Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

UN WEB SERVICE PER APPLICAZIONI DI GAMIFICATION DEGLI ITINERARI TURISTICI

Relatore:
Chiar.mo Prof.
LUCIANO BONONI
Correlatore:
Dr.
LUCA BEDOGNI

Presentata da:
DANIELE CORTESI

Sessione II
Anno Accademico 2014/2015

Introduzione

La *gamification*, o *gameful design*, come suggerisce Deterding [3], consiste nell'utilizzo di tecniche di game design ed elementi tipici dei giochi in contesti a essi non correlati, allo scopo di migliorarne e renderne piacevole l'esperienza, così che ci si senta invogliati a ripeterla.

La tesi propone uno strumento che applica questo concetto ai viaggi turistici, mettendo a disposizione degli utenti dei **percorsi**, composti da **obiettivi** (le pietre miliari dell'itinerario) e creati da **operatori** convenzionati (quali aziende, associazioni o altri soggetti), al termine dei quali potranno essere vinti **premi** (sconti, omaggi, ...) se verrà data prova di aver attraversato gli obiettivi richiesti.

Il Web Service offre la possibilità alle applicazioni che ne fanno uso del di mostrare agli utenti gli itinerari in maniera **geolocalizzata**, applicando in questo modo la tecnica del *marketing di prossimità*.
Essa consiste nell'operare limitatamente a un determinato luogo geografico utilizzando tecnologie di comunicazione di tipo visuale e mobile, il cui target non sono utenti con un profilo ben definito, bensì coloro che si trovano nell'area predeterminata.
Ciò è realizzato permettendo di richiedere esclusivamente i percorsi che attraversano una particolare città o che sono entro una certa distanza da precise coordinate geografiche.

In un mondo dominato dagli smartphone, l'importanza di una strategia che ne sfrutti le potenzialità in questo senso è evidente, dato che essa può essere in grado di "effettuare attività promozionali e informative al cliente che si trova vicino o all'interno del negozio nel momento in cui è disposto ad acquistare" [7] e, inoltre, buone campagne di marketing di prossimità possono valorizzare il *brand* aziendale e favorire sponsor e investimenti [5].

Lo scenario d'uso tipico riguarda specialmente chi seguirà il percorso come intrattenimento durante il proprio periodo di vacanza, incentivato dalla consapevolezza che al

termine dell'itinerario otterrà un qualche premio recandosi nel negozio dell'azienda, ma questo non è l'unico scenario possibile. Un operatore infatti può creare i tragitti nel modo che preferisce, per esempio definendone uno che attraversa tutti i suoi negozi o un loro sottoinsieme.

L'applicazione ha il fine, principalmente, di mettere nelle mani delle aziende uno strumento aggiuntivo per attrarre nuova clientela, rendendone l'esperienza allo stesso tempo divertente ed economicamente vantaggiosa. Infatti, come scrive Sciortino, "applicare strategie volte ad attrarre e fidelizzare un cliente/visitatore attraverso logiche ludiche in un contesto per sua natura non propriamente *gaming*, [...] porta il pubblico a cambiare il proprio atteggiamento da consumatore a protagonista della propria esperienza" [10]. Inoltre, anche se non sono state fatte ricerche specifiche sugli effetti della *gamification* nel campo del marketing, essi hanno un effetto molto positivo sulle metriche di qualità del servizio, fondamentali in questo settore [6].

Esistono già nel mercato applicazioni con caratteristiche simili, discusse nel capitolo 1, ma nessuna di esse presenta le funzionalità di questo Web Service, in particolare di offrire agli operatori la possibilità di creare i propri percorsi totalmente personalizzati a cui associare precisi premi.

Indice

Capitolo 1

Stato dell'arte

Il marketing di prossimità non è un fenomeno totalmente nuovo, ma negli ultimi anni si sta affermando in modo deciso grazie ai progressi tecnologici. Descritto ampiamente nella tesi a opera di Negri [7], la sua importanza è anche testimoniata da un articolo [5] che sottolinea come questa strategia possa valorizzare il brand aziendale, e da un altro [11] che ne evidenzia l'efficacia di abbinare alla tecnica aspetti *social*.

Inoltre, uno studio sulla diffusione di applicazioni location-based in Croazia [9] ha condotto delle interviste che mostrano come gli utenti utilizzino queste app per ottenere informazioni a loro rilevanti nel modo e tempo che più li aggrada e come la motivazione principale per cui ne fanno uso sia la possibilità di incontrare conoscenti vicini alla loro posizione. L'articolo sottolinea anche i vantaggi che ne traggono le aziende, grazie soprattutto al rapporto che riescono così a instaurare con i clienti, che non rappresenta un grosso investimento ma al contrario un'efficiente attività promozionale.

Nell'uso di questa strategia sono però da tener presente le implicazioni legali: un articolo [2] rileva che le tecnologie di geolocalizzazione permettono di delineare un profilo preciso delle abitudini di spostamento degli utenti, nonchè di averne sempre a disposizione la posizione corrente, invadendone così la privacy. Il testo chiarifica che le applicazioni più esposte a problemi legali sono quelle client-side, dato che quelle lato server non si interessano di identificare un utente particolare. L'articolo suggerisce di tutelarsi fornendo tutte le informazioni a riguardo negli User Agreements, di ottenere anticipatamente il consenso dell'utente a riguardo di queste pratiche e di garantire la sicurezza delle informazioni raccolte.

Nemmeno la gamification rappresenta un concetto nuovo, ma solo di recente si è rivelata uno degli strumenti più efficaci per favorire il coinvolgimento dei consumatori.

Il termine è oggetto di un articolo [3] che ne cerca una definizione precisa analizzandone la storia e le diverse sfaccettature, riuscendo a riassumerlo in "the use of game design elements in non-game contexts" ed esplicita l'impossibilità di stabilire se un dato sistema è gamified senza conoscere le intenzioni degli sviluppatori o l'esperienza che ne hanno gli utenti.

