionType")
    public Query<Action> actions;
}

```

```

/**
 * @param name
 */
public TipoAzione(String name) {
    this.name = name;
}

public static TipoAzione find(String string) {
    TipoAzione tp = all().filter("name", string).get();
    if(tp==null){
        tp = new TipoAzione(string);
        tp.insert();
    }
    return tp;
}
private static Query<TipoAzione> all() {
    return Model.all(TipoAzione.class);
}

public static TipoAzione findById(Long id) {
    return all().filter("id", id).get();
}
}

```

---

*Models/Photo.java*

```

package models;

import java.util.Date;

import play.mvc.Router;

import siena.Column;
import siena.Filter;
import siena.Id;
import siena.Json;
import siena.Model;
import siena.Query;

public class Photo extends Model {

    @Id
    public Long id;
    public Date data;
    public Json imageLinks;

    //Relazioni
    @Column("user")
    public User user;
}

```

```

@Column("event")
public Event event;

@Filter("photo")
public Query<Presence> presences;

/**
 * @param data Quanto è stata caricata la foto
 * @param url URL dove si trova la foto
 * @param idUser id dell'utente che ha caricato la foto
 * @param idEvent id dell'evento al quale appartiene la foto
 * @throws Exception
 */
public Photo(User user, Event event, Date data, Json
imageLinks) throws Exception {
    this.data = data;
    this.imageLinks = imageLinks;
    if(user!=null)
        this.user = user;
    else
        throw new Exception("User can't be null");
    if(event!=null)
        this.event = event;
    else
        throw new Exception("Event can't be null");
}

private static Query<Photo> allModel() {
    return Model.all(Photo.class);
}

public static Photo findById(Long id) {
    return allModel().filter("id", id).get();
}

@Override
public void insert() {
    super.insert();
    TipoAzione tp = TipoAzione.find("caricato una foto");
    try {
        new Action(user, tp, new Date(),
Router.getFullUrl("Photos.show")+"?id="+this.id).insert();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}

```

---

*Models/User.java*



```

package models;

import java.util.ArrayList;
import java.util.List;

import siena.Column;
import siena.Filter;
import siena.Id;
import siena.Max;
import siena.Model;
import siena.NotNull;
import siena.Query;
import siena.embed.Embedded;

public class User extends Model {

    @Id
    public Long id;

    public String nome;
    public String email;
    public String webId; //ID of the user in the provider website
    public String service; //Provider website
    public String passwordHash;

    //Relazioni
    @Filter("user")
    public Query<Event> events;
    @Filter("user")
    public Query<Place> places;
    @Filter("user")
    public Query<EventVote> eventVotes;
    @Filter("user")
    public Query<EventComment> eventComments;
    @Filter("user")
    public Query<EventCommentVote> eventCommentVotes;
    @Filter("user")
    public Query<PlaceVote> placeVotes;
    @Filter("user")
    public Query<PlaceComment> placeComments;
    @Filter("user")
    public Query<PlaceCommentVote> placeCommentVotes;
    @Filter("user")
    public Query<Action> actions;
    @Filter("user")
    public Query<Medal> medals;
    @Filter("user")
    public Query<Tag> tags;
    @Filter("user")
    public Query<Presence> presences;

```

```

@Filter("user")
public Query<Photo> photos;

@Embedded
private List<Long> idFriends= new ArrayList<Long>();

/**
 *
 * @param email
 * @param name
 * @param webId id of this user in the provider website
 * @param service provider website where the user is signed in
 (like facebook)
 * @param passwordHash hash of the password
 * @throws Exception
 */
public User(String email, String name,String webId, String
service,String passwordHash) throws Exception {
    if (email!=null)
        this.email=email;
    else
        throw new Exception("An email is required");
    if (name!=null)
        this.name=name;
    else
        throw new Exception("A name is required");

    this.webId=webId;
    this.passwordHash = passwordHash;
    this.service = service;
}

public void setEmail(String email) throws Exception{
    if (email!=null)
        this.email=email;
    else
        throw new Exception("An email is needed");
}

static Query<User> all() {
    return Model.all(User.class);
}

public static User findById(Long id) {
    return all().filter("id", id).get();
}

public static User findByEmail(String email){
    return all().filter("email", email).get();
}

```

```

public String toString() {
    return nome;
}

public void addFriend(User userFriend){
    if(userFriend!=null){
        if(!idFriends.contains(userFriend.id)){
            idFriends.add(userFriend.id);
            this.update();
        }
    }
}

public void removeFriend(User userFriend){
    if(userFriend!=null){
        if(idFriends.contains(userFriend.id)){
            idFriends.remove(userFriend.id);
            this.update();
        }
    }
}

public boolean isFriend(User userFriend){
    if(userFriend!=null){
        if(idFriends.contains(userFriend.id))
            return true;
    }
    return false;
}

public List<Long> getFriends(){
    return idFriends;
}

public List<Action> getFriendsActions(){
    List<Action> actions = new ArrayList<Action>();
    for (Long id : idFriends) {
        actions.add(User.findById(id).actions.order("-
data").get());
    }
    return actions;
}
}

```

---

*Models/Event.java*

```

package models;

import java.io.File;
import java.util.ArrayList;
import java.util.Collections;

```

```

import java.util.Comparator;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import play.mvc.Router;

import com.google.appengine.api.blobstore.BlobKey;

import siena.Column;
import siena.Filter;
import siena.Id;
import siena.Json;
import static siena.Json.*;
import siena.Model;
import siena.Query;
import siena.embed.Embedded;

public class Event extends Model{
    @Id
    public Long id;

    public String nome;
    public String nomePiccolo; //necessario per ricerca
    public String descrizione;
    public Json imageLinks;
    public Date dataInizio;
    public Date dataFine;
    public Boolean isRicorrente;
    public Long ricorrenza;          //espresso in secondi

    public Integer points;

    //Relazioni
    @Column("user")
    public User user;
    @Column("place")
    public Place place;

    @Filter("event")
    public Query<EventComment> comments;
    @Filter("event")
    public Query<EventVote> votes;
    @Filter("event")
    public Query<Photo> photos;
    @Embedded
    private List<Long> idTags= new ArrayList<Long>();

```

```

/**
 * @param nome
 * @param descrizione
 * @param string
 * @param dataInizio
 * @param durata
 * @param isRicorrente
 * @param ricorrenza
 * @param tagList
 * @throws Exception
 */
public Event(User user, Place place, String nome, String
descrizione, Json imageLinks2, Date dataInizio, Date dataFine,
Boolean isRicorrente, Long ricorrenza, List<Tag> tagList) throws
Exception {
    if(user!=null)
        this.user = user;
    else
        throw new Exception("User can't be null");
    if(place!=null)
        this.place = place;
    else
        throw new Exception("Place can't be null");

    this.nome = nome;
    nomePiccolo = nome.toLowerCase();
    this.descrizione = descrizione;
    if(imageLinks2!=null)
        this.imageLinks = imageLinks2;
    else{
        this.imageLinks=map().put("original",
"http://placekitten.com/600/400")
                                .put("small_square",
"http://placekitten.com/64/64")
                                .put("large_thumbnail"
, "http://placekitten.com/300/200")
                                .put("delete_page",
"" );
    }
    this.dataInizio = dataInizio;
    this.dataFine = dataFine;
    this.isRicorrente = isRicorrente;
    this.ricorrenza = ricorrenza;
    for (Tag tag : tagList) {
        idTags.add(tag.id);
    }
}
@Override
public void insert() {

```

```

        super.insert();
        Tag t;
        for (Long id : idTags) {
            t=Tag.findById(id);
            t.addEvent(this.id);
            t.update();
        }

        TipoAzione tp = TipoAzione.find("creato un evento");
        try {
            new Action(user, tp, new Date(),
Router.getFullUrl("Events.show")+"?id="+this.id).insert();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    static Query<Event> all() {
        return Model.all(Event.class);
    }

    public static Event findById(Long id) {
        return all().filter("id", id).get();
    }

    public String toString() {
        return nome;
    }

    public static List<Event> getAllByPoints(Double latitude,
Double longitude){
        List<Event> l = getAll(true,latitude,longitude);
        Collections.sort(l, new Comparator<Event>() {@Override
        public int compare(Event e1, Event e2) {
            if(e1.points>e2.points)
                return -1;
            else
                return 1;
        }
        });
        return l;
    }

    public static List<Event> getAll(boolean
notExpires,Double latitude, Double longitude) {
        List<Event> l;
        if(notExpires)
            l= all().filter("dataFine>", new
Date()).fetch();
        else

```

```

        l= all().fetch();
        if((latitude!=null)&&(longitude!=null)){
            List<Event> tempList = new ArrayList<Event>();
            for (Event event : l) {

                event.place=Place.findById(event.place.id); //TODO Better
performance
                if(event.place.isNear(latitude,longitude))
                    tempList.add(event);
            }
            l=tempList;
        }

        return l;
    }

    public int getPoints(){
        List<EventVote> votesP = votes.filter("isPositive",
true).fetch();
        List<EventVote> votesN = votes.filter("isPositive",
false).fetch();
        this.points= votesP.size()-votesN.size();
        return this.points;
    }
    public void removeTag(Long id) throws Exception{
        if(idTags.contains(id))
            idTags.remove(id);
        else
            throw new Exception("Tag not found");
    }

    public List<Long> getTags(){
        return idTags;
    }

    @Override
    public void delete() {
        for (Long id : idTags) {
            try {
                Tag.findById(id).removeEvent(this.id);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        super.delete();
    }
    public void addVote(User user, Boolean isPositive) throws
Exception {
        EventVote e=EventVote.findByIdUserAndEvent(user,this);
        if(e==null)

```

```

        new EventVote(user, this, isPositive).insert();
    else if(!e.isPositive.equals(isPositive))
        e.delete();

    }

    public static List<Event> search(){
        return all().search("ritorniamo",
"descrizione").fetch();
    }

}

```

---

*Models/EventComment.java*

```

package models;

import java.util.Date;

import play.mvc.Router;

import siena.Column;
import siena.Filter;
import siena.Id;
import siena.Model;
import siena.Query;
import siena.embed.Embedded;

public class EventComment extends Model {

    @Id
    public Long id;

    public String testo;
    public Date data;

    //Relazioni
    @Column("user")
    public User user;
    @Column("event")
    public Event event;

    @Filter("eventComment")
    public Query<EventCommentVote> votes;

