ALMA MATER STUDIORUM - UNIVERSITA'
DI BOLOGNA CAMPUS DI CESENA

SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE
INFORMATICHE

SVILUPPO DI UN APPLICATIVO PER LA GESTIONE DI EVENTI

Relazione finale in

Mobile Web Design

Relatore

Prof.
Mirko Ravaioli

Presentata da

Marcantognini Luca

Sessione III
Anno Accademico 2013/2014

Indice

1 Introduzione

Negli ultimi dieci anni con l'avvento degli smartphone abbiamo assistito ad un radicale cambiamento nella tecnologia passando da dispositivi specifici come cellulari e PDA ad altri , gli smartphone, che ne unissero le capacità, integrandole assieme ad un altra serie di funzionalità come l'accesso a internet, riproduzione multimediale, fotocamera, geolocalizzazzione e tutto ciò che da esso è derivato.

Da subito questi dispositivi hanno riscosso un enorme successo conquistando una porzione di mercato sempre più consistente fino a superare le vendite di PC.

Molti infatti utilizzano il proprio smartphone per svolgere compiti per i quali prima era necessario usare il PC.
Gli smartphone sono diventati tanto sfruttati che tutti i maggiori produttori di smartphone hanno implementato un assistente vocale, seppur con notevoli differenze, nei loro sistemi.

Questa evoluzione è in parte dovuta all'integrazione sempre più numerosa di sensori (soprattutto nei dispositivi di fascia alta) che vanno a coprire le funzionalità più disparate come giroscopio, barometro cardiofrequenzimetro, lettore di impronte digitali eccetera.

1.1 Sistemi a confronto

A questi dispositivi in costante ascesa bisogna associare un sistema operativo che stia al passo con l'evolversi delle esigenze dell'utenza.
I sistemi che attualmente dominano il mercato sono:

- iOS
- Android
- Windows Phone

iOS

Iniziamo partendo da iOS poichè è stato il primo dei tre sistemi ad aver raggiunto il pubblico in veste simile a quella attuale.
iOS è stato ed è il sistema presente su tutti i dispositivi mobile targati Apple.
Ha ricevuto molti aggiornamenti fino ad arrivare all'attuale versione che è la numero 8, uscita assieme ad iPhone 6.
iOS è stato anche il primo dei tre sistemi ad aggiungere un assistente vocale, chiamato Siri, che interagisce con l'utente per mezzo di una voce, similmente ad una conversazione.

Android

Android è il sistema per dispositivi mobile attualmente più diffuso al mondo.
É presente su dispositivi di molti produttori e con fasce di prezzo e caratteristiche molto varie, incluse diverse personalizzazioni del sistema da parte della casa produttrice del dispositivo, che rallentano l'aggiornamento del sistema portando ad avere device con versioni del sistema piuttosto datate.

Questa diversià è il motivo della frammentazione del mondo Android che porta chi sviluppa le app a dover tener conto di queste differenze allungando quindi i tempi di sviluppo.

Windows Phone

L'ultimo dei tre sistemi ad essere giunto sul segmento smartphone è in realtà il piu vecchio dei tre sistemi in quanto alle sue prime versioni veniva montato su molti PDA.
Sviluppato da Miscrosoft attualmente lo si trova su tutti i dispositivi Nokia Lumia in quanto prodotti in collaborazione. Il sistema nella sua veste grafica ricorda molto l'interfaccia Metro della versione PC e Xbox One del sistema.
Questa scelta è dettata dalla volontà di Microsoft di costruire un ecosistema attorno ai propri prodotti come fatto da Apple.

1.2 Scelta della piattaforma

Il sistema scelto per lo sviluppo dell'app è Android.
Questa decisione è stata presa in base alla diffusione dei sistemi e a disponibilità tecnologiche.
Allo stato attuale infatti Android ha circa l'85% della quota di mercato, seguito da iOS al 12% e Windows Phone con il 3%.
Questi dati si riferiscono alle vendite dei dispositivi di fine 2014.

Se andiamo analizzare il numero di richieste di pagine web generate da dispositivi mobile scopriamo che Android, nonostante la diffusione superiore, genera solo il 5% di richieste più di iOS (47% vs. 42%).

Questo sta ad indicare che gli utenti che scelgono Apple siano più attivi, in media, su internet di quanto lo siano quelli che scelgono Android.

Per quanto riguarda le esigenze tecnologiche, per lo sviluppo di app per iOS è quasi indispensabile Xcode , disponibile solo per Mac OSX , che ha portato alla scelta di Android come piattaforma di sviluppo dato che il suo amiente di sviluppo è disponibile sia per Windows che per Mac che per Linux.

1.3 App simili sullo store

Cercando sul Play Store le applicazioni che si pongono sullo stesso segmento di mercato sono davvero poche e sono:

- Eventa
- Eventevo
- Where's Up

Mentre le prima due permettono solo la ricerca o poco più Where's Up è molto più simile a Around.

Mentre le altre due sono solo un'interfaccia di ricerca ma necessitano comunque della versione web per l'inserimento degli eventi, la versione mobile di Where's Up è sufficiente a svolgere tutti i compiti. Dato che dei tre è l'app più completa e svolge tutte le funzioni delle altre due verrà presa per il confronto solo l'ultima.

Se le funzionalità di base tra Where's Up e Around sono le stesse (partecipazione a eventi, ricerca ecc.) esse differiscono molto sulla

pubblicazione di un nuovo evento e la sua gestione.
Questa differenza è dovuta ad una diversa visione del prodotto che dal lato di Around prevede una pubblicazione degli eventi da parte di un locale, mentre dal lato di Where's Up qualunque utente può creare i suoi eventi.

Se la necessità di pubblicare gli eventi come locale possa sembrare una limitazione o un inutile layer aggiuntivo, ciò permette di evitare la pubblicazione di eventi fake presso il proprio locale e di ripartire automaticamente la gestione degli eventi tra più utenti semplicemente inserendoli in una lista di PR.
Permette inoltre di non dover inserire i dati del locale ad ogni nuovo inserimento.

Questa differenza comporta però l'utilizzo per la pubblicazione degli eventi ad uso praticamente esclusivo dei locali i quanto non è possibile per un utente pubblicare un evento scollegato da essi.
Altra differenza da Where's Up è la possibilità per i locali di gestire le prenotazioni e per gli utenti di creare bus per eventi e relative prenotazioni.

Funzionalità non presenti su Around che sono invece presenti su Where's Up sono l'integrazione con i social network (in questo caso Facebook), interfaccia web e la possibilità di creare eventi per periodi non contigui.

2 Progettazzione

2.1 Scopo dell'App

L'applicazione si pone l'obbiettivo di facilitare la ricerca di eventi, permettendo agli organizzatori di promuovere le proprie attività.

2.2 Introduzione agli argomenti trattati

L'app vuole essere uno strumento per facilitare la localizzazzione di eventi e locali per il proprio tempo libero.
Se l'utilità può risultare limitata in un ambiente a noi familiare come la nostra città, si può dimostrare molto più utile ad esempio in vacanza o in qualunque altro luogo con cui non si ha familiarità.
Parte fondamentale del sistema sono utenti e locali.
Gli utenti sono gli utilizzatori e i creatori di contenuti mentre i locali sono la base della struttura su cui si regge l'intera applicazione essendo necessari alla pubblicazione di un evento e a quanto ne segue direttamente.

2.3 Percorso di progettazione

La progettazione è avvenuta partendo dall'analisi delle funzionalità che si volevano rendere disponibili in un quadro generale.
Ciò ha portato ad avere una visione più chiara della situazione da affrontare e di distinguere i seguenti oggetti di base:

- Utenti
- Locali
- Eventi

- Prenotazioni
- Bus

La sequenza di passi di progettazione è stata divisa in quanto segue:

1. Analisi delle funzionalità
2. Progettazione della basi di dati
3. Triggers per tabelle Oauth
4. Scelta di Software e tecnologie
5. Progettazione routing
6. Progettazione schermate App

2.4 Analisi delle funzionalità

Il software in questione deve permettere la registrazione di nuovi utenti nonché l'inserimento dei dati basilari.
Deve inoltre permettere la creazione del locale con i dati necessari e la pubblicazione di eventi da parte dei gestori.
Gli eventi devono riportare il locale presso il quale avranno luogo nonché la data e l'ora sia di inizio che di fine, oltre che al prezzo.
Si vuole anche permettere di definire alcuni tipi di prenotazione disponibili per gli eventi che possono essere effettuate dagli utenti.
Deve poi essere data la possibilità agli utenti di organizzare dei pullman per un evento e prenotare uno o più posti per lo stesso.
Di questi devono essere specificate le fermate e il prezzo per posto.
Per ogni locale deve poi essere permessa la creazione di PR che si occupino della gestione degli eventi.

Gli utenti non devono avere una divisione fissa in ruoli ma il loro ruolo varia in base al locale selezionato.
Cioè è possibile che un utente sia proprietario in un locale, PR in un altro e semplice cliente in un terzo.

2.5 Progettazione della base di dati

Il primo punto riguardante la progettazione vera e propria riguarda la progettazione del database, cioè la rappresentazione delle entità coinvolte e dei legami che intercorrono tra di loro.
Il database è un software che permette di strutturare le informazioni e collegarle tra loro nonché di effettuare su di esse operazioni, dette query, che permettono l'interrogazione dello stesso o la manipolazione dei dati oltre che alla modifica della struttura del database stesso.
Il modello attraverso cui vengono strutturate e mantenute le informazioni dipende dal tipo di database in esame.
Nel nostro caso utilizzeremo un database relazionale.
Queste funzionalità sono garantite da un software dedicato chiamato DBMS.
Il modello utilizzato per rappresentare le basi di dati in questo progetto è il modello E/R, che è anche il modello attualmente più diffuso.
Il modello E/R (Entity/Relation) è un modello per la rappresentazione concettuale dei dati ad un alto livello di astrazione del quale i principali elementi costitutivi sono:

Entità Rappresentano una classe di oggetti con le stesse proprietà. Ogni istanza rappresenta un singolo oggetto che appartiene alla classe.

Associazione Rappresenta un legame tra due o più entità

Attributi Descrivono le entità o gli attributi. Tutti gli oggetti della stessa entità o associazione hanno gli stessi attributi Gli attributi vengono definiti con in base al livello di dettaglio con il quale vogliamo descrivere l'entità (o l'associazione) della quale fanno parte.

Identificatori É costituito da uno o più attributi dell'entità e deve essere unico all'interno della stessa.

Dal modello concettuale passiamo direttamente alla progettazione logica e alla definizione delle tabelle.
Iniziamo la descrizione delle tabelle e i relativi campi partendo da *user* in quanto è necessaria la registrazione per accedere al resto dei servizi ed è quindi la prima tabella sul quale effettuiamo operazioni e proseguendo con le tabelle da esso dipendeti e con *place* e le sue .

User Comprende informazioni basilari sull'utente, la tabella è organzzata secondo i seguenti campi

nick Nick dell'utente. Identificatore per la tabella

email Email dell'utente. Non sono possibili duplicati

password Password dell'utente per il sistema. Crittografata con BCrypt per ragioni di sicurezza

firstname Nome dell'utente

lastname Cognome dell'utente

birthday Data di nascita dell'utente. Facoltativa. Salvata con il formato yyyy-MM-dd HH:ii:ss

picture Nome del file dell'immagine profilo dell'utente

telephone Numero di telefono dell'utente. Facoltativo. Utile specialmente per contattare i PR.