Un'altra pubblicazione [6] raccoglie i risultati di studi sugli effetti della gamification in diversi campi e ne sottolinea l'impatto positivo, che nel campo del marketing riguarda l'aumento percepito della qualità del servizio. Il testo mette anche in evidenza che gli effetti positivi non sono sempre a lungo termine ma dovuti alla novità dell'iniziativa, e che una volta gamificato il servizio, tornare indietro sarebbe controproducente perchè probabilmente gli utenti non gradirebbero di perdere eventuali punti ottenuti. Infatti un articolo [11] enfatizza come i consumatori provino grande divertimento nel ricevere ricompense raggiungendo certi obiettivi, il tutto competendo con altri tramite i social network. Il testo spiega anche come i premi ricevuti dagli utenti, monetari e non, siano una delle cose che li spingano maggiormente a recarsi in negozio.

La tesi di Sciortino dedicata alla gamification [10] sottolinea l'importanza della ludicizzazione nel marketing orientato al turismo e presenta diversi casi di studio, tra cui anche applicazioni mobili, il cui mondo è presentato in modo approfondito nella dissertazione di Buresti [1].

In essa sono nominate applicazioni che si sono imposte in questo campo, tra cui **Foursquare** (foursquare.com), che mette gli utenti di una città in competizione al fine di stilare una classifica di chi visita più luoghi della stessa e assegnare badge di riconoscimento a chi raggiunge determinati obiettivi.
Le similitudini con il lavoro descritto in questa tesi sono evidenti: in entrambi i casi gli utenti si recano in luoghi predeterminati e ottengono premi nel farlo. La differenza sostanziale è che Foursquare non mette nelle mani degli gli operatori nè la definizione dei percorsi, nè quella dei i premi.

Un'altra importante realtà discussa è **TripAdvisor** (tripadvisor.com), insieme alla quale può essere nominata **Gogobot** (gogobot.com). Entrambe aiutano nella pianificazione di viaggi turistici mettendo a disposizione recensioni di altri viaggiatori.

L'unico aspetto in comune con il Web Service proposto è il settore turistico. L'interoperazione però sarebbe facile: le applicazioni che forniscono agli utenti un'interfaccia al servizio in oggetto potrebbero segnalare le recensioni su TripAdvisor e Gogobot degli operatori che offrono premi, mentre queste potrebbero indicare se nella città considerata sono presenti percorsi registrati nel Web Service.

Nella tesi di Buresti viene inoltre descritta l'innovativa **Freeppie** (freeppie.com), che permette agli utenti di usufruire gratuitamente degli spazi invenduti (come camere o tavoli) degli operatori convenzionati in cambio di recensioni live sui social network, accumulando così punti utili per prenotare altre offerte inserite nell'applicazione.
Queste offerte possono essere viste come i premi del lavoro proposto, mentre i punti necessari a ottenerli come gli obiettivi da raggiungere, ma le caratteristiche comuni terminano qui.

Un'ulteriore app del settore non citata è **Geocaching** (geocaching.com), una caccia al tesoro *community-driven*, dove i giocatori nascondono oggetti e ne indicano la posizione GPS affinchè altri li trovino.
In questo caso gli aspetti simili riguardano solamente la geolocalizzazione e la visita di luoghi precisi. Tuttavia, questo gioco può servire da spunto per possibili utilizzi differenti del servizio proposto in questa tesi, come descritto nelle conclusioni.

Capitolo 2

Progettazione

Prima di spiegare nel dettaglio cosa è stato realizzato, è necessario introdurne gli elementi fondamentali.

Utenti. Sono coloro che percorrono i tragitti, ne completano gli obiettivi e vincono i premi.

Operatori. Essi creano percorsi e obiettivi e ne stabiliscono i premi associati.

Percorsi. Sono gli itinerari creati dagli operatori a beneficio degli utenti. Essi sono costituiti da obiettivi da raggiungere, al completamento dei quali è prevista l'assegnazione di un premio.

Obiettivi. Sono le pietre miliari del percorso, attraverso cui gli utenti devono dare prova di essere passati per vincere i premi.

Premi. Consistono ad esempio in sconti e omaggi ed è a loro collegata una lista di obiettivi obbligatori (relativi a ogni percorso) che gli utenti devono attraversare per poterli vincere.

Il Web Service predispone procedure per le applicazioni client-side mediante le quali utenti e operatori possono registrare un account ed effettuare login al fine di usufruire del servizio. L'accesso può anche avvenire tramite Facebook indicandone l'id del profilo, che se risulta essere utilizzato per la prima volta determina un processo di registrazione.

Le informazioni dell'account possono essere modificate in seguito in qualsiasi momento ed è anche possibile collegarvi un profilo Facebook successivamente a una registrazione avvenuta nel modo tradizionale.

Sono previste operazioni per ottenere i percorsi e tutte le informazioni a loro associate (cioè obiettivi, premi e operatore di appartenenza) in maniera geolocalizzata, indicando la città o il punto geografico di interesse.

È predisposto un servizio dedicato agli utenti che permette loro di dar prova di aver attraversato ogni singolo obiettivo, e che comunicherà se e quali premi sono stati vinti e le informazioni su tutti i percorsi che passano per l'obiettivo verificato.

Una volta vinto un premio, gli utenti possono ottenerne il codice di riscossione (valido solamente entro un tempo limitato) tramite un'apposita operazione. Il codice deve poi essere mostrato all'operatore a cui appartiene il premio, che a sua volta si avvale del servizio per verificarne la validità.

Infine, sono dedicate agli operatori tutta una serie di funzionalità per creare, modificare ed eliminare percorsi e premi. Gli obiettivi invece, una volta inseriti non sono più modificabili, poichè sono condivisi da tutti gli operatori.

Capitolo 3

Implementazione

3.1 Note generali

Il Web Service è raggiungibile all'indirizzo daniele.cortesi2.web.cs.unibo.it/wsgi, è realizzato in python facendo uso del framework Flask [8] e fornisce un'API JSON che mette a disposizione tutte le funzionalità previste dall'applicazione.

Si appoggia a un database MySQL (figura 3.3), nel quale sono inserite tutte le informazioni su operatori, utenti, percorsi, obiettivi e premi, e utilizza il modulo MySQLdb [4] per interagirci.

Si compone dei seguenti moduli:

main.py Il modulo principale dell'applicazione, esso definisce tutti gli URL delle possibili richieste e le procedure interne utilizzate nella loro realizzazione;

query.py Implementa la classe Query che interagisce con il modulo MySQLdb per eseguire le query richieste e riorganizzarne i risultati. Essa espone costanti tramite la classe interna code che identificano le query attuabili;

tables.py, jsonkeys.py Esprimono le costanti che vengono utilizzate all'interno del Web Service. Il primo espone la classe Tables, comprensiva delle variabili con i nomi delle tabelle del database e la classe F (che sta per Fields) che specifica i nomi degli attributi delle tabelle. Il secondo definisce la classe Keys, che espone le stringhe usate come chiavi nei JSON delle richieste e delle risposte HTTP.