    /**
     * @param user
     * @param event

```



```

    * @param testo
    * @param data
    * @throws Exception
    */
    public EventComment(User user, Event event, String testo, Date
data) throws Exception {
        if(user!=null)
            this.user = user;
        else
            throw new Exception("User can't be null");
        if(event!=null)
            this.event = event;
        else
            throw new Exception("Event can't be null");
        this.testo = testo;
        this.data = data;
    }

    public String toString() {
        return testo;
    }

    static Query<EventComment> all() {
        return Model.all(EventComment.class);
    }

    public static EventComment findById(Long id) {
        return all().filter("id", id).get();
    }
    public String getUsername(){
        if(user==null)
            return "NULL";
        else
            return user.nome;
    }
    @Override
    public void insert() {
        super.insert();
        TipoAzione tp = TipoAzione.find("commentato un evento");
        try {
            new Action(user, tp, new Date(),
Router.getFullUrl("Events.show")+"?id="+this.event.id).insert();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

---

*Models/EventVote.java*

```

package models;

import java.util.Date;
import java.util.List;

import play.mvc.Router;

import siena.Column;
import siena.Id;
import siena.Model;
import siena.Query;

public class EventVote extends Model {

    @Id
    public Long id;
    public Boolean isPositive;

    //Relazioni
    @Column("user")
    public User user;
    @Column("event")
    public Event event;

    /**
     * @param user
     * @param event
     * @param isPositive è un commento positivo?
     * @throws Exception
     */
    public EventVote(User user, Event event, Boolean isPositive)
throws Exception {
        if(user!=null)
            this.user = user;
        else
            throw new Exception("User can't be null");
        if(event!=null)
            this.event = event;
        else
            throw new Exception("Event can't be null");
        this.event = event;
        this.isPositive = isPositive;
    }

    static Query<EventVote> all() {
        return Model.all(EventVote.class);
    }
    public static EventVote findById(Long id) {
        return all().filter("id", id).get();
    }
}

```

```

        public static EventVote findByUserAndEvent(User user, Event
event) {
            return all().filter("user", user).filter("event",
event).get();
        }
        @Override
        public void insert() {
            super.insert();
            TipoAzione tp = TipoAzione.find("votato un evento");
            try {
                new Action(user, tp, new Date(),
Router.getFullUrl("Events.show")+"?id="+this.event.id).insert();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

```

---

*Models/Tag.java*

```

package models;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.List;

import play.mvc.Router;
import play.mvc.Scope.Session;

import com.sun.jndi.url.ldaps.LdapsURLContextFactory;

import siena.Column;
import siena.Filter;
import siena.Id;
import siena.Model;
import siena.Query;
import siena.embed.Embedded;
import siena.remote.Common;

import lib.*;

public class Tag extends Model implements Comparable<Tag> {

    @Id
    public Long id;
    public String nome;
}

```

```

//Relazioni
@Column("user")
public User user;
@Embedded
private List<Long> idEvents= new ArrayList<Long>();

/**
 * @param nome indica il nome del Tag
 * @param user indica l'utente che ha creato questo tag
 * @throws Exception
 */
public Tag(User user,String nome) throws Exception {
    this.nome = nome;
    if(user!=null)
        this.user = user;
    else
        throw new Exception("User can't be null");
}

public static Query<Tag> all() {
    return Model.all(Tag.class);
}

public static Tag findById(Long id) {
    return all().filter("id", id).get();
}

public static List<Tag> filterByName(String search) {
    return all().search(search+"*", "nome").fetch();
}

public static Tag findByName(String search, Session session)
throws Exception {
    Tag t = all().search(search.replace(" ", "-"),
"nome").get();

    if(t==null){
        if(session!=null){
            t=new
Tag(CommonFunctions.getLoggedInUser(session),search.replace(" ",
"-"));

            t.insert();
        }
    }
    return t;
}

public void removeEvent(Long id) throws Exception {
    if(idEvents.contains(id))
        idEvents.remove(id);
    else

```

```

        throw new Exception("Event not found");
    }
    public void addEvent(Long id) {
        if(!idEvents.contains(id))
            idEvents.add(id);
    }

    @Override
    public void delete() {
        for (Long id : idEvents) {
            try {
                Event.findById(id).removeTag(this.id);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        super.delete();
    }

    public List<Event> getEvents() {
        List<Event> events= new ArrayList<Event>();
        Event tmp=null;
        for (Long idEvent : idEvents) {
            tmp = Event.findById(idEvent);
            if(tmp!=null)
                events.add(tmp);
        }
        //if(events.size()==0)
        //events=null;
        return events;
    }

    public static List<Tag> getTagCloud(){
        List<Tag> l = all().fetch();
        Collections.sort(l);
        return l;
    }

    @Override
    public int compareTo(Tag o) {
        if(this.idEvents.size()<o.idEvents.size())
            return 1;
        else if(this.idEvents.size()==o.idEvents.size())
            return 0;
        else
            return -1;
    }

    @Override
    public void insert() {
        super.insert();
    }

```

```

        TipoAzione tp = TipoAzione.find("creato un etichetta");
        try {
            new Action(user, tp, new Date(), "#").insert();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

---

*Models/Presence.java*

```

package models;

import java.util.Date;

import play.mvc.Router;
import siena.Column;
import siena.Id;
import siena.Model;

public class Presence extends Model {
    @Id
    public Long id;
    public Integer PosizioneX;
    public Integer PosizioneY;

    //Relazioni
    @Column("user")
    public User user;
    @Column("photo")
    public Photo photo;

    /**
     * @param posizioneX indica la posizione sull'asse delle x del
utente da cui l'idUser
     * @param posizioneY indica la posizione sull'asse delle y del
utente da cui l'idUser
     * @param user utente presente nella foto da cui photo
     * @param photo foto al quale si riferisce questa presenza
     */
    public Presence(User user, Photo photo, Integer posizioneX,
Integer posizioneY) {
        PosizioneX = posizioneX;
        PosizioneY = posizioneY;
        this.photo=photo;
        this.user=user;
    }
    @Override

```

```

        public void insert() {
            super.insert();
            TipoAzione tp = TipoAzione.find("taggato se stesso in una
foto");
            try {
                new Action(user, tp, new Date(),
Router.getFullUrl("Photos.show")+"?id="+this.photo.id).insert();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

```

---

*Models/PlaceVote.java*

```

package models;

import java.util.Date;

import play.mvc.Router;
import siena.Column;
import siena.Id;
import siena.Model;

public class PlaceVote extends Model {

    @Id
    public Long id;
    public Boolean isPositive;

    //Relazioni
    @Column("user")
    public User user;
    @Column("place")
    public Place place;

    /**
     * @param user
     * @param place
     * @param isPositive
     * @throws Exception
     */
    public PlaceVote(User user, Place place, Boolean isPositive)
throws Exception {
        if(user!=null)
            this.user = user;
        else
            throw new Exception("User can't be null");
        if(place!=null)

```

```

        this.place = place;
    else
        throw new Exception("Place can't be null");
    this.isPositive = isPositive;
}
@Override
public void insert() {
    super.insert();
    TipoAzione tp = TipoAzione.find("votato un locale");
    try {
        new Action(user, tp, new Date(),
Router.getFullUrl("Places.show")+"?id="+this.place.id).insert();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}

```

---

*Models/PlaceComment.java*

```

package models;

import java.util.Date;

import play.mvc.Router;

import siena.Column;
import siena.Filter;
import siena.Id;
import siena.Model;
import siena.Query;

public class PlaceComment extends Model {

    @Id
    public Long id;

    public String testo;
    public Date data;

    //Relazioni
    @Column("user")
    public User user;
    @Column("place")
    public Place place;

    @Filter("placeComment")
    public Query<PlaceCommentVote> votes;
}

```



```

/**
 *
 * @param user
 * @param place
 * @param testo
 * @param data
 * @throws Exception
 */
public PlaceComment(User user, Place place, String testo, Date
data) throws Exception {
    if(user!=null)
        this.user = user;
    else
        throw new Exception("User can't be null");
    if(place!=null)
        this.place = place;
    else
        throw new Exception("Place can't be null");
    this.testo = testo;
    this.data = data;

}
@Override
public void insert() {
    super.insert();
    TipoAzione tp = TipoAzione.find("commentato un locale");
    try {
        new Action(user, tp, new Date(),
Router.getFullUrl("Places.show")+"?id="+this.place.id).insert();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}

```

---

*Models/Place.java*

```

package models;

import static siena.Json.map;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Map;

```

```

import play.mvc.Router;

import com.google.appengine.repackaged.org.json.JSONException;

import lib.CommonFunctions;

import siena.Column;
import siena.Filter;
import siena.Id;
import siena.Json;
import siena.Model;
import siena.Query;

public class Place extends Model {

    @Id
    public Long id;

    public String nome;
    public String nomePiccolo;
    public String descrizione;
    public Json imageLinks;
    public String indirizzo;
    public Integer points;

    public Double latitude;
    public Double longitude;
    //Relazioni
    @Column("user")
    public User user;
    @Filter("place")
    public Query<Event> events;
    @Filter("place")
    public Query<PlaceVote> votes;
    @Filter("place")
    public Query<PlaceComment> comments;

    /**
     * @param nome indica il nome del locale
     * @param descrizione
     * @param imageLinks
     * @param indirizzo
     * @throws Exception
     */
    public Place(User user, String nome, String descrizione, Json
imageLinks,String indirizzo,Double latitude, Double longitude)

```

```

throws Exception {
    if(user!=null)
        this.user = user;
    else
        throw new Exception("User can't be null");
    this.nome = nome;
    nomePiccolo=nome.toLowerCase();
    this.descrizione = descrizione;
    if((imageLinks!=null)&&(!imageLinks.equals(""))))
        this.imageLinks = imageLinks;
    else{
        this.imageLinks=map().put("original",
"http://placekitten.com/600/400")
        .put("small_square", "http://placekitten.com/64/64")
        .put("large_thumbnail",
"http://placekitten.com/300/200")
        .put("delete_page", "");
    }

    this.indirizzo = indirizzo;
    this.latitude=latitude;
    this.longitude=longitude;
}

static Query<Place> all() {
    return Model.all(Place.class);
}
public static List<Place> getAll(Double latitude, Double
longitude) {
    List<Place> l = all().fetch();
    if(latitude!=null){
        List<Place> tempList= new ArrayList<Place>();
        for (Place place : l) {
            if(place.isNear(latitude, longitude))
                tempList.add(place);
        }
        l=tempList;
    }
    return l;
}

public static List<Place> filterByName(String name) {

    return all().search(name+"*", "nomePiccolo").fetch();
}

public static Place findById(Long id) {
    return all().filter("id", id).get();
}

```