Tutti i campi della tabella ad eccezione del nick possono essere modificati dopo l'accesso.

Friendship La tabella *friendship* serve a mantenere le richieste di amicizia in attesa e quelle confermate.
La cancellazione di una tupla sul database sta ad indicare una richiesta rifiutata.
La tabella dipende solo da *user* e presenta i seguenti campi:

nickSender Nick dell'utente che effettua la richiesta. Assieme a *nickReceiver* forma la chiave primaria

nickReceiver Nick dell'utente verso il quale viene fatta la richiesta. Assieme a *nickSender* forma la chiave primaria

status Boolean che indica lo stato della richiesta.
0 = In Attesa
1 = Confermata

addDate Data che indica la data di invio della richiesta o dell'accettazione della stessa.

Altre tabelle dipenti da *user* sono *eventPartecipation, busBooking, ticketBooking* ma verranno trattare in seguito in quanto dipendenti anche da altre tabelle.

Place La tabella in questione contiene tutti i dati che descrivono i locali come nome, posizione e informazioni di contatto

idplace Codice identificativo del locale. Chiave primaria

name Nome del locale

street Via

street_num Numero civico

city Città

provState Provincia, stato o altra area amministrativa

country Nazione

email Email del locale. Facoltativo

telephone Numero telefonico del locale. Facoltativo

website Sito web. Facoltativo

type Tipo di locale. Il valore presente corrisponde all'ID nella tabella *placeType*, descritta a seguire.

lat Latitudine del locale. Derivata in base all'indirizzo fornito.

lng Longitudine del locale. Derivata in base all'indirizzo fornito.

placeType Questa tabella contiene i tipi di locali disponibili nel sistema.
I dati inseriti in questa tabella non sono modificabili dagli utenti ma solo interrogabili.
La scelta di questa decisione è dovuta alla necessità di evitare la definizione di tipi inutilmente ridondanti.
La tabella è composta dai seguenti campi:

idPlaceType Identificativo del tipo di locale. Chiave Primaria

description Descrizione testuale del tipo di locale.

PR *PR* è una delle tabelle più importanti poiché memorizza i dati dei PR e permette quindi di sapere a quali è permesso effettuare determinate operazioni su eventi, locali e prenotazioni.
É costituita da i campi

nick Nick dell'utente. Assieme a *idplace* forma la chiave primaria

idplace Id del locale per cui l'utente è PR. Assieme a *nick* forma la chiave primaria.

role Booleano che indica il ruolo.
0 = PR
1 = Proprietario

Event Questa tabella contiene i dati degli eventi. É composta dai seguenti campi:

idEvent Identificativo dell'evento.Chiave primaria

name Nome dell'evento

idplace Id del locale presso il quale ha luogo

timestartStart Ora e data di inizio. Espresso con il formato yyyy-MM-dd HH:ii:ss

timestartEnd Ora e data di fine. Espresso con il formato yyyy-MM-dd HH:ii:ss.

description Descrizione dell'evento.

type Tipo di evento tra quelli messi a disposizione. Contiene l'Id della tabella *eventType* descritta di seguito.

price Prezzo dell'evento. Espresso in euro.

bill Nome del file contenente l'immagine della locandina / manifesto o flyer.

eventType La tabella contiene i tipi di eventi disponibili.
Non è possibile per gli utenti modificare i dati della tabella ma solamente interrogarla.

idEventType Identificatore del tipo di evento. Chiave Primaria

description Descrizione testuale del tipo di evento

Ticket La tabella *ticket* contiene i tipi di prenotazione per l'evento in questione.
É quindi possibile definire più tipi di prenotazione come per esempio possono essere i biglietti per un concerto o un tavolo in un locale di cabaret.
La definizione è quindi lasciata all'utente.
La tabella è composta dai seguenti campi:

idTicket Identificativo per i tipi di prenotazioni. Chiave primaria

idEvent Id dell'evento a cui è associato

type Descrizione testuale del tipo (es: bliglietto)

total Quantità disponibile

price Prezzo unitario. Può differire dal prezzo impostato sull'evento.

TicketBooking La tabella mantiene tutte le prenotazioni effettuate dagli utenti. É costituita dai seguenti campi:

code Codice della prenotazione. Chiave primaria. Può essere usato come seriale per identificare la prenotazione.

idticket Identificativo del tipo di prenotazione a cui appartiene.

nick Nick dell'utente che effettua la prenotazione.

num Numero di prenotazioni effettuate.

date Data in cui è stata effettuata la prenotazione.

eventPartecipation La tabella mantiene la partecipazione agli eventi degli utenti ed è formata dai seguenti campi:

nick Nick dell'utente partecipante

idEvent Id dell'evento al quale si partecipa

rating Voto dato all'evento dall'utente

Bus La tabella mantiene i bus organizzati per gli eventi. Qualunque utente può organizzare un bus verso un evento anche senza essere PR o Proprietario del locale.

idBus Identificativo del Bus. Chiave primaria

idEvent Id dell'evento verso il quale il bus è stato organizzato.

price Prezzo per posto

totalSeats Numero di posti

nickOrganizer Nick dell'utente che ha organizzato il bus.

busBooking La tabella contiene i dati relativi alle prenotazioni per i bus effettuate dagli utenti ed è composta dai seguenti campi:

idBusBooking Id della prenotazione del bus. Chiave Primaria

idBus Id del bus per il quale è stata svolta la prenotazione

nickUser Nick dell'utente che ha effettuato la prenotazione.

numSeats Numero di posti prenotati

busStop La tabella contiene i dati relativi alle fermate effettuate dal bus.

idBusStop identificatore della fermata del bus.

idBus Id del bus che effettua la fermata.

lat Latitudine della fermata espressa in float.

lng Longitudine della fermata espressa in float.

timestamp Data e ora della fermata

Oltre alle tabelle qui riportate nel database sono presenti altre tabelle sfruttate da OAuth per mantenere in memoria token e altre informazioni ma non verranno riportate poiché facenti parte della libreria utilizzata e non progettate personalmente.
A queste tabelle sono stati aggiunti trigger per permettere che periodicamente venissero cancellate le tuple obsolete.

2.6 Scelta di software e tecnologie

Per lo sviluppo del progetto sono state utilizzate diversi software e tecnologie, cercando di realizzare una architettura quanto più simile ad un contesto reale.
Di seguito vengono elencate le tecnologie, i software e i protocolli utilizzati nonché i motivi che hanno portato alla scelta delle stesse.

2.6.1 Android

La scelta del sistema operativo mobile per il quale sviluppare l'applicazione è ricaduta su Android dato che risulta essere il S.O. mobile attualmente più diffuso e la grande disponibilità di materiale come guide, tutorial ed esempi che sono disponibili online coprono la maggior parte dei problemi che si possono incontrare durante lo sviluppo.
La diffusione del sistema è da attribuire in buona parte anche alla quasi assenza di altri grandi competitori dai prodotti di fasci medio-bassa.
Il progetto Android One lanciato da Google per portare smartphone a basso costo (meno di 100€) nei paesi in via di svluppo, permetterà al sistema di aumentare considerevolmente la sua quota di mercato.

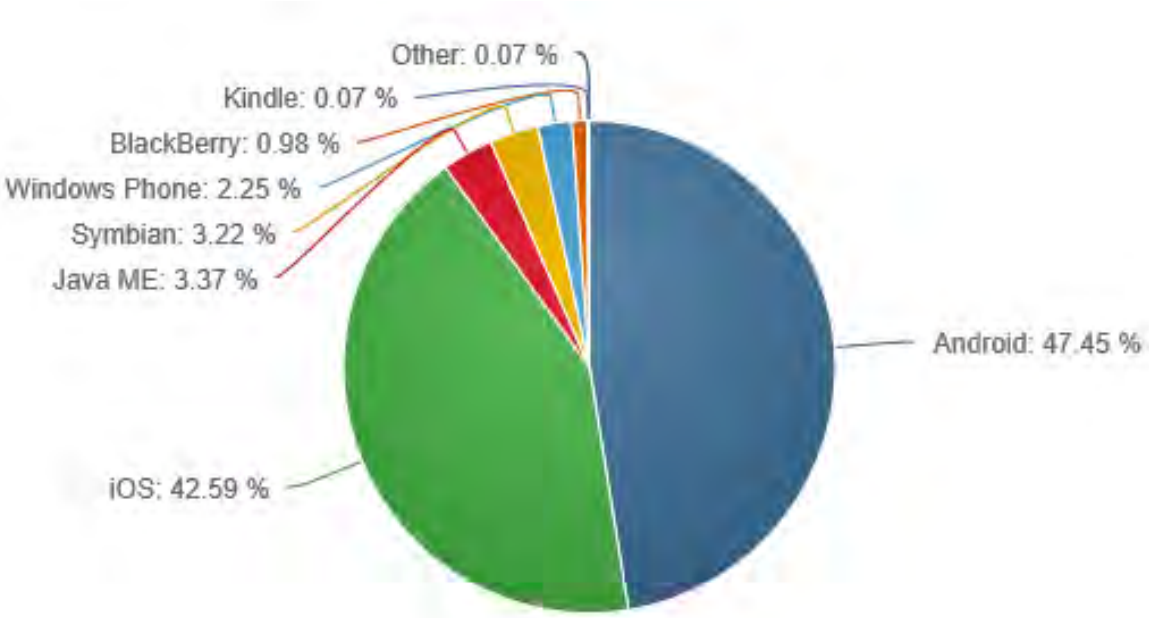


Figura 1: Distribuzione OS Mobile in base al traffico web

La versione minima delle API per cui è stato deciso di sviluppare l'app è la 15, nonostante il codice prodotto possa, almeno per la

maggior parte, essere eseguito su versioni inferiori grazie alla libreria di supporto.
Questa scelta è stata effettuata in virtù della distribuzione delle diverse versioni di Android.