3.2 Caratteristiche

Di seguito sono descritte le caratteristiche degli elementi fondamentali del servizio.

3.2.1 Percorsi

I percorsi si compongono dei seguenti dettagli:

- Nome
- Descrizione
- Operatore di appartenenza
- Giorni di validità dal completamento di un obiettivo
- Elenco di tag
- Elenco di obiettivi
- Elenco di premi con probabilità di essere vinti e obiettivi obbligatori

3.2.2 Obiettivi

Gli obiettivi si compongono delle seguenti caratteristiche:

- Nome
- Descrizione
- Posizione (latitudine e longitudine)
- Codice di validazione

Gli obiettivi sono in comune tra tutti gli operatori. Questo comporta che una volta inseriti non potranno essere modificati o cancellati.

3.2.3 Premi

I premi si compongono dei seguenti dettagli:

- Nome
- Descrizione
- Visibilità agli utenti
- Giorni che devono trascorrere tra vincite differenti

3.2.4 Utenti

Agli utenti sono associate le seguenti informazioni:

- Username
- Password
- Elenco di tag che identificano le preferenze sui percorsi
- Elenco di obiettivi completati con indicazione della data
- Elenco di premi vinti (riscossi e non)

3.2.5 Operatori

Gli operatori si compongono delle seguenti caratteristiche:

- Username
- Password
- Nome azienda
- Posizione (latitudine e longitudine)
- Elenco di tag che identificano il tipo di premi offerti

3.3 Database

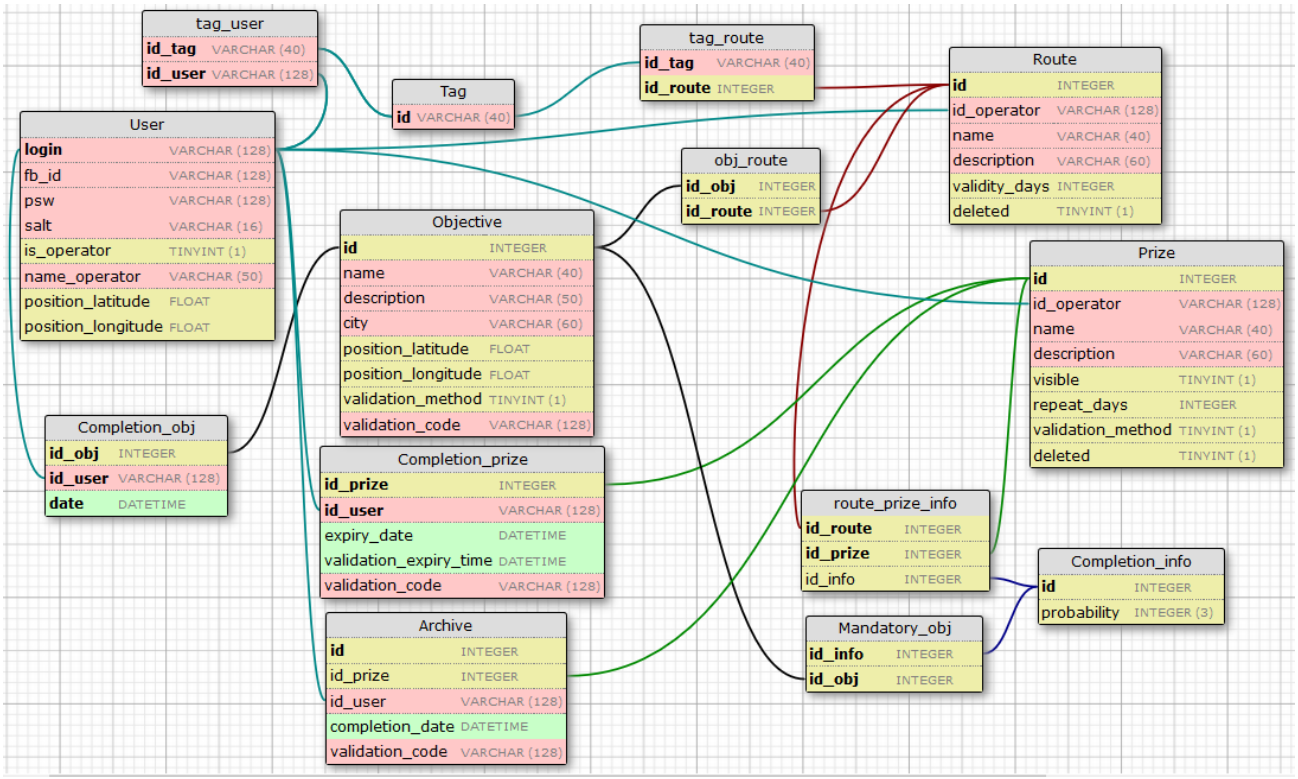

Figura 3.1: Database alla base del Web Service

3.4 Glossario

Nel corso del capitolo verrano utilizzati i seguenti termini, che indicano frammenti di JSON composti nel modo descritto:

info percorso

```
{
  "id": int,
  "id_operator" : basestring, (hash del login dell'operatore)
  "name": basestring,
  "description": basestring,
  "validity_days": int,
  "tags": [basestring]
}
```

info obiettivo

```
{
  "id": int,
  "name": basestring,
  "description": basestring,
  "city": basestring,
  "position_latitude": float,
  "position_longitude": float,
  "validation_method": int
}
```

info premio

```
{
  "id": int,
  "id_operator" : basestring,
  "name": basestring,
  "description": basestring,
  "visible": bool,
  "repeat_days": int,
  "validation_method": int
}
```

info operatore

```
{
  "id" : basestring, (hash del login dell'operatore)
  "name_operator": basestring,
  "position_latitude": float,
  "position_longitude": float,
  "tags": [basestring]
}
```

3.5 Verifica forma JSON

La validità di tutti i dati in formato JSON ricevuti come parametri delle richieste è controllata attraverso la funzione is_well_formed. Essa è utilizzata estensivamente all'interno del programma per verificare la correttezza della forma di dizionari, appurando che contengano tutte le chiavi richieste e che i valori siano del tipo corretto.
Prende in input due dizionari, il primo è quello da controllare, mentre il secondo quello che definisce il formato da rispettare. Quest'ultimo deve essere nella forma {chiave: tipo | [tipo] | dict}.
La valutazione del formato del primo parametro avviene verficando che tutte le chiavi

del secondo siano in esso presenti e che il loro valore sia del tipo indicato.
Quello che avviene nei tre casi può essere riassunto così:

tipo - controlla se il valore è del tipo "tipo" (N.B. int e long sono considerati equivalenti)

[tipo] - controlla se il valore sia una lista e che tutti gli elementi siano del tipo "tipo"

dict - applica ricorsivamente la funzione is_well_formed al valore

La funzione restituisce in output un valore booleano per comunicare l'esito della valutazione.