```

    }
    public static Place findByString(String s) {
        String[] split = s.split(":");
        return all().filter("nome", split[0]).filter("indirizzo",
split[1]).get();
    }

    public String toString() {
        return nome;
    }
    public int getPoints(){
        int votesP = votes.filter("isPositive", true).count();
        int votesN = votes.filter("isPositive", false).count();
        this.points= votesP-votesN;
        return this.points;
    }

    public boolean isNear(Double latitude2, Double longitude2) {
        Double diff = 0.2;
        try{
            if((latitude<latitude2+diff)&&(latitude>latitude2-
diff)&&(longitude<longitude2+diff)&&(longitude>longitude2-diff))
                return true;
        }catch (Exception e) {
            if(latitude==null){
                try {
                    Map<String, Double> map =
CommonFunctions.geocode(indirizzo);
                    latitude=map.get("latitude");
                    longitude=map.get("longitude");
                    this.update();
                } catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                } catch (JSONException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
            }
        }
        return false;
    }
    @Override
    public void insert() {
        super.insert();
        TipoAzione tp = TipoAzione.find("creato un locale");
        try {
            new Action(user, tp, new Date(),
Router.getFullUrl("Places.show")+"?id="+this.id).insert();

```

```

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

---

*Models/TipoMedaglia.java*

```

package models;

import java.util.HashSet;
import java.util.Set;

import siena.Filter;
import siena.Id;
import siena.Model;
import siena.Query;

public class TipoMedaglia extends Model{

    @Id
    public Long id;

    public String name;
    public Integer value;

    //Relazioni
    @Filter("medalType")
    public Query<Medal> medals;

    /**
     * @param name
     * @param value
     */
    public TipoMedaglia(String name, Integer value) {
        this.name = name;
        this.value = value;
    }

    public static TipoMedaglia find(String string) {
        TipoMedaglia tp = all().filter("name", string).get();
        if(tp==null){
            tp = new TipoMedaglia(string,1);
            tp.insert();
        }
        return tp;
    }
    private static Query<TipoMedaglia> all() {

```

```

        return Model.all(TipoMedaglia.class);
    }

    public static TipoMedaglia findById(Long id) {
        return all().filter("id", id).get();
    }
}

```

---

*Models/Action.java*

```

package models;

import java.util.Date;

import siena.Column;
import siena.Id;
import siena.Model;

public class Action extends Model {

    @Id
    public Long id;
    public Date data;
    public String URL;

    //Relazioni
    @Column("user")
    public User user;
    @Column("actionType")
    public TipoAzione actionType;

    /**
     * @param data data in cui è stata compiuta l'azione
     * @param uRL URL che si riferisce all'azione
     * @param idUser utente che ha compiuto l'azione
     * @param idActionType Azione compiuta
     * @throws Exception
     */
    public Action(User user, TipoAzione tpAzione, Date data,
String uRL) throws Exception {
        if(user!=null)
            this.user = user;
        else
            throw new Exception("User can't be null");
        if(user!=null)
            this.actionType = tpAzione;
        else
            throw new Exception("TipoAzione can't be null");
        this.data = data;
        URL = uRL;
    }
}

```

```

    }
    @Override
    public String toString() {
        return user.nome + " ha " + actionType.name;
    }

    @Override
    public void insert() {
        super.insert();
        try {
            if (user.actions.count() > 10) {
                if (user.medals.filter("medalType",
TipoMedaglia.find("Primino")).count() == 0)
                    new
Medal(TipoMedaglia.find("Primino"), user, new Date()).insert();
            }
            if (user.actions.count() > 100) {
                if (user.medals.filter("medalType",
TipoMedaglia.find("Novizio")).count() == 0)
                    new
Medal(TipoMedaglia.find("Novizio"), user, new Date()).insert();
            }
            if (user.actions.count() > 1000) {
                if (user.medals.filter("medalType",
TipoMedaglia.find("Intermedio")).count() == 0)
                    new
Medal(TipoMedaglia.find("Intermedio"), user, new Date()).insert();
            }
            if (user.actions.count() > 10000) {
                if (user.medals.filter("medalType",
TipoMedaglia.find("Esperto")).count() == 0)
                    new
Medal(TipoMedaglia.find("Esperto"), user, new Date()).insert();
            }
            if (user.events.count() > 10) {
                if (user.medals.filter("medalType",
TipoMedaglia.find("Promotore")).count() == 0)
                    new
Medal(TipoMedaglia.find("Promotore"), user, new Date()).insert();
            }
            if (user.events.count() > 100) {
                if (user.medals.filter("medalType",
TipoMedaglia.find("Organizzatore")).count() == 0)
                    new
Medal(TipoMedaglia.find("Organizzatore"), user, new Date()).insert();
            }
            if (user.events.count() > 1000) {
                if (user.medals.filter("medalType",
TipoMedaglia.find("PR Ufficiale")).count() == 0)
                    new Medal(TipoMedaglia.find("PR
Ufficiale"), user, new Date()).insert();
            }
        }
    }

```

```

        }
        if (user.eventComments.count ()
+user.placeComments.count ()>10) {
            if (user.medals.filter ("medalType",
TipoMedaglia.find ("Commentatore")) .count ()==0)
                new
Medal (TipoMedaglia.find ("Commentatore"), user, new Date ()) .insert ();
        }
        if (user.eventComments.count ()
+user.placeComments.count ()>100) {
            if (user.medals.filter ("medalType",
TipoMedaglia.find ("Opinionista")) .count ()==0)
                new
Medal (TipoMedaglia.find ("Commentatore"), user, new Date ()) .insert ();
        }
        if (user.eventComments.count ()
+user.placeComments.count ()>1000) {
            if (user.medals.filter ("medalType",
TipoMedaglia.find ("Politico")) .count ()==0)
                new
Medal (TipoMedaglia.find ("Commentatore"), user, new Date ()) .insert ();
        }
        if (user.eventVotes.count ()
+user.placeVotes.count ()>10) {
            if (user.medals.filter ("medalType",
TipoMedaglia.find ("Giudice")) .count ()==0)
                new
Medal (TipoMedaglia.find ("Commentatore"), user, new Date ()) .insert ();
        }
        if (user.eventComments.count ()
+user.placeCommentVotes.count ()>10) {
            if (user.medals.filter ("medalType",
TipoMedaglia.find ("Giudicatore")) .count ()==0)
                new
Medal (TipoMedaglia.find ("Commentatore"), user, new Date ()) .insert ();
        }
        if (user.eventComments.count ()
+user.placeCommentVotes.count ()>10) {
            if (user.medals.filter ("medalType",
TipoMedaglia.find ("Votatore estremo")) .count ()==0)
                new
Medal (TipoMedaglia.find ("Commentatore"), user, new Date ()) .insert ();
        }

    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace ();
    }
}
}

```



---

*Models/Medal.java*

```
package models;

import java.util.Date;

import siena.Column;
import siena.Id;
import siena.Model;

public class Medal extends Model {

    @Id
    public Long id;
    public Date data;

    //Relazioni
    @Column("user")
    public User user;
    @Column("medalType")
    public TipoMedaglia medalType;

    /**
     * @param data
     * @param medalType
     * @param user
     * @throws Exception
     */
    public Medal(TipoMedaglia medalType, User user, Date data)
    throws Exception {
        if(user!=null)
            this.user = user;
        else
            throw new Exception("User can't be null");
        if(medalType!=null)
            this.medalType = medalType;
        else
            throw new Exception("Tipo Medaglia can't be null");
        this.data = data;
    }
    @Override
    public String toString() {
        return medalType.name;
    }
}
```

---

*Lib/CommonFunctions.java*

```
package lib;

import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;
import java.util.HashMap;
import java.util.Map;

import models.User;
import play.data.Upload;
import play.mvc.Scope.Session;
import siena.Json;
import static siena.Json.*;
import siena.sdb.ws.Base64;

import com.google.appengine.repackaged.org.json.JSONException;
import com.google.appengine.repackaged.org.json.JSONObject;

public class CommonFunctions {
    public static User getLoggedUser(Session session){
        Long userID=null;
        if (session.get("userID")!=null)
            userID=Long.parseLong(session.get("userID"));
        User user=null;
        if(userID!=null){
            user = User.findById(userID);
        }
        return user;
    }

    private static final String
    imgurKey="081f3bb95260fbe572c2354724af2bac";

    public static Json uploadImage(Upload immagine) throws
    Exception {
```

```

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        baos.write(immagine.asBytes());

        String data = URLEncoder.encode("image", "UTF-8") + "=" +
        URLEncoder.encode(Base64.encodeBytes(baos.toByteArray()), "UTF-8");
        data += "&" + URLEncoder.encode("key", "UTF-8") + "=" +
        URLEncoder.encode(imgurKey, "UTF-8");

        URL url = new URL("http://api.imgur.com/2/upload.json");

        URLConnection conn = url.openConnection();
        conn.setDoOutput(true);
        OutputStreamWriter wr = new
        OutputStreamWriter(conn.getOutputStream());
        wr.write(data);
        wr.flush();

        BufferedReader br =new BufferedReader(new
        InputStreamReader(conn.getInputStream()));
        String inputLine;
        String a="";
        while ((inputLine = br.readLine()) != null)
            a=a+inputLine;
        br.close();

        Json json=null;
        try {
            JSONObject j = new JSONObject(a);
            j =
            j.getJSONObject("upload").getJSONObject("links");
            json = map().put("original",
            j.getString("original"))
                                .put("small_square",
            j.getString("small_square"))
                                .put("large_thumbnail",
            j.getString("large_thumbnail"))
                                .put("delete_page",
            j.getString("delete_page"));
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        return json;
    }

    public static Map<String,Double> geocode(String address)
    throws IOException, JSONException{
        URL url = new
        URL("http://maps.googleapis.com/maps/api/geocode/json?address=" +

```

```

URLDecoder.decode(address, "UTF-8")+"&sensor=false");

URLConnection conn = url.openConnection();

BufferedReader br =new BufferedReader(new
InputStreamReader(conn.getInputStream()));

String inputLine;
String a="";
while ((inputLine = br.readLine()) != null)
    a=a+inputLine;
br.close();
JSONObject j =new JSONObject(a);
Map<String,Double> map = new HashMap<String, Double>();

j=j.getJSONArray("results").getJSONObject(0);
j=j.getJSONObject("geometry");
j=j.getJSONObject("location");
map.put("latitude", j.getDouble("lat"));
map.put("longitude", j.getDouble("lng"));

return map;
}
}