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	7.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	6.7%
4.1.x	Jelly Bean	16	19.2%
4.2.x		17	20.3%
4.3		18	6.5%
4.4	KitKat	19	39.1%

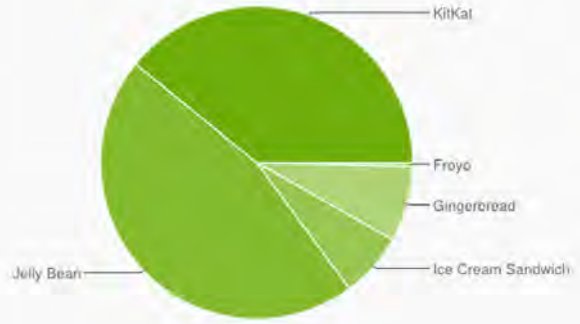


Figura 2: Distribuzioni di Android a Gennaio 2015

Come è possibile notare dalla tabella posta sopra il numero di dispositivi che dispongono almeno delle API 15 sono il 91.8% dei dispositivi Android, questa scelta non segna una limitazione alla diffusione ne alla compatibilità dell'app.
Il linguaggio preposto per lo sviluppo per Android è Java, il quale, stanto ai dati pubblicati da TIOBE è il secondo linguaggio più popolare al mondo con il 15% di market share contro il 6% di Objective-C (per iOS) e il 5,7% di C# (per Windows Phone).
L'indice non rappresenta la reale diffusione del linguaggio ma prende in esame i dati ricavati da Google, MSN e Yahoo! valutando la disponibilità di risorse inerenti il linguaggio in rete.
Va sottolineato che mentre Objecive-C è quasi esclusivo dello sviluppo iOS, ciò non vale per Android e Windows Phone i quali linguaggi sono molto usati anche in altri ambiti.
Ad ogni modo avere a disposizione molte risorse sul linguaggio anche se non specificamente per Andriod, facilita molto il processo di

Feb 2015	Feb 2014	Change	Programming Language	Ratings	Change
1	1		C	16.488%	-1.85%
2	2		Java	15.345%	-1.97%
3	4	▲	C++	6.612%	-0.28%
4	3	▼	Objective-C	6.024%	-5.32%
5	5		C#	5.738%	-0.71%
6	9	▲	JavaScript	3.514%	+1.58%
7	6	▼	PHP	3.170%	-1.05%

Figura 3: Market share dei primi 7 linguaggi valutati dal TIOBE Index

sviluppo del software.

Per quanto riguarda Windows Phone non è ancora riuscito a ritagliarsi una fetta di mercato consistente e perciò anche la quantità di materiale reperibile online risulta inferiore.

Discorso diverso per iOS dato che pur essendo disponibili molte risorse in quanto è riuscito a guadagnarsi la sua fetta di mercato, per lo sviluppo è praticamente indispensabile utilizzare macchine Apple per lo sviluppo.
Inoltre la necessità di dover apprendere un altro linguaggio avrebbe allungato i tempi di sviluppo.

Per queste ragioni e anche per ragioni di preferenza personale è stato scelto Android come target dell'app client del servizio.

2.6.2 PHP

PHP è un linguaggio di programmazione interpretato che viene usato principalmente per lo sviluppo di applicazione web server-side ma può essere usato anche per script da linea di comando o applicazioni standalone.

Come molti linguaggi ha una sintassi C-like il che rende l'apprendimento del linguaggio piuttosto facile e la sintassi risulta familiare anche a chi non ha avuto un esperienza diretta con il linguaggio ma ha già usato linguaggi simili.
Il linguaggio è a tipizzazione debole, cioè il tipo della variabile viene stabilito in base al dato che inseriamo.
Dalla versione 5 in avanti è notevolmente stato migliorato il supporto al paradigma ad oggetti.

La scelta è ricaduta su PHP data l'esperienza pregressa con il linguaggio e la sua diffusione rispetto ai concorrenti nell'ambito delle applicazioni server side.
Risulta infatti dai dati estratti dai motori di ricerca che circa l' 82% dei siti web sia sviluppato con PHP contro il 17% di ASP.NET e il 2,8% di Java. Come scritto in figure alcuni siti web e servizi utilizzano più di un linguaggio di programmazione server-side.

PHP è inoltre il linguaggio con cui sono stati sviluppati in tutto o in parte alcuni servizi web molto famosi come Facebook, Wikipedia, Yahoo!, Flickr e SourceForge.net.

La versione di PHP utilizzata nello sviluppo è la 5.5.
Questa release è stata scelta perché, tra i vari miglioramenti, da questa PHP viene distribuito con Zend OPCache gia integrato.

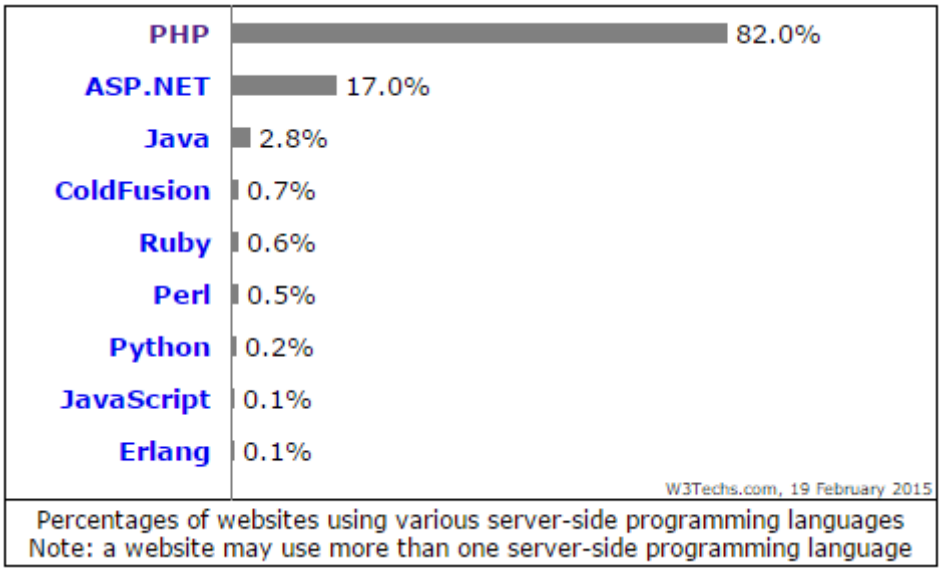


Figura 4: Grafico di diffusione dei linguaggi per applicazioni server side

OPcache è un PHP accelerator, cioè un estensione del linguaggio, o meglio dell'interprete, che tiene traccia del codice gia interpretato e lo mantiene in modo da rieseguirlo senza dover nuovamente interpretare il codice.
Molto spesso questo codice viene mantenuto in memoria centrale, evitando quindi la lettura da disco e riducendo di conseguenza i tempi di esecuzione.

A PHP sono stati affiancati Apache v2.4 e MySQL Server v5.5 data la semplicità con cui collaborano assieme questi programmi, il tutto su una distribuzione Xubuntu 14.04.

2.6.3 Zend Framework 2

Per lo sviluppo dell'applicazione
lato server invece che utilizzare solamente
PHP ci si è appoggiatati ad un framework
che in questo caso è Zend Framework 2.
Il framework
è stato completamente riscritto rispetto
alla prima versione dello stesso per includere
caratteristiche di PHP come i namespace.
Non risulta quindi retrocompatibile con i
progetti scritti per la versione 1 del framework.
Per un corretto funzionamento è richiesta almeno la versione 5.3 di PHP.

Figura 5: Zend Framework 2 Logo

Zend Framework 2 offre un architettura MVC
(Model-View-Controller) che permette di separare la logica
applicativa (Controller) dai dati (Model) e dalla
rappresentazione(View).
Ogni componente è strutturato in modo da avere un numero esiguo di dipendenze e ciò rende possibile l'utilizzo di questi senza la necessità di installare tutto il framework.

Nel framework è inoltre presente un componente chiamato Service Manager che permette di recuperare servizi(oggetti) che sono necessari al funzionamento della nostra applicazione.
Il Service Manager è un implementazione del design pattern Service Locator.
Grazie a questo componente è possibile utilizzare la tecnica dell' Inversion Of Conotrol.

La scelta di utilizzare un framework è stata fatta per concentrarsi sulla logica applicativa lasciando l'implementazione delle chiamate REST al framework.
Inoltre l'utilizzo di un framework obbliga a seguire una certa organizzazione dei sorgenti evitando disordine e rendendo la struttura più ordinata e più facile da mantenere.

2.6.4 OAuth

OAuth non è un software ma un protocollo che permette di accedere o rendere disponibili delle API in sicurezza con un metodo standard.

Grazie a questo protocollo un utente può accedere alle sue informazioni su di un servizio (detto service provider) da un applicazione o un altro servizio, detto consumer, senza condividere la propria identità.

L'autenticazione al service provider viene mantenuta tramite un token che garantisce al consumer l'accesso ai dati dell'utente.
Il token ha in genere un limite temporale dopo il quale deve essere richiesto nuovamente.

Analizziamo le operazioni da effettuare con OAuth per ottenere l'autenticazione.

1. Il consumer invia al service provider una richesta per il request code.
2. Il service provider risponde al consumer con un request code
3. Il consumer redirige l'utente ad una pagina di autenticazione
4. L'utente effettua l'autenticazione sul service provider
5. Il service provider conferma l'avvenuta autenticazione
6. L'avvenuta autenticazione viene inoltrata al Consumer

7. Il consumer richiede il token di accesso al service provider
8. Il service provider genera un token e lo invia in risposta al consumer
9. Con il token il consumer è in grado di accedere alle risorse del service provider
10. Il service provider soddisfa le richieste del consumer solo se il token è valido

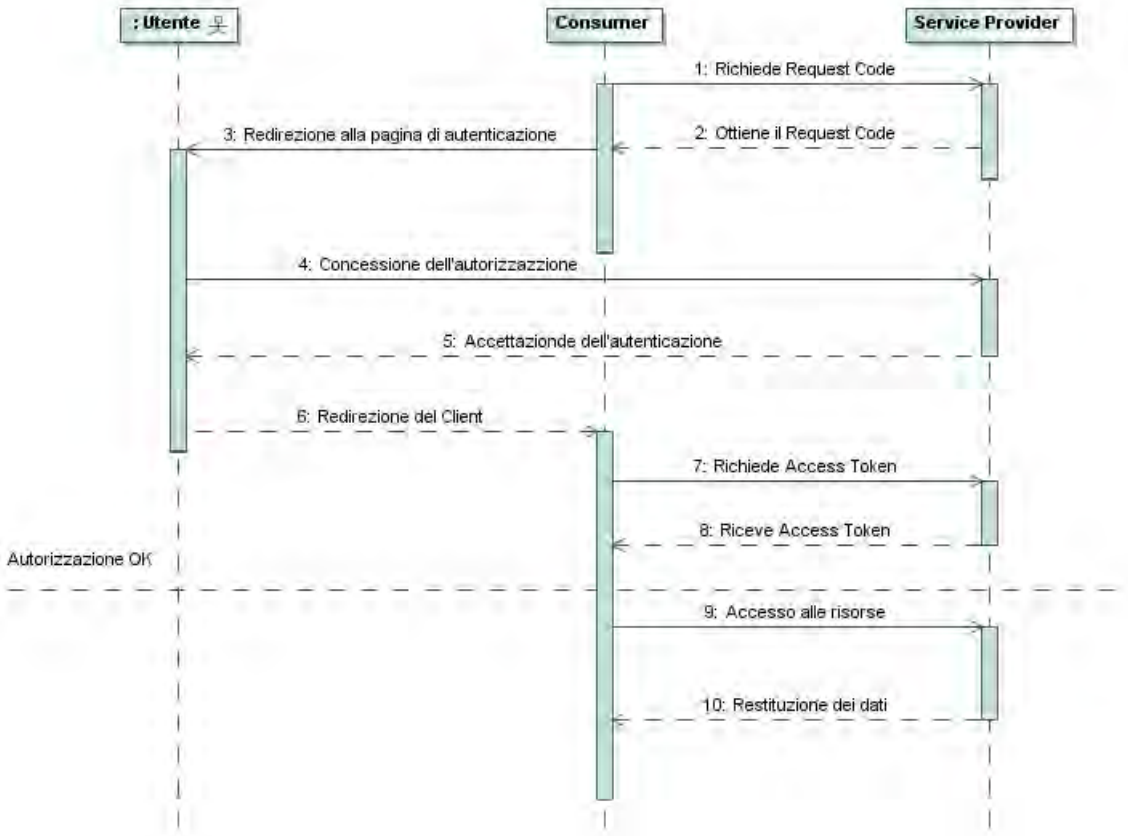


Figura 6: Flusso dei dati di una autenticazione con OAuth

L'access token da solo è sufficente a garantire l'accesso alle risorse ma per garantirne la sicurezza è necessario che la connessione sia utilizzi lo scheme HTTPS in modo da evitare sniffing.