Nel caso il controllo fallisca, viene inviata una risposta HTTP con codice 400 BAD REQUEST. Se invece ha successo e l'operazione richiesta va a buon fine, la risposta HTTP conterrà codice 200 OK ed eventualmente un JSON nel corpo.

3.6 Esecuzione delle query

Le query sono tutte effettuate invocando il metodo execute della classe Query, che prende come parametri il database su cui eseguire la query, il codice della stessa, un dizionario (o lista di dizionari per le query insert) contenente i parametri e un booleano che stabilisce se effettuare o no il commit dei cambiamenti al database.
Le chiavi del dizionario corrispondono, nel caso in cui il parametro identifichi un campo di una tabella, alle costanti definite in tables.py, altrimenti (ad esempio per il parametro che stabilisce minuti di validità del codice per la riscossione di un premio) alle costanti esposte nella classe var, contenuta in Query.

In base al tipo della query, il metodo restituisce risultati differenti:

select - lista di dizionari, dove le chiavi sono le stesse di cui sopra;

insert - id del primo record inserito;

update e delete - numero di record modificati.

3.7 Registrazione account

La registrazione di un nuovo account viene effettuata inviando una richiesta POST all'URL /signup, fornendo le informazioni necessarie ed eventualmente anche tag da collegarvi. Se il login indicato risulta non essere già presente nel database, allora l'account viene creato.

JSON atteso[1]:

```
{
  "login": basestring,
  "psw": basestring,
  "is_operator": bool,
  ("name_operator": basestring,)
  ("position_latitude": float,)
  ("position_longitude": float,)
  ("tags": [basestring])
}
```

La verifica che non esista un account con lo stesso login avviene eseguendo una query e se fallisce comporta una risposta 409 CONFLICT.
Altrimenti l'account viene registrato chiamando la funzione add_new_user, passando come parametro il dizionario costruito da Flask a partire dal JSON ricevuto. Essa aggiunge al database l'utente le cui informazioni sono contenute nel dizionario in input, senza però controllarne la validità della forma, che in questo caso è già stata precedentemente verificata.
Genera casualmente un salt di 16 caratteri per la password utilizzando os.urandom e computa la password da inserire nel database come l'hash (sha512) della concatenazione di password e salt.

I tag eventualmente forniti nella richiesta sono gestiti chiamando la funzione edit_user_tags. Essa non è altro che un'interfaccia per la procedura edit_tags_of, che invoca passandogli i propri parametri, cioè il login dell'utente e un dizionario dove cercare i tag, e in più imposta il parametro of_what a 'id_user', in modo tale che venga eseguito il codice

[1] Indicati tra parentesi tonde sono i parametri opzionali.

per modificare i tag degli utenti.
La funzione controlla se nel dizionario sono presenti le seguenti chiavi, e in tal caso che il valore rispetti la forma corretta:

```
{
  "tags_to_add": [basestring],
  "tags_to_remove": [basestring]
}
```

La procedura aggiunge i tag contenuti in tags_to_add e rimuove quelli in tags_to_remove dal percorso o utente chiamando, rispettivamente, le funzioni add_tags_to e remove_tags_from. Queste due procedure prendono come parametro una variabile che stabilisce se modificare i tag di percorsi o utenti, un id (del percorso o login utente), la lista dei tag e un booleano che stabilisce se effettuare o meno il commit dei cambiamenti al database. Esse eseguono query, rispettivamente, per collegare e rimuovere i tag forniti e restituiscono un booleano per comunicare il successo dell'operazione.

La funzione edit_tags_of non esegue il commit dei cambiamenti e restituisce anch'essa un valore booleano che identifica se tutte l'operazioni sono andate a buon fine oppure no, nel caso in cui si sia cercato di aggiungere un tag inesistente o già collegato.

3.8 Login e logout

Il login può essere effettuato sia con un account creato nel modo appena descritto, sia utilizzando quello di un profilo Facebook. Nel primo caso si invia una richiesta POST all'URL /login, contenente login e password, nel secondo a /loginfb, contenente l'id dell'account.
Se l'autenticazione ha successo viene impostata la sessione HTTP tramite cookies e fornite le informazioni dell'account, ad eccezione di login e password.

Nel caso di /login, il JSON richiesto è:

```
{
  "login": basestring,
  "psw": basestring
}
```

L'autenticazione avviene verificando la correttezza delle credenziali chiamando la funzione authenticate, che determina se la coppia (utente, password) fornita in input è presente nel database effettuando l'hash (sha512) della password concanata al salt, ricavato eseguendo una query.
Restituisce un dizionario con le informazioni dell'utente se le credenziali risultano corrette, altrimenti None.
In quest'ultimo caso viene inviata una risposta 401 UNAUTHORIZED.

Nel caso di URL /loginfb il JSON atteso è:

```
{
  "fb_id": basestring,
  ("is_operator": bool,)
  ("login": basestring,)
  ("psw": basestring)
}
```

L'autenticazione viene eseguita verificando se l'hash dell'id Facebook (fb_id) corrisponde a quello di un account nel database e se ha successo vengono fornite le informazioni utente.
Altrimenti, se nel JSON mancano login e psw, viene creato un nuovo account chiamando la funzione add_new_user (3.7) con login equivalente all'hash dell'id Facebook e password casuale. Se il parametro is_operator non è fornito, si suppone che si voglia effettuare il login con un account di tipo utente.
Se invece login e psw sono presenti, le credenziali vengono verificate tramite la funzione authenticate, e nel caso risultino corrette l'id Facebook viene collegato a tale utente. In caso contrario viene inviata una risposta 401 UNAUTHORIZED.