```

---

*Lib/Facebook.java*

```

package lib;

import play.mvc.Router;

public class Facebook {
    // get these from your FB Dev App

    private static final String api_key =
"5bc3606c3e355c858133cdd9086ac14e";
    private static final String secret =
"419695d70a09f23dff8735026f7a22f4";
    private static final String client_id = "141083772629937";
    private static final String redirect_uri =
"http://kefacciamo.appspot.com/fbauth";/**/
    /*private static final String api_key =
"5bc3606c3e355c858133cdd9086ac14e";
    private static final String secret =
"43473f3efa37691661389dce5fb48064";
    private static final String client_id = "202159310993";
    private static final String redirect_uri =
"http://localhost:9000/fbauth";/**/

    // set this to your servlet URL for the authentication

```

servlet/filter

```
    /// set this to the list of extended permissions you want
    //private static final String[] perms = new String[]
{"publish_stream", "email"};

    public static String getAPIKey() {
        return api_key;
    }

    public static String getSaecret() {
        return secret;
    }

    public static String getLoginRedirectURL() {
        return "https://graph.facebook.com/oauth/authorize?
client_id=" +
            client_id + "&display=page&redirect_uri=" +
            redirect_uri+"&scope=publish_stream,email,user_location
,user_birthday";//+StringUtil.delimitObjectsToString(",", perms);
    }

    public static String getAuthURL(String authCode) {
        return "https://graph.facebook.com/oauth/access_token?
client_id=" +
            client_id+"&redirect_uri=" +
            redirect_uri+"&client_secret="+secret+"&code="+authCode
;
    }
}
```

---

*Controllers/Tags.java*

```
package controllers;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import lib.CommonFunctions;
import models.Event;
import models.Place;
import models.Tag;
import models.User;
import play.mvc.Controller;
import play.mvc.Router;
public class Tags extends Controller {
    public static void save(String nome){
        User user = CommonFunctions.getLoggedUser(session);
```

```

        try {
            new Tag(user,nome).insert();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            flash.error("Utente non trovato");
        }
        Application.index(null);
    }

    public static void getTags(String search){
        if(search.lastIndexOf(',')>0)
            search = search.substring(search.lastIndexOf(',')
+1);

        search = search.replaceAll(" ", "");
        List<Tag> l = Tag.filterByName(search);
        renderJSON(l);
    }

    public static void getByTag(String tag,String address){
        List<Event> events=null;
        Double lat=null,lng=null;
        if(address!=null){
            try{
                Map<String, Double> location =
CommonFunctions.geocode(address);
                lat=location.get("latitude");
                lng=location.get("longitude");
            }catch (Exception e) {
                e.printStackTrace();
            }
        }
        try {
            Tag t = Tag.findByName(tag, null);
            events = t.getEvents();
            if((lat!=null)&&(lng!=null)){
                List<Event> tempList = new ArrayList<Event>();
                for (Event event : events) {

                    event.place=Place.findById(event.place.id); //TODO Better
performance
                    if(event.place.isNear(lat,lng))
                        tempList.add(event);
                }
                events=tempList;
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

```

```

        User user=CommonFunctions.getLoggedInUser(session);
        List<Tag> tagcloud = Tag.getTagCloud();
        String actionVicino = "/tag/"+tag;
        renderTemplate("Events/index.html",
user,events,tagcloud,address,actionVicino);
    }

}

```

---

### *Controllers/Events.java*

```

package controllers;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Map;

import lib.CommonFunctions;
import models.Event;
import models.EventComment;
import models.Place;
import models.Tag;
import models.User;
import play.data.Upload;
import play.mvc.Before;
import play.mvc.Controller;
import siena.Json;
public class Events extends Controller {

    @Before
    static void checkAuthenticated() {
        String expires = session.get("expires");
        System.currentTimeMillis();
        System.out.println(expires);
    }

    public static void index(String address) {
        User user=CommonFunctions.getLoggedInUser(session);
        Double lat=null,lng=null;
        if(address!=null){
            try{
                Map<String, Double> location =
CommonFunctions.geocode(address);

```

```

        lat=location.get("latitude");
        lng=location.get("longitude");
    }catch (Exception e) {
        e.printStackTrace();
    }
}
List<Event> events= Event.getAllByPoints(lat,lng);
for (Event event : events) {
    event.user = User.findById(event.user.id);
}
List<Tag> tagcloud = Tag.getTagCloud();
render(user,events,tagcloud,address);
}

public static void create(Long id){

    User user = CommonFunctions.getLoggedInUser(session);
    List<Tag> tagcloud = Tag.getTagCloud();
    render(id,user,tagcloud);
}

public static void save(String placeDesc, String descrizione,
String dataInizio,String dataFine, String nome, Boolean
isRicorrente, Upload immagine, Long ricorrenza, String tags) throws
FileNotFoundException, IOException {
    User user=CommonFunctions.getLoggedInUser(session);
    Date dataInizi;
    Date dataFin;
    DateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");

    tags = tags.replaceAll(" ", "");
    String[] tagArray = tags.split(",");
    List<Tag> tagList= new ArrayList<Tag>();
    for (String string : tagArray) {
        try {
            tagList.add(Tag.findByName(string,session));
        } catch (Exception e) {}
    }
    Json imageLinks=null;
    try {
        imageLinks=CommonFunctions.uploadImage(immagine);
    } catch (Exception e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    try {
        dataInizi = formatter.parse(dataInizio);
        dataFin =formatter.parse(dataFine);

        Place place = Place.findByString(placeDesc);
        new

```



```

Event(user,place,nome,descrizione,imageLinks,dataInizi,dataFin,isRi
corrente,ricorrenza,tagList).insert();
    } catch (ParseException e) {
        e.printStackTrace();
        flash.error("Errore nel creare evento");
    }
    catch (Exception e) {
        e.printStackTrace();
        flash.error("Errore nel creare evento");
    }

    index(null);
}
public static void vote(Long id, Boolean isPositive, Boolean
redirectToEvent){
    Event event = Event.findById(id);
    User user = CommonFunctions.getLoggedInUser(session);
    try {
        event.addVote(user,isPositive);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        flash.error("Errore nel creare voto: " +
e.getMessage());
    }
    if(redirectToEvent)
        show(id);
    else
        index(null);
}

public static void show(Long id){
    Event event = Event.findById(id);
    int votes=event.points;
    event.user=User.findById(event.user.id);
    event.place=Place.findById(event.place.id);
    List<EventComment> comments = event.comments.fetch();
    for (EventComment comment : comments) {
        comment.user=User.findById(comment.user.id);
    }
    List<Tag> tags = new ArrayList<Tag>();
    Tag temp=null;
    for (Long idTag : event.getTags()) {
        temp=Tag.findById(idTag);
        if(temp!=null)
            tags.add(temp);
    }
    User user = CommonFunctions.getLoggedInUser(session);
    List<Tag> tagcloud = Tag.getTagCloud();
    render(event,votes,comments,user,tags,tagcloud);
}

```

```

        public static void addComment(Long Id, String comment){
            Event event = Event.findById(Id);
            User user = CommonFunctions.getLoggedInUser(session);
            try {
                new EventComment(user, event, comment, new
Date()).insert();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                flash.error("Errore nel creare commento");
            }
            show(Id);
        }
    }
}

```

---

### *Controllers/Places.java*

```

package controllers;

import java.io.IOException;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.google.appengine.repackaged.org.json.JSONException;

import lib.CommonFunctions;
import models.Place;
import models.PlaceComment;
import models.PlaceVote;
import models.Tag;
import models.User;
import play.data.Upload;
import play.mvc.Before;
import play.mvc.Controller;
import play.mvc.Router;
import siena.Json;

public class Places extends Controller {
    @Before
    static void checkAuthenticated() {
        String expires = session.get("expires");
        System.out.println(System.currentTimeMillis()/1000);
        System.out.println(expires);
    }
    public static void index(String address) {

        User user=CommonFunctions.getLoggedInUser(session);
    }
}

```

```

        Double lat=null,lng=null;
        if(address!=null){
            try{
                Map<String, Double> location =
CommonFunctions.geocode(address);
                lat=location.get("latitude");
                lng=location.get("longitude");
            }catch (Exception e) {
                e.printStackTrace();
            }
        }
        List<Place> places = Place.getAll(lat,lng);
        for (Place place : places) {
            place.user=User.findById(place.user.id);
        }
        List<Tag> tagcloud = Tag.getTagCloud();
        String actionVicino = Router.getFullUrl("Places.index");
        render(user,places,tagcloud,address,actionVicino);
    }

    public static void create(){
        User user = CommonFunctions.getLoggedUser(session);
        List<Tag> tagcloud = Tag.getTagCloud();
        render(user,tagcloud);
    }

    public static void show(Long id){
        Place place = Place.findById(id);
        place.user=User.findById(place.user.id);
        int votes=place.points;
        List<PlaceComment> comments = place.comments.fetch();
        for (PlaceComment comment : comments) {
            comment.user=User.findById(comment.user.id);
        }
        User user = CommonFunctions.getLoggedUser(session);
        List<Tag> tagcloud = Tag.getTagCloud();
        render(place,votes,comments,user,tagcloud);
    }

    public static void save(String nome, String descrizione,
String indirizzo, Upload immagine){
        User user = CommonFunctions.getLoggedUser(session);
        Json imageLinks=null;
        try {
            imageLinks = CommonFunctions.uploadImage(immagine);
        } catch (Exception e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        Map<String, Double> location;
        try {

```

```

        location = CommonFunctions.geocode(indirizzo);

    } catch (IOException e1) {
        // TODO Auto-generated catch block
        location = new HashMap<String, Double>();
        e1.printStackTrace();
    } catch (JSONException e1) {
        // TODO Auto-generated catch block
        location = new HashMap<String, Double>();
        e1.printStackTrace();
    }
    try {
        new
Place(user,nome,descrizione,imageLinks,indirizzo,location.get("lati
tude"),location.get("longitude")).insert();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        flash.error("Utente non trovato");
    }
    Application.index(null);
}

public static void addComment(Long Id, String comment){
    Place place= Place.findById(Id);
    User user = CommonFunctions.getLoggedUser(session);

    try {
        new PlaceComment(user, place, comment, new
Date()).insert();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        flash.error("Utente o luogo non trovato");
    }

    show(Id);
}

public static void vote(Long id, Boolean isPositive, Boolean
redirectToPlace){
    Place place = Place.findById(id);
    User user = CommonFunctions.getLoggedUser(session);
    try {
        new PlaceVote(user, place, isPositive).insert();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        flash.error("Utente o luogo non trovato");
    }
    place.update();
}