I token possono essere rilasciati anche per una sezione particolare delle risorse, cioè restringendone la validità a determinate funzionalità del sistema e non con validità globale.

2.6.5 REST

REST non è un sistema concreto ne un protocollo ben definito ma un insieme di principi architetturali per la progettazione e l'implementazione di un servizio web.

Principi REST

- Le funzionalità dell'applicazione e il suo stato è suddiviso in risorse
- Ogni risorsa è unica e indirizzabile usando una sintassi universale (es: URI)
- Tutte le risorse condividono la stessa interfaccia per il trasferimento tra client e server, è necessario quindi avere un insieme ben definito e condiviso di operazioni.
- Il protocollo usato per la comunicazione deve inoltre essere
  - Senza stato
  - Client-Server
  - Cachable
  - A Livelli

Applicando i principi REST si ottiene un sistema scalabile che può far evolvere indipendentemente server e client finchè non si modifica l'interfaccia.

Ogni comunicazione tra client e server deve contenere inoltre tutte le informazioni necessarie a permettere l'accesso alle risorse.
La possibilità di fare caching delle risposte può ridurre le comunicazioni client-server, migliorando le performance del server (che non deve elaborare nuovamente i dati) e ridurre quindi i tempi di attesa per il client.

Punto centrale dell'architettura REST sono le risorse.

Per utilizzare le risorse client e server comunicano attraverso un protocollo e scambiano rappresentazioni di queste.
Il client per poter richiedere una risorsa deve conoscere solamente l'identificatore e l'azione da svolgere su di essa.
Deve conoscere anche il formato di scambio utilizzato per la rappresentazione.
L'applicazione sviluppata utilizza HTTP come interfaccia di comunicazione e JSON come formato per lo scambio di informazioni.

Nella sezione seguente verrà descritto come sono state organizzate le risorse e l'URI ad esso associato.

2.7 Struttura delle Risorse REST

L'organizzazione delle risorse è avvenuta dividendo le possibili chiamate in base alle entità principali (Utenti, Locali ed Eventi).
Sono state poi definite altre chiamate di primo livello per funzioni che non riguardavano risorse particolari.

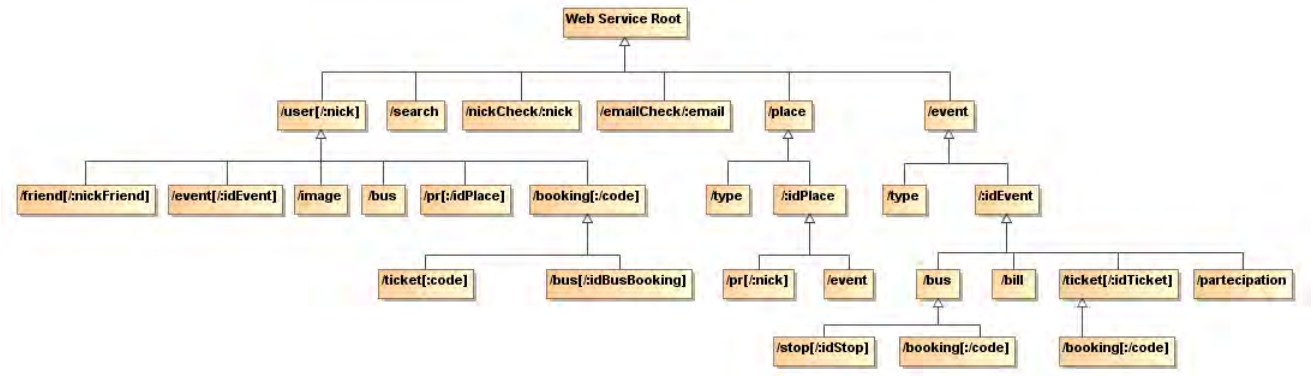


Figura 7: Schema generale dell'organizzazzione dei percorsi REST

La sintassi usata per indicare le componenti dell'URL sia nello schema soprastante che in questa sezione è la stessa utilizzata nello schema di routing di Zend Framework 2.

Le porzioni indicate tra paretesi quadre risultano opzionali mentre i nomi preceduti dai due punti indicano dei parametri.
Si fa però notare che anche se le parti indicate tra parentesi quadre sono opzionali sono necessarie qualora si acceda ad una risorsa di livello inferiore.
Per esempio l'url */user/test* indica la risorsa utente il cui nick è *test*.

Per *place* e *event* il parametro non è stato inserito opzionale poichè è stato riportato in un altro segmento in quanto è presente anche il segmento *type* sullo stesso livello.
Nonstante siano stati inserite divise rappresentano comunque l'Id della risorsa a cui accedere.

Verranno ora esaminate e spiegate tutte le richieste con i relativi metodi.
Verrà riportato anche il grafico delle chiamate per la sezione interessata.

2.7.1 User

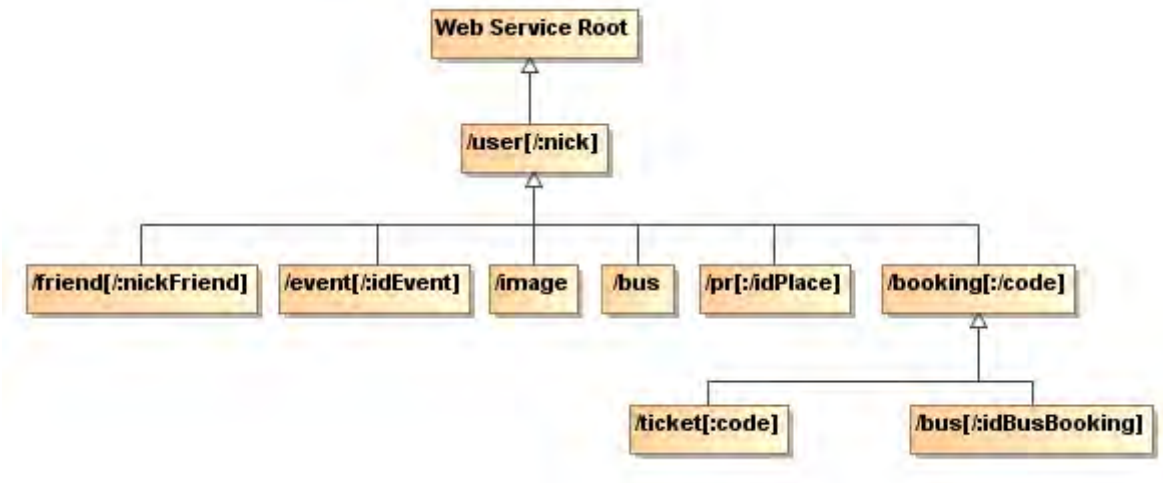


Figura 8: Schema dei path per */user*

/user[/:nick] Indica il profilo di un utente o la collection di questi.

Collezione

POST Permette l'inserimento di un nuovo utente nella collection

Risorse

GET Restituisce la rappresentazione dell'utente

PUT Aggiorna le informazioni dell'utente (tranne l'immagine di profilo)

DELETE Elimina l'utente

/user/:nick/event[:/idEvent] Indica le partecipazioni agli eventi per l'utente specificato in *:nick*

Collezione

GET Restituisce la lista di tutte le partecipazioni

POST Inserimento della partecipazione ad un evento

Risorse

GET Restituisce la partecipazione ad un singolo evento

PUT Aggiorna le informazioni sulla partecipazione (solo il rating)

DELETE Elimina la partecipazione

/user/:nick/pr[/:idplace] Indica i locali per i quali l'utente specificato in *:nick* è PR o proprietario

Collezione

GET Restituisce l'elenco dei locali con relativo ruolo

Risorse

GET Restituisce il ruolo svolto presso il locale

DELETE Elimina il ruolo di PR

/user/:nick/booking Questo path non ha risorse associate ma serve solo a raggrupparne altre

/user/:nick/booking/ticket[/:code] Identifica le prenotazioni dell'utente *:nick* per gli eventi

Collezione

GET Restituisce tutte le prenotazioni effettuate

POST Creazione di una nuova prenotazione

Risorse

GET Restituisce la prenotazione

PUT Aggiorna la prenotazione

DELETE Elimina la prenotazione

/user/:nick/booking/bus[/:idBusBooking] Identifica le prenotazioni dell'utente *:nick* per i bus

Collezione

GET Restituisce tutte le prenotazioni effettuate

POST Creazione di una nuova prenotazione

Risorse

GET Restituisce la prenotazione

PUT Aggiorna la prenotazione

DELETE Elimina la prenotazione

/user/:nick/friend[/:nickFriend] Identifica le prenotazioni dell'utente *:nick*

Collezione

GET Restituisce tutte le amicizie (sia accettate che in sospeso)

POST Invia una richiesta di amicizia

Risorse

GET Restituisce una richiesta di amicizia

PUT Conferma la richiesta di amicizia (solo per il ricevente)