Il collegamento di un account già esistente a Facebook può anche avvenire, dopo aver effettuato il login, inviando una richiesta POST all'URL /connectfb con JSON:

```
{ "fb_id" : basestring }
```

Questo comporta l'invocazione di connect_fb che esegue la query per settare l'attributo fb_id. Se fallisce (perchè a un altro utente era già stato assegnato quell'fb_id), allora la funzione restituisce False, comportando una risposta 409 CONFLICT, altrimenti True.

Il logout avviene mediante una richiesta POST all'URL /logout, che resetta la sessione HTTP del client.

3.9 Modifica informazioni account

Dopo aver completato la registrazione, è possibile modificare le informazioni dell'account con una richiesta all'URL /user/edit, indicandole in un JSON insieme ad eventuali tag da aggiungere o rimuovere:

```
{
  ("name_operator": basestring,)
  ("position_latitude": float,)
  ("position\_longitude": float,)
  ("tags_to_add": [basestring],)
  ("tags_to_remove": [basestring])
}
```

Se la richiesta è eseguita senza aver precedentemente compiuto login viene inviata una risposta 401 UNAUTHORIZED.
Altrimenti vengono modificate le informazioni utente eseguendo una query e i tag chiamando la procedura edit_user_tags (3.7).

3.10 Operazioni comuni

3.10.1 Ottenere i percorsi

Una delle funzionalità più importanti è quella di esporre i percorsi indicandone i loro obiettivi e premi. Queste informazioni possono essere ottenute effettuando una richiesta GET all'URL /routes , che risponde con un JSON nella seguente forma[2]:

```
{
  "routes": [
    {
      info percorso,
      "objs": [int],
      "prizes": [
        {
          "id": int,
          "mandatory_objs": [int],
          "probability": int
        }
      ]
    }
  ],
  "objs": [{info obiettivo}],
  "prizes": [{info premio}],
  "operators": [{info operatore}]
}
```

Se la richiesta avviene senza aver prima eseguito il login o avendolo compiuto con un account utente, vengono inseriti tutti i percorsi e i relativi obiettivi e premi. Se invece la richiesta proviene da un account operatore, allora sono inclusi solamente i percorsi ad esso appartenenti, con l'aggiunta di tutti gli obiettivi presenti nel database.

È anche possibile richiedere i dettagli dei percorsi in una certa città, oppure entro un determinato raggio da una posizione geografica qualsiasi, aggiungendo queste informazio-

[2]Il significato di info percorso, obiettivo, premio e operatore è descritto nel glossario (3.4)

ni nell'URL, rispettivamente, nei modi seguenti: /routes/<città> e /routes/<latitudine>/-<longitudine>/<raggio>.

Il JSON è costruito chiamando la funzione get_all_info_from_routes, che prende in input i risultati di una query che seleziona i percorsi e le loro informazioni e ne ricava gli obiettivi, premi, operatori (che inserisce solamente una volta anche se parte di più percorsi) e i tag effettuando ulteriori query.
Per procurarsi questi ultimi chiama la funzione get_tags, che prende in input una lista di login di account e una lista di id di percorsi ed esegue le query necessarie per selezionare tutti i loro tag e restituirli.
Per ricavare le specifiche dei premi in relazione ai percorsi (probabilità e obiettivi obbligatori) ricorre alla funzione parse_prizes_from_routes: essa prende in input i risultati di una query che seleziona premi e i loro obiettivi obbligatori e il dizionario contenente i percorsi, a cui, per ognuno, aggiunge le informazioni su ciascun premio, cioè la probabilità di essere assegnato rispetto agli altri e la lista degli id degli obiettivi obbligatori per ottenere quel premio. Restituisce tutti i dati collezionati in diversi formati, cioè: una lista contenente tutti i premi, un dizionario che li lega ai loro id, un set con gli id degli operatori a cui appartengono i premi e un dizionario che lega l'hash del login dell'operatore al suo premio.
Per disporre di informazioni e tag degli operatori invoca la funzione get_operators_with_tags, che prende in input un set contenente login di operatori ed effettua una query per selezionarne le informazioni e si affida anch'essa alla funzione get_tags per ricavarne i tag. Restituisce la lista che sarà poi inclusa nel JSON nella chiave 'operators'.

3.10.2 Ottenere gli obiettivi

Se necessario, è possibile ottenere l'elenco di tutti gli obiettivi presenti nel database, anche se non fanno parte di nessun percorso, effettuando una richiesta GET all'URL /objs. Verrà ricevuto un JSON nella forma[3]:

```
{ "objs" : [{info obiettivo}] }
```

[3]Il significato di info obiettivo è descritto nel glossario (3.4)

3.10.3 Ottenere i tag

È possibile ricevere l'elenco di tutti i tag presenti nel database utilizzati per percorsi e utenti con una richiesta GET all'URL /tags ottenendo in risposta un JSON nella forma:

```
{ "tags": [basestring] }
```

3.11 Operazioni lato operatore

Tutte le operazioni di seguito descritte necessitano del login con un account operatore. In caso contrario, verrà inviata una risposta 401 UNAUTHORIZED.

3.11.1 Aggiunta di obiettivi

Nuovi obiettivi possono essere creati per mezzo di una richiesta POST all'URL /objs/add e fornendone i dettagli. Nella risposta verranno comunicati gli id e i codici di validazione degli obiettivi creati. Il JSON atteso è:

```
{
  "objs": [
    {
      "name": basestring,
      "description": basestring,
      "city": basestring,
      "position_latitude": float,
      "position_longitude": float,
      "validation_method": int
    }
  ]
}
```

Gli obiettivi sono aggiunti al database chiamando add_new_objs, che prende in input una lista di dizionari contenenti le loro informazioni e un booleano che stabilisce se compiere o meno il commit alla fine dell'operazione.

Per ogni obiettivo la funzione computa un codice di validazione calcolando l'hash (sha512) della concatenazione tra nome e una stringa causuale risultante invocando os.urandom. Infine, esegue una query per inserire tutti i percorsi nella tabella Objective.
La procedura restituisce una lista con gli id e i codici di validazione degli obiettivi, dove gli elementi sono così composti:

```
{
  "id": int,
  "validation_code": basestring
}
```

Il JSON in risposta conterrà la chiave objs a cui sarà associata la lista risultante dalla chiamata a add_new_objs.