```

```

        if(redirectToPlace)
            show(id);
        else
            index(null);
    }
    public static void getPlaces(String search){
        List<Place> l = Place.filterByName(search);
        renderJSON(l);
    }
}

```

---

### *Controllers/Photos.java*

```

package controllers;

import java.util.Date;
import java.util.List;

import lib.CommonFunctions;
import models.Event;
import models.Photo;
import models.Presence;
import models.User;
import play.data.Upload;
import play.mvc.Controller;
import siena.Json;

public class Photos extends Controller {
    public static void save(Long eventId, Upload photo) throws
    Exception{
        User user = CommonFunctions.getLoggedUser(session);
        Event event = Event.findById(eventId);
        Json json = CommonFunctions.uploadImage(photo);
        try {
            new Photo(user,event,new Date(), json).insert();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            flash.error("Impossibile creare foto");
        }
        Photos.showEvent(eventId);
    }

    public static void showEvent(Long eventId) {
        Event event = Event.findById(eventId);
        User user = CommonFunctions.getLoggedUser(session);
        List<Photo> photos = event.photos.fetch();
        render(photos,user,event);
    }
}

```

```

        public static void show(Long photoId){
            User user = CommonFunctions.getLoggedInUser(session);
            Photo photo = Photo.findById(photoId);
            photo.event=Event.findById(photo.event.id);
            List<Presence> presences = photo.presences.fetch();

            for (Presence presence : presences) {
                presence.user=User.findById(presence.user.id);
            }
            render(user,photo,presences);
        }
        public static void addPresence(Long photoId, Integer x,
Integer y){
            User user = CommonFunctions.getLoggedInUser(session);
            Photo photo = Photo.findById(photoId);
            new Presence(user, photo, x, y).insert();

        }
    }
}

```

---

*Controllers/Application.java*

```

package controllers;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.List;

import lib.CommonFunctions;
import models.Action;
import models.Event;
import models.EventComment;
import models.Medal;
import models.Photo;
import models.Place;
import models.PlaceComment;
import models.Presence;
import models.Tag;
import models.TipoAzione;
import models.TipoMedaglia;
import models.User;
import play.Logger;
import play.mvc.Controller;

import com.restfb.DefaultFacebookClient;
import com.restfb.FacebookClient;
public class Application extends Controller {

```

```

/**
 * RestFB Graph API client.
 */
private static FacebookClient facebookClient=null;

public static void index(String address) {
Events.index(address);
}

public static void search(String s){
User user=CommonFunctions.getLoggedUser(session);
List<Tag> tagcloud = Tag.getTagCloud();
render(user,s,tagcloud);
}

public static void DoLogin(String service){
com.restfb.types.User user = facebookClient.fetchObject("me",
com.restfb.types.User.class);
User u = User.findByEmail(user.getEmail());
if(u==null){
try {
u= new User(user.getEmail(), user.getName(),
user.getId(), service, null);
u.insert();
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
session.put("userID", u.id);
index(null);
}
public static void doLogout(){
session.remove("userID");
index(null);
}
public static void fbauth(String code){
if (code!=null) {
String authURL = lib.Facebook.getAuthURL(code);
try {
URL url = new URL(authURL);
String result = readURL(url);
String accessToken = null;
Integer expires = null;
String[] pairs = result.split("&");
for (String pair : pairs) {
String[] kv = pair.split("=");
if (kv.length != 2) {
throw new RuntimeException("Unexpected auth
response");
} else {

```

```

        if (kv[0].equals("access_token")) {
            accessToken = kv[1];
        }
        if (kv[0].equals("expires")) {
            expires = Integer.valueOf(kv[1]);
        }
    }
    if (accessToken != null && expires != null) {
        facebookClient = new
DefaultFacebookClient(accessToken);
        session.put("accessToken", accessToken);
        session.put("expires",
expires+System.currentTimeMillis()/1000);
        DoLogin("Facebook");
        renderTemplate("Events/index.html");
    } else {
        Logger.error("Access token and expires not
found");
        renderTemplate("Events/index.html");
    }
} catch (IOException e) {
    Logger.error(e.toString());
    renderTemplate("Events/index.html");
}
}

}
private static String readURL(URL url) throws IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    InputStream is = url.openStream();
    int r;
    while ((r = is.read()) != -1) {
        baos.write(r);
    }
    return new String(baos.toByteArray());
}

public static void profile(Long id){
    User user=CommonFunctions.getLoggedInUser(session);
    User userProfile = User.findById(id);
    List<Event> events = userProfile.events.order("-
id").limit(10).fetch();
    List<Place> places = userProfile.places.order("-
id").limit(10).fetch();
    List<EventComment> eventComments =
userProfile.eventComments.order("-data").limit(10).fetch();
    for (EventComment eventComment : eventComments) {

        eventComment.event=Event.findById(eventComment.event.id);
    }
}

```



```

        List<PlaceComment> placeComments=
        userProfile.placeComments.order("-data").limit(10).fetch();
        for (PlaceComment placeComment : placeComments) {
            placeComment.place=Place.findById(placeComment.place.id);
        }
        List<Action> actions = userProfile.actions.fetch();
        for (Action action : actions) {
            action.actionType =
        TipoAzione.findById(action.actionType.id);
            action.user = User.findById(action.user.id);
        }
        List<Tag> tagcloud = Tag.getTagCloud();
        Boolean isFriend = user.isFriend(userProfile);
        List<Medal> medals = userProfile.medals.fetch();
        for (Medal medal : medals) {
            medal.medalType =
        TipoMedaglia.findById(medal.medalType.id);
        }
        List<Presence> presences = userProfile.presences.fetch();
        for (Presence presence : presences) {
            presence.photo = Photo.findById(presence.photo.id);
        }

        render(user,userProfile,tagcloud,events,places,eventComments,placeC
        omments,isFriend,actions,medals,presences);
    }
    public static void addFriend(Long id){
        User user=CommonFunctions.getLoggedInUser(session);
        User userFriend = User.findById(id);
        user.addFriend(userFriend);
        if(userFriend!=null)
            profile(userFriend.id);
        else
            profile(user.id);
    }
    public static void removeFriend(Long id){
        User user=CommonFunctions.getLoggedInUser(session);
        User userFriend = User.findById(id);
        user.removeFriend(userFriend);
        if(userFriend!=null)
            profile(userFriend.id);
        else
            profile(user.id);
    }
    public static void test(){
    }
}

```

views/main.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>#{get 'title' /}</title>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <link rel="stylesheet" type="text/css" media="screen"
href="@{'/public/stylesheets/style.css'}">
    <link rel="stylesheet" type="text/css" media="screen"
href="@{'/public/stylesheets/color-green.css'}">
    <link rel="stylesheet/less" type="text/css"
href="@{'/public/stylesheets/main.less'}">
    <link type="text/css"
href="@{'/public/stylesheets/jquery-ui-1.8.13.custom.css'}"
rel="Stylesheet" />

    <link rel="shortcut icon" type="image/png"
href="@{'/public/images/favicon.png'}">

    <script src="http://lesscss.googlecode.com/files/less-
1.1.3.min.js" type="text/javascript"></script>
    <script src="http://code.jquery.com/jquery-
latest.js"></script>

    <script src="@{'public/javascripts/jquery.mystique.js'}"
type="text/javascript" charset="utf-8"></script>
    <script type="text/javascript"
src="@{'public/javascripts/jquery-ui-
1.8.13.custom.min.js'}"></script>
    <script src="@{'public/javascripts/jquery.tagcloud-
2.js'}" type="text/javascript" charset="utf-8"></script>
    <script
src="http://yui.yahooapis.com/3.3.0/build/yui/yui-min.js"
charset="utf-8"></script>
  </head>
  <body class="home col-2-right fluid">
    <div id="page">
      <!-- header -->
      <div class="page-content header-wrapper">
        <div id="header" class="bubbleTrigger">
          <div id="site-title" class="clearfix">
            <h1 id="logo"><a
href="@{Application.index()}">InformaMi</a></h1>

            <p class="headline">dove trovare gli eventi
migliori</p>
          </div>
```

```

        <div class="shadow-left">
            <div class="shadow-right clearfix">
                <p class="nav-extra"><a
href="http://twitter.com/mauriziopez" class="nav-extra twitter"
title="Follow me on Twitter"><span>Follow me on
Twitter</span></a><a href="#" class="nav-extra rss" title="RSS
Feeds"><span>RSS Feeds</span></a></p>

                <!-- main navi -->
                <ul id="navigation" class="clearfix">
                    <li class="active home"><a class="home
active fadeThis" href="@{Application.index()}"><span
class="title">Home</span></a></li>
                    <li>
                        <a class="fadeThis"
href="@{Places.index()}"><span class="title">Locali</span></a>
                    </li>
                    <li>
                        <a class="fadeThis"
href="@{Application.profile(user.id)}"><span
class="title">Profilo</span></a>
                    </li>

                    <li><a class="fadeThis"
href="@{Application.doLogout()}"><span class="title">Log
out</span></a></li>

                    </li>
                </ul>
                <!-- /main navi -->

            </div>
        </div>
    </div>
</div>
<!-- /header -->
<!-- left+right bottom shadow -->
<div class="shadow-left page-content main-wrapper">
    <div class="shadow-right">
        <!-- main content: primary + sidebar(s) -->
        <div id="main">
            <div id="main-inside" class="clearfix">
                <!-- primary content -->
                <div id="primary-content">
                    <div class="blocks">
                        <!-- doLayout -->
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

```

```

#{include 'sidebar.html' /}

</div>
</div>
<!-- /main content -->

<div id="error">
    ${flash.error}
</div>
<!-- footer -->
<div id="footer">
    <div class="page-content">
        <div id="copyright">
            Mystique theme by <a
href="http://digitalnature.ro">digitalnature</a><br />
            <a class="rss-subscribe" href="#"
title="RSS Feeds">RSS Feeds</a> <a class="valid-xhtml"
href="http://validator.w3.org/check?uri=referer" title="Valid
XHTML">XHTML 1.1</a> <a id="goTop" class="js-link"
name="goTop">Top</a> <!--[if lte IE 6]> <script
type="text/javascript"> isIE6 = true; isIE = true; </script> <![endif]-->

            <!--[if gte IE 7]> <script
type="text/javascript"> isIE = true; </script> <![endif]-->
        </div>
    </div>
</div>
<!-- /footer -->
</div>
<!-- /shadow -->

<!-- page controls -->
<div id="pageControls"></div>
<!-- /page controls -->
</div>
<!-- /page -->
</body>
</html>

```

---

views/sidebar.html