DELETE Elimina la richiesta di amicizia

/user/:nick/bus Identifica i bus organizzati dall'utente *:nick*

Collezione

GET Restituisce tutti i bus organizzati dall'utente

/user/:nick/image Identifica l'immagine di profilo dell'utente *:nick*

Collezione

POST Carica una nuova immagine

2.7.2 Place

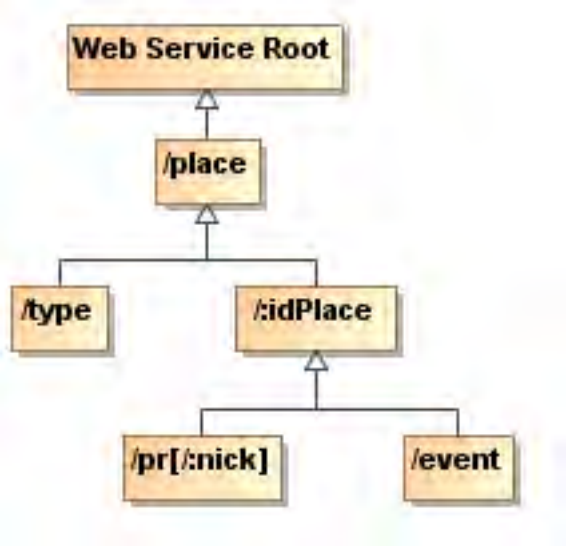


Figura 9: Schema dei path per */place*

/place[/:idplace] Identifica i locali

Collezione

POST Creazione di una nuovo locale

Risorse

GET Restituisce il locale

PUT Aggiorna il locale

DELETE Elimina il locale

/place/:idplace/pr[/:nick] Identifica i pr del locale con ID *:idplace*

Collezione

GET Restituisce tutti i PR del locale

POST Creazione di un nuovo PR

Risorse

PUT Aggiorna il ruolo del PR

DELETE Elimina il PR

/place/:idplace/event Identifica gli eventi dei locali

Collezione

GET Restituisce la lista degli eventi del locale

/place/type Identifica gli eventi dei locali

Collezione

GET Restituisce la lista del tipo di locali

2.7.3 Event

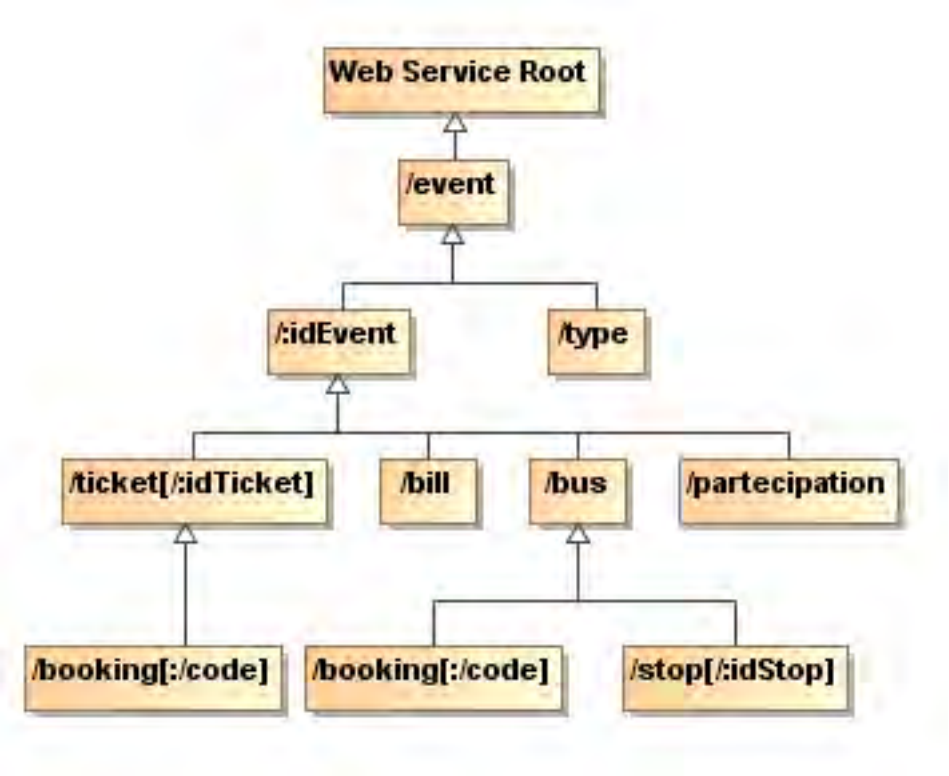


Figura 10: Schema dei path per */event*

/event[/:idEvent Identifica gli eventi

Collezione

POST Creazione di un nuovo evento

Risorse

GET Restituisce l'evento

PUT Aggiorna l'evento

DELETE Elimina l'evento

/event/type Identifica i tipi di eventi

Collezione

GET Restituisce i tipi di evento

/event/:idEvent/bus[/:idBus] Identifica i bus per l'evento *:idEvent*

Collezione

GET Restituisce tutti i bus per l'evento *:idEvent*

POST Crea un nuovo bus

Risorse

GET Restituisce un bus

PUT Aggiorna il bus

DELETE Elimina il bus

/event/:idEvent/bus/:idBus/booking[/:idBusBooking] Identifica le prenotazioni per il bus *:idBus*

Collezione

GET Restituisce tutte le prenotazioni

/event/:idEvent/bus/:idBus/stop[/:idStop] Identifica le fermate per il bus *:idBus*

Collezione

GET Restituisce tutte le fermate

POST Crea una nuova fermata

Risorse

GET Restituisce una fermata

PUT Aggiorna una fermata

DELETE Elimina una fermata

/event/:idEvent/ticket[/:idTicket] Identifica i tipi di prenotazione per l'evento *:idEvent*

Collezione

GET Restituisce tutti i tipi di prenotazioni

POST Crea un nuovo tipo di prenotazione

Risorse

GET Restituisce un tipo

PUT Aggiorna il tipo

DELETE Elimina il tipo

/event/:idEvent/ticket/:idTicket/booking[/:code] Identifica le prenotazioni del tipo *:idTicket* per l'evento *:idEvent*

Collezione

GET Restituisce tutte le prenotazioni

/event/:idEvent/partecipation Identifica le partecipazioni all'evento *:idEvent*

Collezione

GET Restituisce tutte le partecipazioni

/event/:idEvent/bill Identifica l'immagine dell'evento *:nick* (es: locandina)

Collezione

POST Carica una nuova immagine

2.7.4 Altre Chiamate

Concludiamo con listare altre funzioni che non sono state racchiuse sotto un altra risorsa perché sono chiamate di utilità (*nickCheck, emailCheck* o perché appartengono trasversalmente a più di una (*search*).

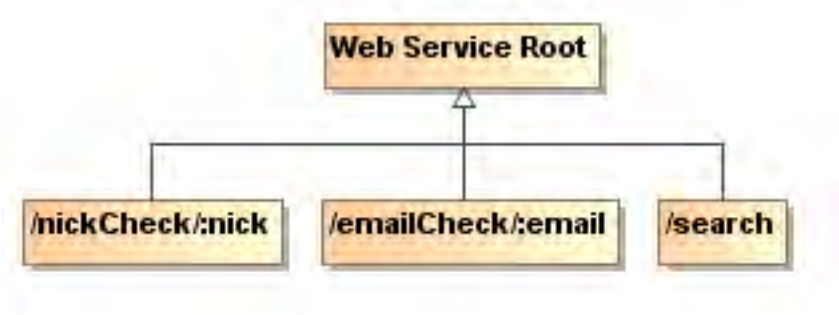


Figura 11: Schema dei path per le altre chiamate

/nickCheck Identifica il controllo per la presenza del nick nel sistema

Risorse

GET Restituisce la presenza del nick nel sistema

/emailCheck Identifica il controllo per la presenza del email nel sistema

Risorse

GET Restituisce la presenza dell'email nel sistema

/search Identifica il controllo per la presenza del nick nel sistema

Collezione

POST Effettua una nuova ricerca

3 Implementazioni

Per l'implementazione inizieremo dalla parte server per poi proseguire con quella client.
Questo approccio è stato scelto per seguire sia l'ordine di sviluppo sia per mostrare prima la parte che espone il servizio in modo da rendere più chiaro come il client lo utilizzi.
Sia per il client che per il server verrà riportato parte del codice sviluppato.

Tuttavia non verrà presentato il codice di ogni componente poiché molti componenti hanno una logica simile (es: controller lato server).

3.1 Lato Server

Per il server verranno esposti i seguenti elementi che coprono tutti gli aspetti del progetto.

- Meccanismo di Autorizzazione
- Routing
- Interazione con il database
- Controller
- Controller genitore
- Ricerca

Nonostante gli ultimi due siano controllor a loro volta sono stati comunque trattati a parte poichè differiscono abbastanza da tutti gli altri controller. // Tralasciando questa eccezione per ogni componente verrà presentato un solo esempio in quanto la logica

applicativa sottostante è molto simile.

Si eviterà inoltre di soffermarsi sui componenti messi a disposizione dal framework dandone una rapida spiegazione per comprenderne il funzionamento solo quando indispensabile.
Per una spiegazione dettagliata dei componenti si rimanda alla documentazione ufficiale del framework.

3.1.1 Meccanismo di autorizzazione

Cuore del meccanismo di autorizzazione è una classe chiamata **AuthController**.
Il metodo *check* messo a disposizione prende in ingresso un oggetto di tipo *Zend\Http\Request* e restituisce un boolean che indica se l'utente è autorizzato ad accedere alla risorsa richiesta.

Per fare ciò viene estratto l'URL dalla oggetto *Request* e diviso in parti.
Si ottiene inoltre il nick associato al token presente nella richiesta come mostrato nel codice sottostante:

```
1 $token = $this::getTokenFromRequest($request);
2         $accessTokenTable = $this->getServiceLocator()->get('
    AccessTokenTable');
3         $this->nickByToken = $accessTokenTable->getNickByToken(
    $token);
4
5         $uri = $request->getUri()->getPath();
6         $uri = substr($uri, 1);
7         $arrayUri = explode('/', $uri);
8
9         $method = $request->getMethod();
```

Tramite un albero decisionale espresso come una serie di switch annidati si arriva all'URL associato e alla verifica tramite alcune funzioni preposte dell'autorizzazione per la risorsa.

```
1 switch ($arrayUri[0]) {
2 ...
3     case 'place':
4         switch ($arrayUri[2]) {
5             case 'pr':
6                 switch ($method) {
7                     case 'GET':
8                         return true;
9                     case 'POST':
10                    case 'PUT':
11                    case 'DELETE':
12                        return $this->isOwner($arrayUri[1]);
13                }
14            case 'event':
15                switch ($method) {
16                    case 'GET':
17                        return true;
18                    case 'POST':
19                    case 'PUT':
20                    case 'DELETE':
21                        return false;
22                }
23            case '':
24                switch ($method) {
25                    case 'GET':
26                    case 'POST':
27                        return true;
28                    case 'PUT':
29                    case 'DELETE':
30                        return $this->isOwner($arrayUri[1]);
31                }
32        }
33    break;
34 ...
```

Il segmento di codice sopra mostrato è la parte dell'albero che riguarda le autorizzazioni discendenti da */place*.
I ... stanno ad indicare altro codice.

Come è possibile notare nelle linee 12 e 30 sono presenti funzioni che controllano la validità dell'autorizzazione dell'utente per una determinata risorsa.
Le funzioni sviluppate sono:

isMe Funzione per la restrizione al solo utente, un esempio di questa limitazione è la modifica del profilo

isFriend Funzione per la restrizione agli amici dell'utente a cui appartiene la risorsa

isPR Funzione per la restrizione agli utenti che sono PR del locale a cui appartiene la risorsa

isOwner Funzione per la restrizione agli utenti che sono proprietari del locale a cui appartiene la risorsa. (Contiene isPR)

isOrganizer Funzione per la restrizione agli utenti che sono organizzatori del bus.

Oltre ai valori restituiti da queste funzioni alcuni percorsi dell'albero possono avere valori statici.

Un valore true significa che la chiamata è pubblica, cioè chiunque può effettuarla.
Un valore *false* indica che la chiamata non è disponibile per nessuno.

3.1.2 Routing

Il routing, cioè l'indirizzamento delle richieste per uno specifico URL verso il controller preposto, viene effettuato sfruttando l'architettura messa a disposizione da ZF2.
Mappare le richieste è estremamente semplice.