3.11.2 Aggiunta e modifica di percorsi

Gli operatori possono aggiungere e modificare percorsi mediante richieste, rispettivamente, POST all'URL /route/add e PUT a /route/edit, fornendone le caratteristiche. È anche possibile indicare obiettivi esistenti o nuovi da aggiungere al percorso (o da rimuovere, nel caso di modifica) e tag da aggiungere (o rimuovere) per catalogare il percorso.

Il JSON richiesto[4] per l'aggiunta è:

```
{
  "name": basestring,
  "description": basestring,
  "validity_days": int,
  ("new_objs": [{info obiettivo}],)
  ("id_objs_to_add": [int],)
  ("tags_to_add": [basestring],)
}
```

[4]Il significato di info obiettivo è descritto nel glossario (3.4)

Mentre il JSON atteso per la modifica richiede anche:

```
{
  "id": int,
  ("id_objs_to_remove": [int],)
  ("tags_to_remove": [basestring])
}
```

Se la richiesta riguarda la modifica, allora viene prima controllato che il percorso appartenga all'operatore invocando la funzione route_belongs_to_operator. Essa ne prende in input l'id e il login di un operatore ed esegue una query per selezionare il record dalla tabella Route con tali parametri. Nel caso venga ottenuto un risultato restituisce True, altrimenti False, comportando una risposta con codice 401 UNAUTHORIZED.

In seguito, è attuata un'altra query per aggiornare le informazioni dell'itinerario di cui è stato fornito l'id con quelle ricevute, oppure, nel caso di richiesta di aggiunta, il suo record viene inserito ex novo.

Il parametro opzionale new_objs consente di creare nuovi obiettivi da incorporare nel percorso, assieme al codice di quelli già esistenti specificati nella lista id_objs_to_add.

La modifica degli obiettivi del percorso avviene chiamando la funzione edit_route_objs, che prende in input l'id dello stesso e un dizionario dove cercare gli obiettivi da creare, inserire e rimuovere.
Quelli da creare e collegare al percorso sono cercati nella chiave new_objs e sono effettivamente inseriti nel database chiamando la funzione add_new_objs (3.11.1), e i loro id sono aggiunti alla lista id_objs_to_add del dizionario (che viene creata se non presente).
Successivamente, tutti questi vengono aggiunti all'itinerario chiamando add_objs_to_route, mentre quelli i cui id sono nella lista id_objs_to_remove vengono rimossi, invocando remove_objs_from_route ed eseguendo una query per escluderli da quelli obbligatori di tutti i premi del percorso.
Le due funzioni citate prendono in input l'id del percorso, la lista degli id degli obiettivi e un booleano che stabilisce se effettuare o meno il commit dei cambiamenti, ed eseguono la query, rispettivamente, per aggiungerli o rimuoverli dal tragitto, per poi restituire un booleano che comunica l'esito dell'operazione.
La funzione edit_route_objs non esegue il commit dei cambiamenti e restituisce una tupla il

cui primo elemento è un valore booleano che identifica se tutte le operazioni sono andate a buon fine oppure no, mentre il secondo la lista restituita da add_new_objs (3.11.1).

La modifica dei tag del percorso avviene invocando edit_route_tags, che si comporta come edit_user_tags (3.7), con la differenza che chiama edit_tags_of impostando il parametro of_what a 'id_route', in modo che sia eseguito il codice per modificare i percorsi.

Se tutte le operazioni hanno successo viene inviata una risposta contenente un JSON nella forma:

```
{
  "id_route": int,
  "objs": [
    {
      "id": int,
      "validation_code": basestring
    }
  ]
}
```

3.11.3 Eliminazione di percorsi

La richieste di eliminazione dei percorsi può essere effettuata all'URL /route/delete con metodo DELETE, fornendo gli id di quelli da rimuovere. La cancellazione è di tipo logico, gli itinerari rimangono quindi nel database ma vengono marcati e non risulteranno più in alcuna ricerca.

Il JSON atteso è:

```
{ "id": [int] }
```

Non viene effettuato un controllo esplicito per verificare che il percorso appartenga all'operatore, ma è invece eseguita una query che setta l'attributo deleted a 1 dei percorsi di cui è stato fornito l'id se il loro operatore è quello loggato.
Se nessun record risulta modificato, la risposta sarà 400 BAD REQUEST.

3.11.4 Aggiunta e modifica di premi

Gli operatori possono aggiungere e modificare premi effettuando richieste, rispettivamente, POST all'URL /prize/add e PUT /prize/edit, fornendone le caratteristiche. È anche possibile indicare obiettivi da aggiungere (o rimuovere, nel caso di modifica) da quelli obbligatori per vincere il premio.

Il JSON richiesto per l'aggiunta è nella forma:

```
{
  "name": basestring,
  "description": basestring,
  "visible": bool,
  "repeat_days": int,
  "validation_method": int,
  "id_route": int,
  "probability": int,
  ("mand_objs_to_add": [int])
}
```

Mentre il JSON atteso per la modifica non richiede obbligatoriamente id_route e probability ma:

```
{
  "id": int,
  ("mand_objs_to_remove": [int])
}
```

Per quanto riguarda l'aggiunta, viene controllato che il percorso indicato appartenga all'operatore chiamando la funzione route_belongs_to_operator (3.11.2), e in caso positivo viene creato il premio di cui sono state fornite le informazioni e annesso a quelli del percorso di cui è stato indicato l'id chiamando la funzione add_prize_to_route, che prende in input i loro id e li collega effettuando due query, la prima per creare un nuovo record in Completion_info e la seconda per legarli effettivamente inserendone uno in route_prize_info, utilizzando l'id ottenuto dalla query precedente.

Nel caso la query fallisca (perchè percorso o premio non esistono o erano già collegati), restituisce None e viene inviata una risposta 400 BAD REQUEST. Altrimenti rimanda in output l'id_info ricavato dalla prima query (che identifica l'associazione tra percorso e premio), utilizzato poi per aggiungere al premio la probabilità di essere vinto e gli eventuali obiettivi obbligatori forniti invocando edit_prize_mand_objs_and_probability.
La funzione ha come parametri un id_info, un dizionario e un booleano che stabilisce se la chiave probability deve essere presente obbligatoriamente e nel caso non lo sia, restituisce False. Altrimenti esegue una query per modificare la probabilità del premio e chiama le funzioni add_mand_obj_to_info e rem_mand_obj_from_info, che prendono in input un id_info, una lista di id id obiettivi e un booleano che distingue se effettuare o meno il commit. Esse eseguono query per, rispettivamente, aggiungere o rimuovere gli obiettivi obbligatori dal premio e restituiscono True se non falliscono, altrimenti False (se l'id_info fornito o uno o più obiettivi non esistono).