```

<!-- sidebar -->
<div id="sidebar">

    <ul class="blocks">

        <!-- search form -->
        <li class="block">

```

```

        <div class="search-form">
            <form method="get" id="searchform"
action="@{Application.search()}" class="clearfix"
name="searchform">
                <fieldset>
                    <div id="searchfield">
                        <input type="text" name="s"
id="searchbox" class="text clearField #{if
s}clearFieldActive#{/if}" value="#{ifnot s}Search#{/ifnot}#{else}$
{s}#{/else}" />
                            </div><input type="submit" value=""
class="submit" />
                    </fieldset>
                </form>
            </div>
        </li>
        <!-- /search form -->

        #{ifnot user}
        <!-- login -->
        <li class="block">
            <div class="block-login clearfix">
                <h3 class="title"><span>Login
utente</span></h3>
                <div class="block-div"></div>
                <div class="block-div-arrow"></div>

                <a href="$
{lib.Facebook.getLoginRedirectURL()}"></a>

            </div>
        </li>
        <!-- /login -->
        #{/ifnot}
        #{if user}
        <!-- logged in -->
        <li class="block">
            <div class="block-login clearfix">
                <h3 class="title"><span>${
{user.nome}</span></h3>
                <div class="block-div"></div>
                <div class="block-div-arrow"></div>
                <ul>
                    <li><a href="@{Events.create()}"
class="button"><span>Crea evento</span></a></li>

```

```

        <li><a href="@{Places.create()}"
class="button"><span>Crea locale</span></a></li>
    </ul>
</div>
</li>
<!-- /logged in -->
    %{
        List<models.Action> actios =
user.getFriendsActions();
        List<models.Action> actions = new
ArrayList<models.Action>();
        for (models.Action action : actios) {
            if(action!=null){
                action.actionType =
models.TipoAzione.findById(action.actionType.id);
                action.user =
models.User.findById(action.user.id);
                actions.add(action);
            }
        }
    }%
    #{if actions.size()}
    <ul>
        #{list items:actions, as:'action'}
        <li><a href="$${action.URL}">$
{action}</a></li>
        #{/list}
    </ul>
    #{/if}
    #{/if}

    <li class="block">
        <div class="block-login clearfix">
            <h3 class="title"><span>Vicino a...</span></h3>
            <div class="block-div"></div>
            <div class="block-div-arrow"></div>
            <ul>
                <form action="#{if actionVicino}$
{actionVicino}#{/if}#{else}@{Application.index()}#{/else}">
                    <input type="text" value="$${address}"
name="address" style="min-width: 150px;width:60%;">
                    <input type="submit" value="Cerca">
                </form>
            </ul>
        </div>
    </li>

    #{if tagcloud}
    <script type="text/javascript">
        jQuery(document).ready(function() {
            var tags = [

```

```

        #{list tagcloud, as:'tag'}
        {tag: "${tag.nome}", count: $
{tag.idEvents.size() }, href:"@{Tags.getByTag(tag.nome) }"},
        #{/list}
        {tag:" ", count:0}
    ];
    jQuery("#tagcloud").tagCloud(tags);
});
</script>
<!-- tagcloud -->
<li class="block">
    <div class="block-login clearfix" style="word-
spacing: 1em;line-height: 3em;">
        <h3 class="title"><span>Tag cloud</span></h3>
        <div class="block-div"></div>
        <div class="block-div-arrow"></div>
        <div id="tagcloud">
            <ul>
                #{list tagcloud, as:'tag'}
                <li>
                    <a
href="@{Tags.getByTag(tag.nome) }">${tag.nome}</a>
                </li>
                #{/list}
            </ul>
        </div>

    </div>
</li>
<!-- /tagcloud -->
#{/if}

</ul>

</div>
<!-- sidebar -->

```

---

views/comments.html

```

<div class="section clearfix" id="section-comments">
    <div id="comments-wrap">
        <div class="clearfix">
            <ul id="comments" class="comments">
                #{list items:comments, as:'comment'}
                <!-- comment entry -->
                <li class="comment byuser
bypostauthor depth-1 withAvatars" id="comment-349">
                    <div class="comment-head comment
byuser bypostauthor depth-1 withAvatars">

```

```

<div class="avatar-
box"></div>

<div class="author">
    <span class="by">
written by <a class="comment-author" id="comment-author-349"
href="@{Application.profile(comment.user?.id)}" rel="nofollow"
name="comment-author-349">${comment.user?.nome}</a></span><br />
    $
{comment.data.since()}

</div>
<div class="controls
bubble">

</div>
</div>
<div class="comment-body
clearfix" id="comment-body-349">

<div class="comment-text">
    <p>${
{comment.testo}</p>

</div>
<a id="comment-reply-349"
name="comment-reply-349"></a>

</div>
</li>
<!-- /comment entry -->
#{/list}
</ul>
</div>
</div>

<!-- comment form -->
<div class="comment-form clearfix" id="respond">
    #{form action}
        <input type="hidden" name="Id" value="#{get
'id' /}" />

        <textarea name="comment" rows="5"></textarea>
        <input type="submit" value="Aggiungi
commento"/>
    #{/form}
</div>
<!-- /comment form -->
</div>

```

---

```
views/Application/profile.html
```

```

#{extends 'main.html' /}
#{set title: 'Profilo di ' + userProfile.nome/}

```



```

<!-- profile -->
<div class="post clearfix">
  <h1 class="title">${userProfile.nome}</h1>
  <div class="post-content clearfix">
    <div style="float:left; width:65%;margin:1%;">
      <div>
        <h2>Ultimi eventi pubblicati</h2>
        <ul>
          #{list items:events, as:'event'}
          <li><a href="@{Events.show(event.id)}">${
{event.nome}</a> </li>
          #{/list}
        </ul>
      </div>
      <div>
        <h2>Ultimi locali pubblicati</h2>
        <ul>
          #{list items:places, as:'place'}
          <li><a href="@{Places.show(place.id)}">${
{place.nome}</a> </li>
          #{/list}
        </ul>
      </div>
      <div>
        <h2>Ultimi commenti ad eventi</h2>
        <ul>
          #{list items:eventComments, as:'eComment'}
          <li>${eComment.testo} - <a
href="@{Events.show(eComment.event?.id)}">${
{eComment.event?.nome}</a> </li>
          #{/list}
        </ul>
      </div>
      <div>
        <h2>Ultimi commenti a locali</h2>
        <ul>
          #{list items:placeComments, as:'pComment'}
          <li>${pComment.testo} - <a
href="@{Places.show(pComment.place?.id)}">${
{pComment.place?.nome}</a></li>
          #{/list}
        </ul>
      </div>
      <div>
        <h2>Ultime azioni</h2>
        <ul>
          #{list items:actions, as:'action'}
          <li><a href="$${action.URL}">${
{action.toString()}</a></li>
          #{/list}
        </ul>
      </div>
    </div>
  </div>
</div>

```

```

        </div>
        <div>
            <h2>Foto in cui compare</h2>
            #{list items:presences, as:'presence'}
                <a
href="@{Photos.show(presence.photo.id)}" style="padding:
10px;"></a>
                #{/list}
        </div>

    </div>
    <div style="float:right; margin:1%;width:30%">
        <br>
        #{ifnot user.equals(userProfile)}#{if isFriend}<a
href="@{Application.removeFriend(userProfile.id)}">Rimuovi dai tuoi
amici</a>#{/if}#{else}<a
href="@{Application.addFriend(userProfile.id)}">Aggiungi ai tuoi
amici</a>#{/else}#{/ifnot}<br>
        Medaglie:#{list items:medals, as:'medal'} ${medal}
    #{/list}
    </div>
</div>
<!-- /profile -->

```

---

views/Application/search.html

```

#{extends 'main.html' /}
#{set title:'Home' /}

```

```

<div id="cse" style="width: 100%;">Loading</div>
<script src="http://www.google.com/jsapi"
type="text/javascript"></script>
<script type="text/javascript">
    function parseQueryFromUrl () {
        var queryParamName = "s";
        var search = window.location.search.substr(1);
        var parts = search.split('&');
        for (var i = 0; i < parts.length; i++) {
            var keyvaluepair = parts[i].split('=');
            if (decodeURIComponent(keyvaluepair[0]) == queryParamName) {
                return decodeURIComponent(keyvaluepair[1].replace(/\+/g, '
')));
            }
        }
    }

```

```

    }
  }
  return '';
}
google.load('search', '1', {language : 'it', style :
google.loader.themes.GREENSKY});
google.setOnLoadCallback(function() {
  var customSearchControl = new
google.search.CustomSearchControl('015980383533509376315:a3rvl_tkvce
');
  customSearchControl.setResultsetSize(google.search.Search.FILTE
RED_CSE_RESULTSET);
  var options = new google.search.DrawOptions();
  options.enableSearchResultsOnly();
  customSearchControl.draw('cse', options);
  var queryFromUrl = parseQueryFromUrl();
  if (queryFromUrl) {
    customSearchControl.execute(queryFromUrl);
  }
}, true);
</script>

```

---

*views/Events/create.html*

```

#{extends 'main.html' /}
#{set title:'Crea locale: '/}

<script type="text/javascript">
<!--
YUI().use('autocomplete', 'autocomplete-highlighters', function (Y)
{

  // Add the yui3-skin-sam class to the body so the default
  // AutoComplete widget skin will be applied.
  Y.one('body').addClass('yui3-skin-sam');

  // The following examples demonstrate some of the different
  // result sources AutoComplete supports. You only need to
  // pick one, you don't need them all. Assume the '#ac-input'
  // element id used in this example refers to an <input>
  // element on the page.