Si riporta sotto un frammento del codice.

```
1  return array(
2  ...
3  'router' => array(
4          'routes' => array(
5              'place' => array(
6                  'type' => 'Segment',
7                  'options' => array(
8                      'route' => '/place[/:idPlace]',
9                      'constraints' => array(
10                         'idPlace' => '[0-9]*',
11                     ),
12                     'defaults' => array(
13                         'controller' => 'Place\Controller\Place',
14                     ),
15                 ),
16                 'may_terminate' => true,
17                 'child_routes' => array(
18                     'pr' => array(
19                         'type' => 'Segment',
20                         'options' => array(
21                             'route' => '/pr[/:nick]',
22                             'constraints' => array(
23                                 'nick' => '([a-z]+(\.[a-z]+)?[0-9]*)
    ',
24                             ),
25                             'defaults' => array(
26                                 'controller' => 'Place\Controller\PR
    ',
27                             )
28                         )
29                     ),
30
31                 ...
32
33             ),
34         ),
35     ),
36 ...
37 );
```

Analizziamo il codice sopra mostrato partendo dall'alto.
A riga 3 e 4 troviamo rispettivamente *router* e *routes* che indicano che vogliamo configurare i percorsi che dovranno seguire le richeste.

A riga 5 troviamo *'place'=¿array(...* che è uno dei possibili routing configurati.
place è il nome che assegneremmo a questo routing ma va sottolineato che non è obbligatoria la corrispondenza tra il nome assegnato al routing e l'URL associato.
Questa scelta è stata fatta solo per una questione mnemonica.

All'iterno del route troviamo *type*.
Questo attributo serve a definire il tipo di routing che vogliamo effettuare.
Nel progetto sono stati utilizzati solamente due tipi di routing, cioè Literal e Segment.
Un route Literal è un route il cui percorso sia uguale a quello definito.
Un route Segment permette invece di definire parti opzionali e parametri all'iterno dell'URL.
Prendendo un esempio dal listato sopra abbiamo */place[/:idPlace]*.
In questo esempio abbiamo che */:idPlace* è opzionale, perchè espresso tra quadre e che *idPlace* è un parametro perché preceduto dai due punti.

Quindi alcuni URL che sarebbero accettati dal route in questione possono essere ad esempio:

- */place*
- */place/4*
- */place/30*

Literal è stato usato nei casi in cui l'URL non presentava parametri (ES: */place/type*

Proseguendo troviamo option che contiene tutte le impostazioni del route.
Tra questi troviamo *route*(linea 8), *constraints*(linea 9) e *defaults*(linea 12).
Route è la parte di URL da mappare e la sintassi è stata spiegata sopra.
Constraints contiene la definizione delle espressioni regolari che devono essere soddisfatte dai valori passati come parametri.
In mancanza di parametri come con il tipo Literal è ovviamente superfluo.

Infine *defaults* indica quale controller debba rispondere alla chiamata.
É anche possibile indicare un action di risposta predefinita con *'action'=>'nomeAction'* con *nomeAction* come nome dell'action in caso di necessità.
Nel progetto non è mai stato utilizzatain quanto i controller REST messi a disposizione dal framework stabiliscono da soli l'action appropriata in base al metodo della richiesta.

may_terminate(linea 16) indica se il routing può interrompersi al percorso definito in *route* o debba necessariamente continuare con un route figlio.

child_routes(linea 17) è un array contenente i route figli la cui struttura è identica a quella già esposta.

Oltre al metodo qui esposto è possibile configurare il router anche con un approccio ad oggetti tramite il linguaggio PHP.
Questo metodo risulta però meno prolisso e più facilmente gestibile.
Indipendentemente dalla scelta le possibilità sono le stesse.

3.1.3 Interazione con il database

L'interazione con il database costituisce il model dell'applicazione server-side.
L'interazione con ogni tabella del database avviene attraverso una classe appositamente definita.
Ognuna di queste classi viene aggiunta al service manager tramite file di configurazione in modo da evitare di doverla istanziare più volte.

Anche i parametri di connessione al database vengono aggiunti al service manager tramite il file di configurazione globale *global.php*.
Riportiamo quindi il file di configurazione.

```
1  return array(
2      'db' => array(
3          'username' => 'root',
4          'password' => 'password',
5          'driver' => 'Pdo',
6          'dsn' => 'mysql:dbname=Around;host=localhost',
7          'driver_options' => array(
8              PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\''
9          ),
10     ),
11 ...
12 );
```

La definizione del model avviene creando due servizi nel service manager, come illustrato dopo.

Dopo aver definito questi servizi questi saranno disponibili per l'intera applicazione e quindi richiamabili da ogni parte di essa.

```
1  return array(
```

```
2       ...
3       'BusTable' => function ($sm) {
4           $tablegateway = $sm->get('BusTableGateway');
5           $table = new \Event\Model\BusTable($tablegateway);
6           $table->setServiceLocator($sm);
7           return $table;
8       },
9       'BusTableGateway' => function ($sm) {
10          $dbAdapter = $sm->get('Zend\Db\Adapter\Adapter');
11          $resultPrototype = new \Zend\Db\ResultSet\ResultSet();
12          $resultPrototype->setArrayObjectPrototype(new \Event\Model\
    Bus());
13          return new \Zend\Db\TableGateway\TableGateway('bus',
    $dbAdapter, null, $resultPrototype);
14      },
15  ...
16  );
```

Non è obbligatorio dividere in due parti il model ma permette il riutilizzo del gateway senza la necessità di doverlo ridefinire.

Il TableGateway è il servizio che permette l'iterazione vera e propria con il database.
Al suo interno viene impostato quale tipo di oggetto ritorneranno le query SELECT.
Per fare ciò è però necessario che i nomi dei campi in tabella e quelli della classe siano uguali.

Il Tablegateway permette inoltre l'interrogazione del database usando una sintassi ad oggetti per la composizione della query(vedi listato successivo).

L'oggetto Table si occupa invece di mettere a disposizione metodi che incapsulano le query di TableGateway.

Si riporta il metodo *getAll()* della class *FriendshipTable* che illustra quanto esposto prima.

```
1     public function getAll($nick, $columns = null){
2
3         $select = new Select('friendship');
4         $select->where
5             ->equalTo('nickSender', $nick)
6             ->or->nest
7             ->equalTo('nickReceiver', $nick)
8             ->unnest();
9         $select->order("status", Select::ORDER_DESCENDING);
10        try {
11            $rowset = $this->tableGateway->selectWith($select);
12        } catch (\Exception $e) {
13            $e->getMessage();
14            return false;
15        }
16        $i = 0;
17        foreach ($rowset as $row) {
18            $resultArray[$i] = get_object_vars($row);
19            $rowArray = get_object_vars($row);
20            if (strcmp($rowArray['nickSender'], $nick) == 0) {
21                $nick = $rowArray['nickReceiver'];
22            } else {
23                $nick = $rowArray['nickSender'];
24            }
25            $friend = $this->serviceLocator->get('UserTable')->get(
    $nick);
26            unset ($friend->password);
27            $resultArray[$i]['details'] = $friend;
28            foreach ($rowArray as $field => $value) {
29                if (!is_null($columns)) {
30                    if (!in_array($field, $columns)) {
31                        unset($resultArray[$i][$field]);
32                    }
33                }
34                // "status" nel database viene salvato come INT(1)
35                if ($resultArray[$i]["status"]) {
36                    $resultArray[$i]["status"] = true;
37                } else {
38                    $resultArray[$i]["status"] = false;
39                }
40            }
41
42            $i++;
43        }
44        return $resultArray;
45
46    }
```

Nel codice di esempio sopra si può notare come vengano aggiunte informazioni provenienti da altre fonti(un altra tabella del database) al risultato della query.

In sintesi la classe Table astrae l'acquisizione dei dati mentre la classe Tablegateway si occupa del reale reperimento degli stessi.

3.1.4 Controller

I controller sono i componenti che svolge la business logic dell'applicazione, cioè quella parte che riceve l'input , modifica lo stato del sistema ed elabora una risposta.

Nel progetto abbiamo un controller per ogni URL definito in fase progettuale (Vedi Struttura delle Risorse REST).

Il ciclo di vita del controller viene gestito completamente dal framework lasciando più tempo alla definizione della logica applicativa.

Per creare un controller si deve estendere una delle classi che mette a disposizione il framework.
Nel nostro caso estenderemo *MyRestfulController* che a sua volta estende *AbstractRestfulController*.

Ciò che ci interessa di questa classe sono i metodi messi a disposizione e nel nostro caso effettueremmo l'override di *get()*, *getList()*, *update()*, *create()* e *delete()*.

Le funzioni sono mappate ai metodi HTTP come segue:

get() GET su risorsa

getList() GET su collection

create() POST

update() PUT su risorsa

delete() DELETE su risorsa

Oltre a questi sono disponibili anche le funzioni *patch()*, *head()* e *deleteList()* (DELETE su collection) ma non sono stati usati.

Édisponibile anche *option()* ed è stato definito in *MyRestfulController*.
Riportiamo un esempio di metodo GET gestito da un controller:

```
1  class TicketController extends MyRestfulController
2  {
3      protected $collectionOptions = array('POST', 'GET');
4      protected $resourceOptions = array('GET', 'PUT', 'DELETE');
5
6      function __construct()
7      {
8          $this->identifierName = 'idTicket';
9      }
10
11     ...
12
13     public function get($id)
14     {
15         $ticketTable = $this->getServiceLocator()->get('TicketTable'
       );
16         if (!empty($_GET['f'])) {
17             $filters = explode(',', $_GET['f']);
18         }
19         $result = $ticketTable->get($id, $filters);
20
21         if ($result) {
22             $this->response->setStatusCode(200);
23             $payload = $result;
24             $success = true;
25         } else {
```

```
26                 $this->response->setStatusCode(404);
27                 $payload = array(
28                     'message' => 'Ticket not Found'
29                 );
30                 $success = false;
31             }
32             return new JsonModel(
33                 array_merge(
34                     array(
35                         'success' => $success
36                     ),
37                     $payload
38                 ));
39         }
40 
41         ...
42 }
```

Nel costruttore va modificato il nome del parametro che rappresenta l'id della risorsa.

Quello che si imposta deve infatti essere uguale al parametro definito nel routing per la risorsa.
Prendendo come spunto il codice del routing riportato nella apposita sezione, un controller che avesse voluto accedere a quella risorsa avrebbe dovuto modificare il parametro in questo modo:

```
1 $this->identifierName = 'idPlace';
```

Il controller svolge operazioni di base mappando le operazioni CRUD sul database ed elabornadone le risposte sotto forma di file Json.
É compito del controller inoltre impostare il codice di stato HTTP in base al successo o meno dell'azione.

Si riporta il file Json prodotto dal metodo get() sull'URL di esempio */event/8/bus/3*

```
1 {
2     "success": true,
3     "idBus": "3",
```

```
4       "idEvent": "8",
5       "price": "500",
6       "nickOrganizer": "luca91",
7       "totalSeats": "30",
8       "stops": [
9           {
10              "idBusStop": "3",
11              "idBus": "3",
12              "lat": "43.7001",
13              "lng": "13.2186",
14              "timestamp": "1970-01-01 18:20:00"
15          },
16          {
17              "idBusStop": "20",
18              "idBus": "3",
19              "lat": "43.7072",
20              "lng": "13.2062",
21              "timestamp": "1970-01-01 10:00:00"
22          },
23          {
24              "idBusStop": "22",
25              "idBus": "3",
26              "lat": "43.7293",
27              "lng": "13.1882",
28              "timestamp": "0000-00-00 00:00:00"
29          },
30          {
31              "idBusStop": "23",
32              "idBus": "3",
33              "lat": "43.7134",
34              "lng": "13.1857",
35              "timestamp": "2015-02-24 04:30:00"
36          }
37      ]
38  }
```

Quella riportata è la rappresentazione Json di un bus con relative fermate.