Se tutte le operazioni vanno a buon fine, la risposta comunicherà l'id del premio appena creato in un JSON nella forma:

```
{ "id_prize": int }
```

Per quanto riguarda invece la modifica, viene controllato che il premio sia di proprietà dell'operatore invocando prize_belongs_to_operator, che si comporta come route_belongs-_to_operator (3.11.2), ma esegue la query sulla tabella Prize.
Se la verifica ha successo viene effettuata una query per modificare le informazioni del premio (senza effettuare il commit) e, se è fornito id_route, dopo aver accertato che il percorso appartenga all'operatore, si richiama la funzione get_info_from_route_prize, che prende in input l'id del premio, del percorso e il login dell'operatore ed esegue una query per ricavare e restituire l'id_info corrispondente nella tabella route_prize_info, oppure, se non ottiene risultati, None.
In quest'ultimo caso viene inviata una risposta 400 BAD REQUEST, altrimenti si invoca la funzione edit_prize_mand_objs_and_probability (3.11.4) per procedere all'eventuale modifica della probabilità e degli obiettivi obbligatori.

È inoltre possibile aggiungere un premio già esistente a un percorso con una richiesta POST a /route/add/prize, fornendo percorso, premio e tutte le informazioni che li legano.

Il JSON atteso è nella forma:

```
{
  "id_route": int,
  "id_prize": int,
  "probability": int,
  ("mand_objs_to_add": [int])
}
```

Le operazioni compiute sono le medesime di quelle per l'aggiunta di un nuovo premio, con la differenza che questo non deve essere creato, ma soggetto a verifica di appartenenza all'operatore.

3.11.5 Eliminazione di premi

È possibile eliminare i premi sia da un certo percorso, sia completamente, effettuando richieste DELETE, rispettivamente, agli URL /route/remove/prize e /prize/delete. Nel primo caso viene fisicamente eliminato il record che identifica il legame tra premio e percorso, mentre nel secondo la cancellazione è logica.

Il JSON atteso per l'eliminazione da un percorso è:

```
{
  "id_route": int,
  "id_prize": int
}
```

Mentre per la cancellazione del premio in sè:

```
{ "id": [int] }
```

In entrambi i casi non vengono compiuti controlli espliciti per verificare se il premio e il percorso appartengano all'operatore. Sono invece eseguite query che hanno effetto solo se l'operatore a cui appartiene il premio coincide con quello che ha attuato la richiesta.

Nel caso di /route/remove/prize la rimozione è gestita chiamando la funzione remove_prize_from_route, che prende in input l'id del premio, del percorso e il login dell'operatore e cancella il premio dal percorso per mezzo di una query per eliminare il record corrispondente in route_prize_info. Se la tabella risulta modificata restituisce True, altrimenti False.

Invece nel caso di /prize/delete viene eseguita una query che setta l'attributo deleted a 1 dei premi se il loro id è tra quelli forniti e il loro operatore quello loggato.

In entrambe le situazioni, se nessun record risulta modificato viene inviata una risposta 400 BAD REQUEST.

3.11.6 Verifica vittoria premi

Quando un utente si presenta da un operatore dopo aver vinto un premio e fornendone il codice, l'operatore può verificarne la validità con una richiesta POST all'URL /prize/verify, ottenendo le informazioni del premio nel caso il codice sia corretto, altrimenti la data in cui esso è stato vinto l'ultima volta dall'utente.

Il JSON atteso è nella forma:

```
{ "validation_code": basestring }
```

Una query viene effettuata per ottenere il premio a partire dal codice di validazione fornito che sia stato generato entro 5 minuti prima, se la selezione non produce risultati, ne viene eseguita un'altra per ottenere la data in cui il premio è stato vinto l'ultima volta (se è mai stato vinto) e inserita in un JSON così formato:

```
{ "last_won_date": basestring }
```

che viene inviato con codice HTTP 400 BAD REQUEST.
Se invece il codice è valido, viene verificato che l'operatore a cui appartiene sia quello che da cui proviene la richiesta. Se non è così, viene inviata una risposta 403 FORBIDDEN. Altrimenti vengono effettuate query per eliminare il record della vincita del premio dalla tabella Completion_prize e per inserirne uno nella tabella Archive, e viene restituito un JSON contenente le informazioni del premio.

3.12 Operazioni lato utente

Le operazioni di seguito descritte necessitano che sia stato precedentemente effettuato il login. In caso contrario, verrà inviata una risposta 401 UNAUTHORIZED.

3.12.1 Verifica di un obiettivo

Durante l'itinerario di un percorso, ogni qualvolta l'utente giunge a un obiettivo, deve acquisirne il codice di validazione (che sarà ad esempio in forma di QR Code) ed inviare una richiesta POST all'URL /obj/verify per attestare il completamento di tale obiettivo.

Il JSON richiesto è nella forma:

```
{ "validation_code": basestring }
```

Il completamento dell'obiettivo di cui è stato fornito il codice di validazione è registrato chiamando la funzione validate_code_for_user, che prende in input tale codice e il login di un utente. Essa verifica se il codice corrisponde effettivamente a un obiettivo eseguendo una query, e nel caso ne compie un'altra per registrare il completamento dell'obiettivo includendo data e ora. Infine, se il codice non era valido restituisce None e viene inviata una risposta 400 BAD REQUEST, altrimenti manda in output l'id dell'obiettivo.

In questo caso si procede a verificare se l'utente può vincere premi grazie al completamento del nuovo obiettivo chiamando assign_prizes_if_needed, il cui parametro è il login di un utente.
La funzione per prima cosa esegue una query per ottenere gli obiettivi nei percorsi che sono stati completati dall'utente in data utile per vincere premi (in base all'attributo validity_days della tabella Route).
In seguito effettua un'altra query per recuperare i premi collegati a quei percorsi che non sono tra quelli vinti e non ancora riscossi dall'utente e il risultato passato a parse_prizes_from_routes (3.10.1) per ottenerne tutte le informazioni.
Successivamente, per ogni percorso, controlla se sono stati completati gli obiettivi obbligatori di ciascun premio, e nel caso questo viene inserito nella lista di quelli che l'utente può vincere per quel percorso (solo se il premio non era già presente nell'elenco di un percorso precedente), insieme alla probabilità di essere vinto.