  // XHR URL source (no callback). Leave the {query}
placeholder
  // as is; AutoComplete will replace it automatically.
  Y.one('#place').plug(Y.Plugin.AutoComplete, {
    resultHighlighter: 'phraseMatch',
    resultTextLocator: function (result) {

```

```

        return result.nome+':' + result.indirizzo;
    },
    source: '@{Places.getPlaces()}'+'?search={query}'
});
Y.one('#tags').plug(Y.Plugin.AutoComplete, {
    resultHighlighter: 'phraseMatch',
    queryDelimiter: ',',
    resultTextLocator: function (result) {
        return result.nome;
    },
    source: '@{Tags.getTags()}'+'?search={query}'
});

});
$(document).ready(function () {
    $('#dataInizio').datepicker({dateFormat:'dd/mm/yy'});
    $('#dataFine').datepicker({dateFormat:'dd/mm/yy'});
});

//-->
</script>

#{form @Events.save(), enctype:'multipart/form-data'}

<h3>Nome</h3>
<input type="text" class="text" name="nome" />

<h3>Descrizione</h3>
<textarea class="text" name="descrizione"></textarea>

<h3>Locale</h3>
<input type="text" class="text" name="placeDesc"
id="place" value="{id}"/>

<span class="wrapper">
    <span class="cell">
        <h3>Data inizio</h3>
        <input type="text" name="dataInizio"
id="dataInizio"/>
    </span>
    <span class="cell">
        <h3>Data fine</h3>
        <input type="text" name="dataFine" id="dataFine"/>
    </span>
</span>

<h3 style="display: none">È ricorrente?</h3>
<input style="display: none" type="checkbox">

```

```

name="isRicorrente"/>

    <h3 style="display: none">Ogni quanto?</h3>
    <input style="display: none" type="text" class="text"
name="ricorrenza"/>

    <h3>Carica una immagine</h3>
    <input type="file" class="text" name="immagine"/>

    <h3>Tags</h3>
    <input type="text" class="text" name="tags" id="tags"/>

    <p>
        <input type="submit" value="Aggiungi evento" />
    </p>
#{/form}

```

---

views/Events/index.html

```

#{extends 'main.html' /}
#{set title:'Home' /}

#{list items:events, as:'event'}
<div class="post" id="xnews-#{event.id}">
    <a class="post-thumb alignleft"
href="@{Events.show(event.id)}"></a>
    <h2 class="title">
        <a href="@{Events.show(event.id)}">#{event.nome}</a>
    </h2>

    <div class="post-date">
        <p> #{event.dataInizio.since()}</p>
    </div>

    <div class="post-info clearfix with-thumbs">
        <p class="author alignleft">
            Posted by <a
href="@{Application.profile(event.user.id)}">#{
event.user?.nome}</a>
        </p>
        <div class="votes alignright">
            <span class="votenummer" id="xvotes-#{event.id}">#{
event.points}</span>

            <div class="vote-box" id="xvote-#{event.id}">
                <a href="@{Events.vote(event.id,true,false)}"
class="vote up"><span>Vota</span></a>

```

```

        <a id="xreport-#{event.id}"
href="@{Events.vote(event.id, false, false)}" class="vote
down"><span>Sotterra</span></a>
    </div>
</div>
<div class="clear"></div>
</div>

<div class="post-content clearfix">
    #{if event.descrizione.size()<100}
    <p>#{event.descrizione}</p>
    #{/if}
    #{else}
    <p>#{event.descrizione.substring(0,100)}...</p>
    #{/else}
    <a href="@{Events.show(event.id)}">&raquo; Leggi
tutto</a>
</div>

</div>
#{/list}

```

---

views/Events/show.html

```

#{set action: @Events.addComment()/}
#{set id: event.id/}
#{set title: event.nome + ' a ' + event.place?.indirizzo/}
#{extends 'main.html' /}

<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>

<script type="text/javascript">
$(document).ready(function () {
    initialize();
});

var geocoder;
var map;

function initialize() {
    geocoder = new google.maps.Geocoder();
    var latlng = new google.maps.LatLng(-34.397, 150.644);
    var myOptions = {
        zoom: 8,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    }
}

```

```

        map = new
google.maps.Map(document.getElementById("map_canvas"), myOptions);
        codeAddress('${event.place?.indirizzo}');
    }

    function codeAddress(address) {
        geocoder.geocode( { 'address': address}, function(results,
status) {
            if (status == google.maps.GeocoderStatus.OK) {
                map.setCenter(results[0].geometry.location);
                map.fitBounds(results[0].geometry.bounds);
                var marker = new google.maps.Marker();
                marker.setMap(map);
                marker.setPosition(results[0].geometry.location);
                $('#map_canvas').show();
            } else {
            }
        });
    }
</script>

<!-- post -->
<div class="post clearfix">
    <a class="post-thumb alignleft" href="#"></a>
    <h2 class="title"><a href="#" rel="bookmark">${
event.nome}</a></h2>
    <div class="post-date">
        <p class="day">${event.dataInizio.since()}</p>
    </div>
    <div class="post-info clearfix with-thumbs">
        <p class="author alignleft">Posted by <a
href="@{Application.profile(event.user.id)}">${
event.user.nome}</a></p>
        <div class="votes alignright">
            <span class="votenummer" id="xvotes-0">${votes}
</span>
            <div class="vote-box" id="xvote-0">
                <a href="@{Events.vote(event.id,true,true)}"
class="vote up"><span>Vota</span></a>
                <a id="xreport-0"
href="@{Events.vote(event.id,false,true)}" class="vote
down"><span>Sotterra</span></a>
            </div>
        </div>
    </div>
</div>

```

```

    <div class="post-content clearfix">
        <p style="float:left;width:65%;margin:1%;"> $
{event.descrizione}</p>
        <div style="float:right; margin:1%;width:30%">
            <br/>
            <a href="@{Photos.showEvent(event.id)}"
class="button"><span>Altre foto</span></a>
        </div>
    </div>
    <div class="post-content clearfix">
        <h3>Indirizzo</h3>
        <p> ${event.place?.indirizzo}</p>
    </div>

    <div id="map_canvas" style="height: 300px;"></div>

    <div class="post-tags">
        #{list items:tags, as:'tag'}
            <a href="@{Tags.getByTag(tag.nome)}">${tag.nome}</a>
        #{/list}
    </div>
</div>
<!-- /post -->

<!-- tabbed content -->
<div class="tabbed-content post-tabs clearfix" id="post-tabs">
    <!-- tab navigation (items must be in reverse order because of
the tab-design) -->
    <div class="tabs-wrap clearfix">
        <ul class="tabs">
            <li class="comments"><a href="#section-
comments"><span>Comments (${comments.size()})</span></a></li>
        </ul>
    </div>
    <!-- /tab nav -->

    <!-- tab sections -->
    <div class="sections">

        <!-- comments -->
        #{include 'comments.html' /}
        <!-- /comments -->

        <!-- trackbacks -->
        <div class="section" id="section-trackbacks">
            <h6 class="title">No trackbacks yet.</h6>
        </div>
        <!-- /trackbacks -->
    </div>

```



```

        <!-- /tab sections -->
    </div>
    <!-- /tabbed content -->

```

---

views/Photos/show.html

```

#{set title: 'Foto dell\'evento ' + photo.event.nome/}
#{extends 'main.html' /}
<script type="text/javascript">
    var showUserAction = #{jsAction
@Photos.addPresence(':idPhoto','x','y')/}
    function point_it(event){
        pos_x = event.offsetX?(event.offsetX):event.pageX-
document.getElementById("image").offsetLeft;
        pos_y = event.offsetY?(event.offsetY):event.pageY-
document.getElementById("image").offsetTop;
        var url = showUserAction({idPhoto: '$
{photo.id}',x:pos_x,y:pos_y});
        jQuery.get(url,function() {
            alert('Tag effettuato');
        });
    }
</script>
<div >

    <h3 >clicca la foto per taggarti in quel punto</h3>
    #{if presences.size()}<p> in questa foto sono presenti: #{list
items:presences, as:'presence'} <a
href="@{Application.profile(presence.user.id)}">$
{presence.user?.nome}</a> #{/list}</p>#{/if}

</div>

```

---

views/Photos/showEvent.html

```

#{extends 'main.html' /}
#{set title:'Home' /}

<h1><a href="@{Events.show(event.id)}">${event.nome}</a></h1>

#{list items:photos, as:'photo'}
    <a href="@{Photos.show(photo.id)}" style="padding:
10px;"></a>
#{/list}

```

```
<br><br><br>
```

```
<h3>Aggiungi altre foto all'evento</h3>
```

```
{form @Photos.save(), enctype:'multipart/form-data'}
  <label>Immagine</label>
  <input type="file" name="photo">
  <input type="hidden" name="eventId" value="{event.id}">
  <input type="submit" value="Carica">
{/form}
```

---

views/Places/create.html

```
{extends 'main.html' /}
{set title:'Crea evento: '/}
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>

<script type="text/javascript">
  jQuery(document).ready(function () {
    initialize();
  });

  var geocoder;
  var map;
  var marker = new google.maps.Marker();
  function initialize() {
    geocoder = new google.maps.Geocoder();
    var latlng = new google.maps.LatLng(-34.397, 150.644);
    var myOptions = {
      zoom: 8,
      center: latlng,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    }
    map = new
    google.maps.Map(document.getElementById("map_canvas"), myOptions);
    codeAddress('Trieste');
  }

  function codeAddress(address) {
    geocoder.geocode( { 'address': address }, function(results,
status) {
      if (status == google.maps.GeocoderStatus.OK) {
        map.setCenter(results[0].geometry.location);
        map.fitBounds(results[0].geometry.bounds);
        marker.setMap(map);
        marker.setPosition(results[0].geometry.location);
        $('#map_canvas').show();
      } else {
      }
    }
  }
}
```

```

    });
  }
</script>

#{form @Places.save(), enctype:'multipart/form-data'}
  <p>
    <h3>Titolo</h3>
    <input type="text" class="text" name="nome" />
  </p>
  <p>
    <h3>Descrizione</h3>
    <textarea name="descrizione" class="text"></textarea>
  </p>
  <p>
    <h3>Indirizzo</h3>
    <input type="text" name="indirizzo" id="indirizzo"
class="text"
onchange="codeAddress(document.getElementById('indirizzo').value);
"></input>
  </p>
  <div id="map_canvas" style="height: 300px; width:
90%;"></div>
  <p>
    <h3>Carica una immagine</h3>
    <input type="file" name="immagine" class="text"></input>
  </p>
  <p>
    <input type="submit" value="Aggiungi Locale" />
  </p>
#{/form}

```

---

```
views/Places/index.html
```

```

#{extends 'main.html' /}
#{set title:'Home' /}

```

```

#{list items:places, as:'place'}
<div class="post" id="xnews-#{place.id}">

  <h2 class="title">
    <a href="@{Places.show(place.id)}">#{place.nome}</a>
  </h2>

  <div class="post-info">
    <p class="author alignleft">
      Created by <a
href="@{Application.profile(place.user.id)}">#{

```

```

{place.user?.nome}</a>
    </p>
    <div class="votes alignright">
        <span class="votenumner" id="xvotes-#{place.id}">${
{place.points}</span>

        <div class="vote-box" id="xvote-#{place.id}">
            <a href="@{Places.vote(place.id,true,false)}"
class="vote up"><span>Vota</span></a>
            <a id="xreport-#{place.id}"
href="@{Places.vote(place.id,false,false)}" class="vote
down"><span>Sotterra</span></a>
        </div>
    </div>
    <div class="clear"></div>
</div>