3.1.5 MyRestfulController

Il controller in questione è la classe da cui ereditano tutti gli altri controller.

Serve a definire risposte in formato Json per i metodi di cui non viene fatto override e effettua i controlli di autenticazione e autorizzazione.

Per l'autenticazione viene utilizzato il metodo *verifyResourceRequest* esposto dalla classe *OAuth2Server* messa a disposizione dalla libreria mentre per l'autorizzazione viene effettuata mediante il componente *AuthController* presentato in precedenza.

Questi controlli vengono effettuati per ogni path tranne che per le richeste POST su */user* e le chiamate su */emailCheck* e */nickCheck* dato che il loro utilizzo avviene in fase di registrazione dell'utente e non potrebbe avere quindi un token a disposizione da utilizzare per l'autenticazione.

Si verifica inoltre che le connessioni avvengano utilizzando HTTPS in modo da evitare che tramite sniffing ci si possa impadronire del token altrui.

Tutti questi controlli vengono effettuati nel metodo *onDispatch()* del controller.
Questo metodo viene invocato dal framework, e in particolare dal dispatcher, prima della funzione relativa all'azione invocata(*get()*,*getList()*,*delete()*, ecc...)

Questo permette l'esecuzione dei controlli senza doverli esplicitare in ogni controllor figlie di bloccare l'azione nel caso non si si autorizzati ad eseguirla.

```
1     public function onDispatch(MvcEvent $e)
2     {
```

```
3           if ('https' != $this->request->getUri()->getScheme()) {
4               $this->forbidden("HTTPS is required");
5           }
6
7           $this->oauthServer = $this->getServiceLocator()->get('
    OAuth2Server');
8
9           if((($this->request->getMethod() != 'POST') or ($this->
    request->getUri()->getPath() != '/user')) and
10              (
11                  ((strpos($this->request->getUri()->getPath(), '/
    nickCheck')) !== 0)
12                  and
13                  ((strpos($this->request->getUri()->getPath(), '/
    emailCheck')) !== 0)
14              )
15          ) {
16              if ($this->oauthServer->verifyResourceRequest(
    OAuth2Request::createFromGlobals())) {
17                  $AC = $this->serviceLocator->get('AuthControl');
18                  $result = $AC->check($this->request);
19                  if (!$result) {
20                      $this->notAuthorized("Non hai i privilegi per
    eseguire questa azione");
21                  }
22
23              } else {
24                  $this->notAuthorized('Token non valido');
25              }
26          }
27          return parent::onDispatch($e);
28      }
```

La classe defisce anche il comportamento del metodo *option()* rimandando alle sottoclassi la definizione degli array contenti i metodi accettati.

```
1       public function options()
2       {
3           if ($this->params()->fromRoute($this->identifierName, false)
    ) {
4               $options = $this->resourceOptions;
5           } else {
6               $options = $this->collectionOptions;
7           }
8           $response = $this->getResponse();
```

```
9           $response->getHeaders()->addHeaderLine('Allow', implode(',',
    $options));
10          return $response;
11      }
```

Il controller appena esposto non viene mai istanziato direttamente ma serve solo a definire comportamenti comuni a tutti i controller.

3.1.6 Ricerca

La funzione di ricerca messa a disposizione permette sia una ricerca globale, cioè su tutti i tipi di elementi (Utenti, locali ed eventi) sia la possibilità di ricercare solo in una determinata categoria.

É possibile inoltre effettuare per la ricerca di locali ed eventi un filtraggio in base alla distanza e al prezzo massimo e data di inizio solamente per gli eventi.

La funzione di ricerca inoltre tiene conto della distanza e della preferenza per certi locali (espressa in termini di partecipazioni e voto medio) dell'utente.

Ciò significa che un locale a noi più vicino avrà più importanza di uno più lontano nei risultati cosi come uno che si frequenta maggiormente rispetto a uno che si frequenta meno.

Entrambi questi valori contribuiscono a definire l'importanza di un risultato di ricerca e vengono quindi presentati in ordine decrescente di rilevanza.

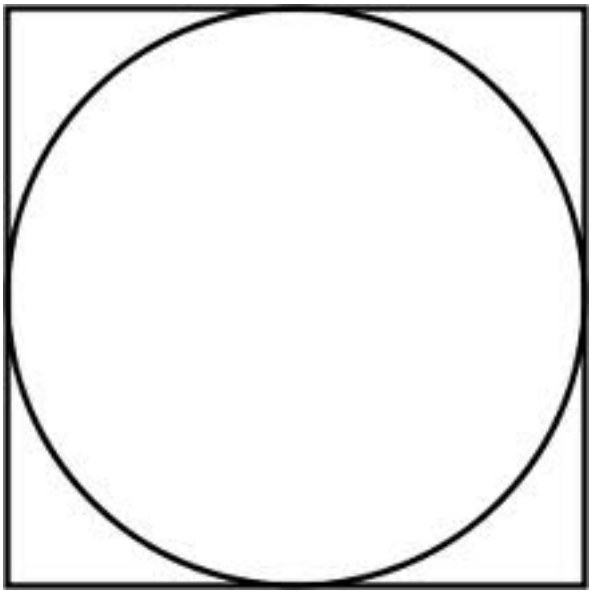
La distanza dall'utente viene calcolata usando la legge dei coseni sferici:

$$\Delta\sigma = \cos^{-1}(\sin\phi_1 \cdot \sin\phi_2 + \cos\phi_1 \cdot \cos\phi_2 \cdot \cos\Delta\lambda)$$

dove ϕ e λ rappresentano rispettivamente latitudine e longitudine del punto in questione.
La formula sopra citata restituisce l'angolo tra le due coordinate rispetto al centro della sfera.

Moltiplicando $\Delta\sigma$ per il raggio terrestre (circa 6371 Km) otteniamo la distanza che separa i due punti.

Per evitare di calcolare questa formula su tutti i locali del pianeta viene prima effettuato un filtraggio eliminando i locali che sono sicuramente troppo lontani.
Calcoliamo perciò la distanza dal centro espressa in latitudine e longitudine massime e minime ottenendo una situazione concettualmente simile a quella in figura. I risultati che otteniamo



dal filtraggio sono rappresentati dal quadrato, applichiamo quindi la

formula di cui sopra e otteniamo i risultati nel cerchio che sono quelli voluti.

Il peso in termini di rilevanza della distanza viene calcolato come:

$$peso(distanza) = \frac{raggio\,di\,ricerca}{distanza\,del\,locale}$$

La rilevanza dei dati di partecipazione vengono invece calcolati come

$$peso(partecipazioni) = 2\log(numero\,di\,partecipazioni)\cdot voto\,medio$$

La funzione logaritmo naturale è stata scelta empiricamente per evitare che partecipazioni frequenti eliminassero la possibilità di concorrenza eclissando gli altri risultati.

Sommando i pesi per distanza e partecipazioni si ottiene il peso totale sul quale poi saranno ordinati i risultati.

Questo tipo di ricerca è stata implementata per dare maggiore rilevanza a quei locali che potrebbero essere più appetibili per l'utente.

3.2 Lato Client

Per lo sviluppo dell'app lato dell'app su Android è stato utilizzato Android Studio.
Il software è un derivato di IntelliJ IDEA, uno dei principali IDE per Java. Viene fornito da Google e include gia l'SDK.

Molto comodo è la presenza di Grandle per la gestione delle dipendenze che ci permette di aggiungere librerie semplicemente indicandole nel file apposito.

Android Studio è recentemente uscito dalla fase beta e la versione corrente è la 1.1.0

I test sull'applicazione non sono stati effettuati utilizzando un emulatore poichè quello fornito con l'SDK risulta poco performante e molto esoso in termini di memoria e si è quindi optato per il test dell'app direttamente su dispositivo.

Il dispositivo impiegato per i test è un Nexus 5 sul quale è presente la versione 5.0.1 di Android Stock.
Tale versione presenta le API di livello 21.

La densità dello schermo è classificata come xxhdpi.

L'utilizzo di un dispositivo fisico ha permesso l'utilizzo di sensori e servizi senza la necessità di emularli.
Nella fattispecie fotocamera, GPS e Google Play Service.

Verranno ora descritte le principali componenti dell'applicazione accompagnati da alcuni screenshot della stessa.

3.2.1 WelcomeActivity

Questa activity viene aperta all'avvio dell'app e permette all'utente di eseguire il login o lo effettua automaticamente in caso venga espressa la volontà dall'utente.

Figura 12: Schermata di inserimento dati per login

L'activity presenta anche un collegamento ad un altra activity che permette la registrazione di un nuovo utente qualora lo si voglia creare.

3.2.2 RegistrationActivity

Grazie a questa activity è possibile registrare un nuovo utente.
I campi da compilare sono divisi in due fragment dei quali il primo contiene i dati dell'utente mentre il secondo permette di scegliere il

nick ed effettua i controlli prima di inoltrare la richiesta di creazione al server.

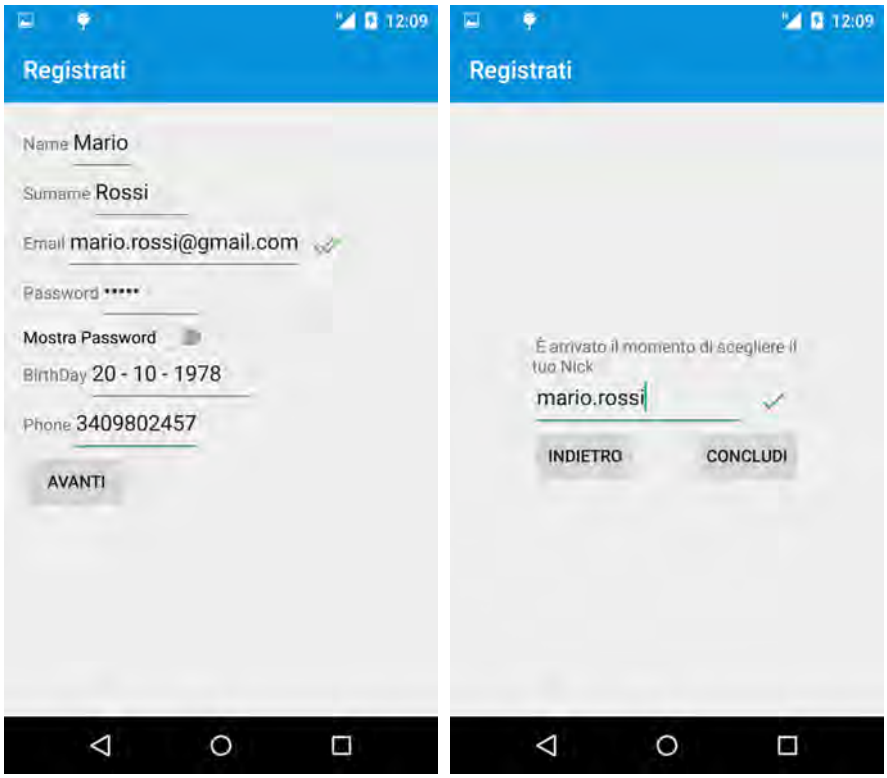


Figura 13: Schermate di Inserimento dati e selezione nick

3.2.3 MainActivity

É l'activity principale dell'applicazione e la prima che incontriamo dopo aver fatto login.
All'apertura mostra una serie di eventi suggeriti ottenuti effettuando una ricerca con i parametri più ampi possibili.