In seguito, per ogni percorso in cui l'utente può vincere un premio, viene operata una scelta casuale pesata per decidere quale premio assegnarli, chiamando la funzione weighted_choice, che prende in input una lista di tuple (valore, peso) e restituisce il valore determinato. Essa crea una lista di somme progressive dei pesi da passare come parametro alla funzione di libreria python bisect assieme a un numero generato dalla procedura random. L'indice così generato viene usato per scegliere il valore da restituire dalla lista in input.

Per i premi così selezionati viene poi effettuata una query per inserirne il record corrispondente nella tabella Completion_prize e assign_prizes_if_needed termina restituendo la lista dei premi vinti e assegnati.

Infine viene eseguita una query per ottenere tutti i percorsi che contengono l'obiettivo precedentemente verificato, il cui risultato è usato come parametro per la chiamata a get_all_info_from_routes (3.10.1) al cui valore di ritorno è aggiunto l'id dell'obiettivo verificato e la lista dei premi vinti. La risposta avrà quindi lo stesso formato di quella che sarebbe ricevuta da una richiesta /routes (3.10.1), con in più[5]:

```
{
  "verified_obj": int,
  "prizes_won": [{info premio}]
}
```

3.12.2 Riscossione di un premio

Una volta che un utente ha vinto un premio, può ottenerne il codice di riscossione con una richiesta POST all'URL /prize/collect, fornendo l'id del premio ricevuto a seguito di quella a /obj/verify. Il codice dovrà poi essere consegnato all'operatore entro 5 minuti per poter fisicamente ritirare il premio.

Il JSON atteso è nel formato:

```
{ "id_prize": int }
```

[5]Il significato di info premio è descritto nel glossario (3.4)

Il codice è generato come hash (sha512) della concatenazione tra id del premio, login dell'utente e stringa casuale risultante da os.urandom.
Il suddetto viene inserito nel record corrispondente nella tabella Completion_prize effettuando una query e sarà valido nei 5 minuti successivi alla richiesta.
Se risulta che nessun cambiamento è avvenuto nel database, significa che l'utente non ha davvero vinto il premio e viene inviata una risposta 403 FORBIDDEN. In caso contrario, il codice è restituito in un JSON così formato:

```
{ "validation_code": basestring }
```

Conclusioni

Il Web Service descritto nella tesi si propone come un nuovo strumento utile alle aziende in particolare per incrementare la propria clientela tramite i premi collegati ai percorsi.

Si è discusso inizialmente in che modo utenti e operatori registrino un account mediante la richiesta all'URL /signup, di come eseguano login e logout grazie a /login e /logout e con quale modalità possano modificare le loro informazioni tramite /user/edit.

È stato scritto come si ottengono tutte le informazioni degli itinerari, obiettivi e premi con una richiesta a /routes in maniera geolocalizzata indicando la città o le coordinate di interesse.

Si è poi illustrato in che modo gli operatori possano inserire questi elementi tramite le richieste /route/add, objs/add e /prize/add, e come gli utenti, nel percorrere i tragitti, forniscano la prova di aver attraversato gli obiettivi inviandone il codice di validazione a /obj/verify, ottenendo le informazioni sugli itinerari da essi composti ed eventualmente dei premi vinti.

Infine, si è parlato di come gli utenti possano riscuotere i premi effettuando una richiesta all'URL /prize/collect ricevendo un codice di verifica, e si è descritto in che modo gli operatori controllino tale codice tramite /prize/verify.

Le possibili estensioni sono molteplici, a partire dall'implementazione di un meccanismo per l'autenticazione delle applicazioni che fanno uso del Web Service (ad esempio utilizzando il protocollo OAuth), e oltre a questo supportare lingue multiple, permettere agli operatori di indicare più luoghi dove riscuotere i premi, rendere loro possibile la fruizione di percorsi comuni e la creazione obiettivi privati, e infine introdurre una modalità di competizione tra gli utenti per ottenere premi specifici.

Le possibili estensioni non sono solo implementative, ma anche del tipo di *uso* che viene fatto del servizio. Un operatore potrebbe, ad esempio, creare un percorso non turistico ma di caccia al tesoro, prendendo spunto dall'app **Geocaching** (capitolo 1), associandovi premi speciali.

Bibliografia

[1] Bucaresti, C., "Applicazioni mobili in ambito turistico: stato dell'arte, tassonomia e valutazione di casi di studio", Università di Bologna, 2014.

[2] DeLuca, K. (2015). "Selling or Spying: The Legal Implications of Target Marketing Through Geolocation Technologies", *Law School Student Scholarship*, Paper 656, 2015, 2-28

[3] Deterding, S., Dixon, D., Khaled, R., Nacke, L., "From game design elements to gamefulness: defining gamification", *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, 2011, 9-15.

[4] Dustman, A., "MySQLdb: a Python interface for MySQL", 2014, mysql-python.sourceforge.net/MySQLdb.html, 2015-08-31.

[5] Haines, E., "The dos and don'ts of proximity marketing in sponsorship", *Journal of Sponsorship*, 2(2), 2009, 113-119.

[6] Hamari, J., Koivisto, J., Sarsa, H., "Does gamification work? – a literature review of empirical studies on gamification", *System Sciences (HICSS), 2014 47th Hawaii International Conference on. IEEE*, 2014, 3025-3034.

[7] Negri, S., "Il Marketing Di Prossimità", Università degli studi di Pisa, 2015.

[8] Ronacher, A., "Flask documentation", 2014, flask.pocoo.org/docs/0.10/, 2015-08-31.

[9] Ruzic, D., Bilos, A., Kelic, I., "Development of mobile marketing in croatian tourism using location-based services", *Tourism and Hospitality Management*, 2012, 151-159.

[10] Sciortino, F.S., "Gamification: uno strumento per il miglioramento della brand image di una città e della sua attrattività turistica", Università degli studi di Pisa, 2014.

[11] Tussyadiah, I. P., "A concept of location-based Social Network Marketing", *Journal of Travel Tourism Marketing*, 29(3), 2012, 205-220.