<div class="post-content clearfix">
    #{if place.descrizione.size()<100}
    <p>#{place.descrizione}</p>
    #{/if}
    #{else}
    <p>#{place.descrizione.substring(0,100)}...</p>
    #{/else}
    <a href="@{Places.show(place.id)}">&raquo; Leggi
tutto</a>
</div>

</div>
#{/list}

```

---

views/Places/show.html

```

#{extends 'main.html' /}
#{set title: place.nome + ' a ' + place.indirizzo/}
#{set action: @Places.addComment()/}
#{set id: place.id/}
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>

<script type="text/javascript">
    $(document).ready(function () {
        initialize();
    });

    var geocoder;
    var map;

```

```

function initialize() {
    geocoder = new google.maps.Geocoder();
    var latlng = new google.maps.LatLng(-34.397, 150.644);
    var myOptions = {
        zoom: 8,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    }
    map = new
google.maps.Map(document.getElementById("map_canvas"), myOptions);
    codeAddress('${place.indirizzo}');
}

function codeAddress(address) {
    geocoder.geocode( { 'address': address}, function(results,
status) {
        if (status == google.maps.GeocoderStatus.OK) {
            map.setCenter(results[0].geometry.location);
            map.fitBounds(results[0].geometry.bounds);
            var marker = new google.maps.Marker();
            marker.setMap(map);
            marker.setPosition(results[0].geometry.location);
            $('#map_canvas').show();
        } else {
        }
    });
}
</script>
<!-- place -->
<div class="post clearfix">
    <a class="post-thumb alignleft" href="#"></a>
    <h2 class="title"><a href="#" rel="bookmark">${
{place.name}}</a></h2>
    <div class="post-date">
        <p class="day"></p>
    </div>
    <div class="post-info clearfix with-thumbs">
        <p class="author alignleft">Posted by <a
href="@{Application.profile(place.user.id)}">${
{place.user?.name}}</a></p>
        <div class="votes alignright">
            <span class="votenumner" id="xvotes-0">${votes}
</span>
            <div class="vote-box" id="xvote-0">
                <a href="@{Places.vote(place.id,true,true)}"
class="vote up"><span>Vota</span></a>
                <a id="xreport-0"

```

```

href="@{Places.vote(place.id,false,true)}" class="vote
down"><span>Sotterra</span></a>
    </div>
</div>
</div>

<div class="post-content clearfix">
    <p style="float:left; width:65%;margin:1%;"> $
{place.descrizione}</p>
    <div style="float:right; margin:1%;width:30%">
        
    </div>
</div>
<div class="post-content clearfix">
    <p> ${place.indirizzo}</p>
</div>
<div id="map_canvas" style="height: 300px;"></div>

</div>
<!-- /place -->

<!-- tabbed content -->
<div class="tabbed-content post-tabs clearfix" id="post-tabs">
    <!-- tab navigation (items must be in reverse order because of
the tab-design) -->
    <div class="tabs-wrap clearfix">
        <ul class="tabs">
            <li class="comments"><a href="#section-
comments"><span>Commenti (${comments.size()})</span></a></li>
        </ul>
    </div>
    <!-- /tab nav -->

    <!-- tab sections -->
    <div class="sections">

        <!-- comments -->
        #{include 'comments.html' /}
        <!-- /comments -->

        <!-- trackbacks -->
        <div class="section" id="section-trackbacks">
            <h6 class="title">No trackbacks yet.</h6>
        </div>
        <!-- /trackbacks -->
    </div>
    <!-- /tab sections -->

```

</div>  
<!-- /[tabbed content](#) -->

---

*Test/BasicTest.java*

```
import static siena.Json.map;

import java.io.IOException;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import lib.CommonFunctions;
import models.Event;
import models.EventComment;
import models.Place;
import models.User;

import org.junit.Before;
import org.junit.Test;

import play.modules.siena.SienaFixtures;
import play.test.UnitTest;
import siena.Json;

import com.google.appengine.repackaged.org.json.JSONException;
import com.google.appengine.repackaged.org.json.JSONObject;

public class BasicTest extends UnitTest {
    Map<String, Double> loc= new HashMap<String, Double>();

    @Before
    public void setup() {
        try {
            SienaFixtures.deleteAllModels();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Test
    public void createUser() {
        try {
            new User("mauriziopz@gmail.com",
                    "Maurizio
Pozzobon","1","facebook",null).insert();
        } catch (Exception e) {}
        User user = User.findById("mauriziopz@gmail.com");
        user = User.findById(user.id);
    }
}
```

```

        assertNotNull(user);
        assertEquals("mauriziopz@gmail.com", user.email);
        assertEquals("Maurizio Pozzobon", user.nome);
        assertEquals("1", user.webId);
        assertEquals(null, user.passwordHash);
    }

    @Test
    public void testGeocode() {
        try {
            loc = CommonFunctions.geocode("Rivignano, Italia");
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        assertNotNull(loc.get("latitude"));
        assertNotNull(loc.get("longitude"));
        assertEquals((Double)13.0425, loc.get("longitude"));
        assertEquals((Double)45.8798518, loc.get("latitude"));
    }

    @Test
    public void createPlace() {
        createUser();
        testGeocode();
        Json json = null;
        json = map().put("original", "http://original")
            .put("small_square", "http://small")
            .put("large_thumbnail", "http://large")
            .put("delete_page", "http://delete");

        User user = User.findByEmail("mauriziopz@gmail.com");
        assertNotNull(user);
        Place place=null;

        try {
            place = new Place(user, "Titolo locale",
                "descrizione del locale creato tramite
test", json,
                "Rivignano, Italia", loc.get("latitude"),
                loc.get("longitude"));
        } catch (Exception e) {
            assertEquals(0,1);
        }
        place.insert();
        user = User.findByEmail("mauriziopz@gmail.com");
        List<Place> p = user.places.fetch();
        assertNotNull(p);
    }

```



```

assertFalse(p.isEmpty());
for (Place pl : p) {
    assertEquals("Titolo locale",pl.nome);
    assertEquals("descrizione del locale creato tramite
test",
                    pl.descrizione);
    assertNotNull(pl.imageLinks);

    assertEquals("http://original",
                    pl.imageLinks.get("original").asString());
    assertEquals("http://small",
                    pl.imageLinks.get("small_square").asString());
    assertEquals("http://large",
                    pl.imageLinks.get("large_thumbnail").asString());
    assertEquals("http://delete",
                    pl.imageLinks.get("delete_page").asString());

    assertEquals("Rivignano, Italia",pl.indirizzo);
    assertNotNull(pl.latitude);
    assertNotNull(pl.longitude);
}
}

@Test
public void createEvent() {
    createUser();
    User user = User.findByEmail("mauriziopz@gmail.com");
    Place place=null;
    try {
        place = new Place(user,"posto","bel
posto",null,null,null,null);
        place.insert();
        assertNotNull(user);
        Event e=null;

        e = new Event(user,place, "Festa","Questa è una gran
bella festa",null,new Date(),new Date(),false,null,null);
        e.insert();
        List<Event> events =user.events.fetch();
        assertNotNull(events);
        assertFalse(events.isEmpty());
        for (Event event : events) {
            assertEquals("Festa",event.nome);
            assertEquals("Questa è una gran bella
festa",event.descrizione);
        }
    }
}

```

```

        } catch (Exception e1) {
            assertEquals(1, 0);
        }

    }
    @Test
    public void commentTest() {
        try {
            new
User("maur@mail.com", "Maurizio", "01", "facebook", "hash").insert();
        } catch (Exception e) {}
        User user = User.findByEmail("maur@mail.com");
        Place place;
        try {
            place = new Place(user, "posto", "bel
posto", null, null, null, null);
            place.insert();
            assertNotNull(user);
            Event e = new Event(user, place, "Festa", "Questa è una
gran bella festa", null, new Date(), new Date(), false, null, null);
            e.insert();
            assertNotNull(user.nome);
            EventComment ec = new EventComment(user, e,
"TestComment", new Date());
            ec.insert();
            List<EventComment> ecs = e.comments.fetch();
            for (EventComment comment : ecs) {
                assertNotNull(comment.user.id);
                User us = User.findById(comment.user.id);
                comment.user = us;
                assertNotNull(comment.user.nome);
            }
        } catch (Exception e1) {
            assertEquals(1, 0);
        }
    }
}

```

*Test/InformaMi.test.html*

```
import com.thoughtworks.selenium.*
```

```
class InformaMi-groovy extends GroovySeleneseTestCase {
```

```

    @Override
    void setUp() throws Exception {
        super.setUp('http://kefacciamo.appspot.com/events/index',
'*chrome')
        setDefaultTimeout(30000)
        setCaptureScreenshotOnFailure(false)
    }
}

```

```

    }

    void testInformaMi-groovy() throws Exception {
        selenium.open("/events/index")
        selenium.clickAndWait("css=img[alt=\"Login with
facebook\"]")
        assertTrue(selenium.isTextPresent("Maurizio Pozzobon"))
        verifyTrue(selenium.isTextPresent("Crea evento"))
        verifyTrue(selenium.isTextPresent("Crea locale"))
        selenium.clickAndWait("//div[@id='sidebar']/ul/li[2]/div/u
l/li[2]/a/span")
        selenium.click("name=nome")
        selenium.type("name=nome", "Selenium Car")
        selenium.type("name=nome", "Selenium Locale")
        selenium.type("name=descrizione", "Descrizione di un locale
creato tramite selenium")
        selenium.type("id=indirizzo", "udine, viale venezia")
        selenium.clickAndWait("css=input[type=\"submit\"]")
        selenium.clickAndWait("//ul[@id='navigation']/li[2]/a/span"
)
        assertEquals("Selenium Locale",
selenium.getText("link=Selenium Locale"))
        verifyTrue(selenium.isTextPresent("Tetris"))
        selenium.clickAndWait("link=Selenium Locale")
        assertEquals("Selenium Locale",
selenium.getText("link=Selenium Locale"))
        verifyTrue(selenium.isTextPresent("Descrizione di un locale
creato tramite selenium"))
        verifyTrue(selenium.isTextPresent("udine, viale venezia"))
        selenium.clickAndWait("//ul[@id='navigation']/li[2]/a/span"
)
        selenium.type("name=address", "udine")
        selenium.clickAndWait("css=form > input[type=\"submit\"]")
        assertEquals("Selenium Locale",
selenium.getText("link=Selenium Locale"))
        verifyFalse(selenium.isTextPresent("Tetris"))
        selenium.clickAndWait("//ul[@id='navigation']/li[4]/a/span"
)
    }
}

```