Da questa schermata è possibile spostarsi alle altre sezioni utilizzando il menu inserito nel navigation drawer posto sulla sinistra.
Nel navigation drawer infatti troviamo nella parte superiore la foto dell'utente con nome e cognome.

Cliccando su tale riquadro si accede al profilo utente.

Subito sotto troviamo il menu per accedere ai contenuti oltre alla voce che permette di effettuare il logout.

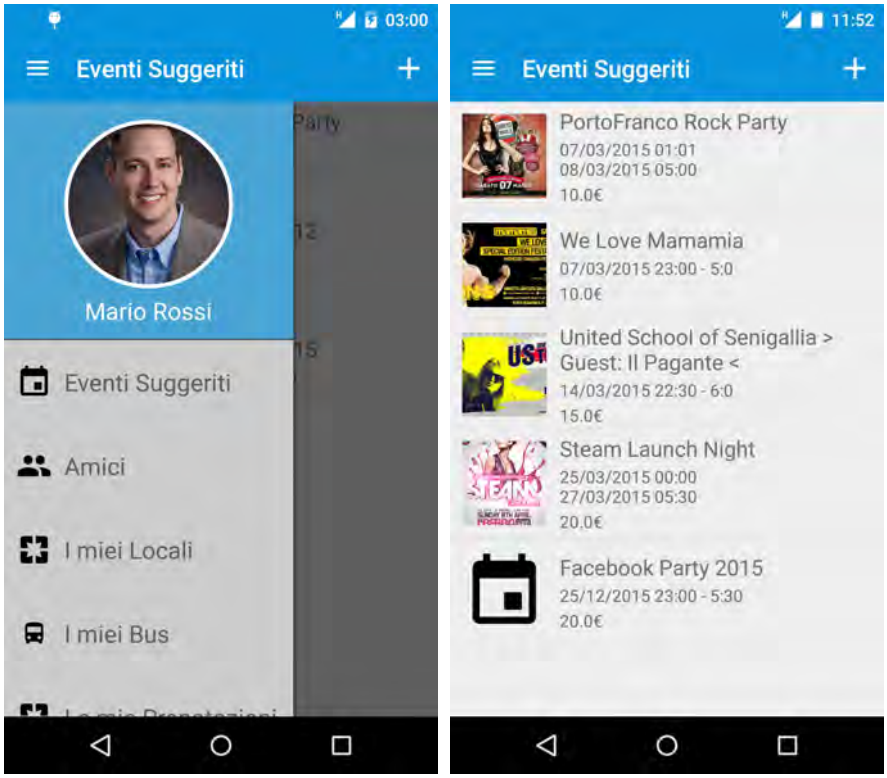


Figura 14: MainAcrivity con NavigationDrawer aperto e suggerimento eventi

3.2.4 FriendFragment

Questo fragment viene caricato da *MainActivity* e mostra lo stato delle richieste di amicizia che riguardano l'utente.
La schermata è composta da una lista con i nomi degli utenti e a fianco un pulsante per eseguire le operazioni inerenti all'amicizia.

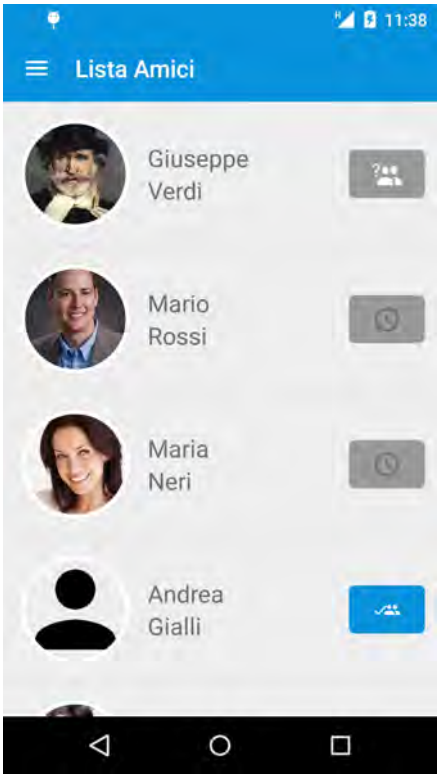


Figura 15: Schermata degli amici

3.2.5 UserDetailActivity

Questa activity ha lo scopo di permettere la visualizzazione del profilo di un utente.

L'icona di modifica che compare nell'ActionBar viene mostrata ovviamente solo per il proprietario del profilo visualizzato.
Cliccando su questa icona si entra in modalità modifica nella quale è possibile modificare i dati del proprio profilo.
Per modificare l'immagine profilo basta cliccare sulla stessa in modalità modifica e un dialog ci permetterà di scegliere se prendere un immagine dalla libreria o scattare direttamente la foto.

Con un click sul segno della spunta, che appare al posto della matita in modalità di modifica, si aggiorna il profilo.

In modalità visualizzazione è possibile cliccando su una delle informazioni di contatto dell'utente aprire l'applicazione per gestirla (Chiamate, mail).

Se visitiamo un profilo diverso dal nostro vedremo anche la presenza di un pulsante che permette di inviare la richiesta di amicizia ad un altro utente.

Figura 16: Profilo dell'utente corrente in modalità visualizzazione

3.2.6 EventDetailActivity

L'activity mostra i dati relativi all'evento selezionato e presenta alche bottoni per l'accesso ad informazioni complementari (PR, Prenotazioni e Bus).

Inoltre da questa activity è possibile indicare la propria partecipazione all'evento ed esprimere un voto oltre a poter consultare la lista di partecipanti.

Figura 17: Dettagli evento in modalità visualizzazione

3.2.7 PlaceDetailActivity

Questa activity mostra le informazioni del locale e permette di contattarlo semplicemente cliccando sopra il campo desiderato tramite telefono o mail o di aprirne il sito internet.

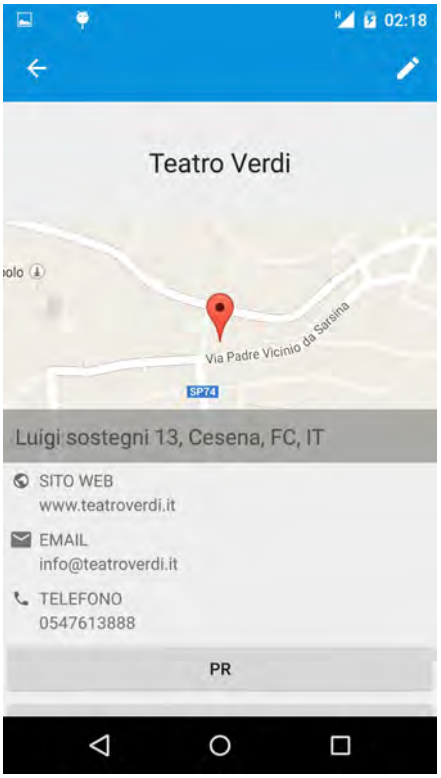


Figura 18: Schermata di dettaglio di un locale in modalità visualizzazione

3.2.8 BusDetailActivity

In questa activity sono mostrate le fermate del bus oltre che il prezzo e i posti disponibili.

Per definire una fermata è sufficente cliccare sul pulsante + posto nell'angolo superiore destro della mappa e verrà aggiunto un pin di colore blu al centro della mappa corrente.

Effettuando un trscinamento del pin nel punto scelto il pin diventerà rosso e l'indirizzo verrà aggiunto alle fermate.

In una lista sottostante vengono infatti riportati gli indirizzi ricavati dalla posizione dei pin definiti.

Per eliminare una fermata è sufficente cliccare sul pin e ancora sulla casella ce si aprirà sopra.

In modalità visualizzazione effettuando un click su una delle voci della lista la mappa si sposterà per fino a centrarsi sul punto scelto.

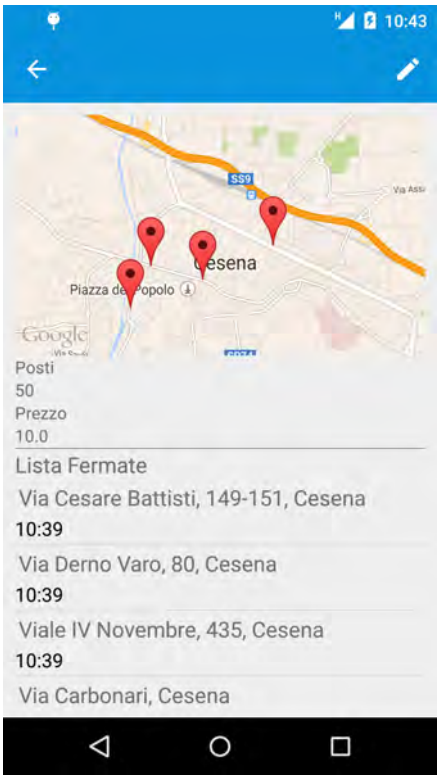


Figura 19: Schermata di dettaglio del bus

3.2.9 SearchActivity

Parte fondamentale di questo tipo di servizio è la funzione di ricerca che deve risultare semplice ed intuitiva e proprio per questo motivo l'interfaccia per la ricerca è stata ridotta al minimo.

L'activity all'apertura presenta infatti solo un campo di ricerca sulla action bar senza particolari controlli.

Sempre sulla action bar è presente anche un icona al cui click viene aperto un dialog con le impostazioni di ricerca.

I risultati di ricerca vengono presentati in un unica lista anche per una ricerca su tutti gli elementi.

Questa scelta è stata fatta per rendere l'interfaccia più user friendly rispetto ad utilizzare una lista per ogni tipo.

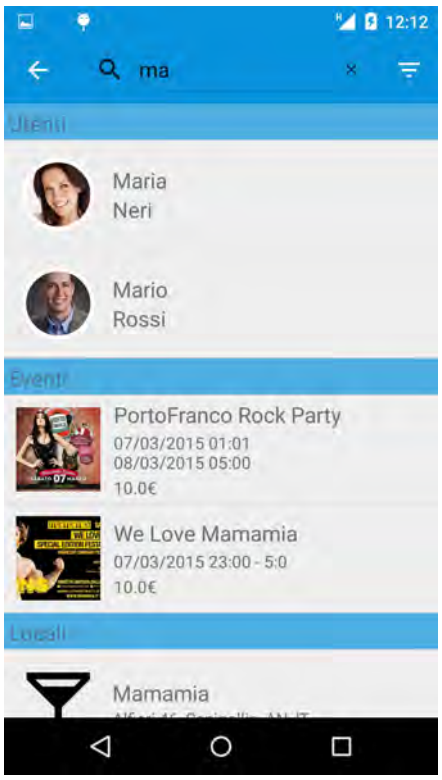


Figura 20: Schermata dei risultati di una ricerca

3.2.10 Altre Activity

Oltre alle activity gia mostrate l'app è composta da numerose altre activity il cui funzionamento cambia però in piccola parte.

Oltre a queste sono presenti altre activity che contengono quasi esclusivamente liste, come quelle riportate sotto.

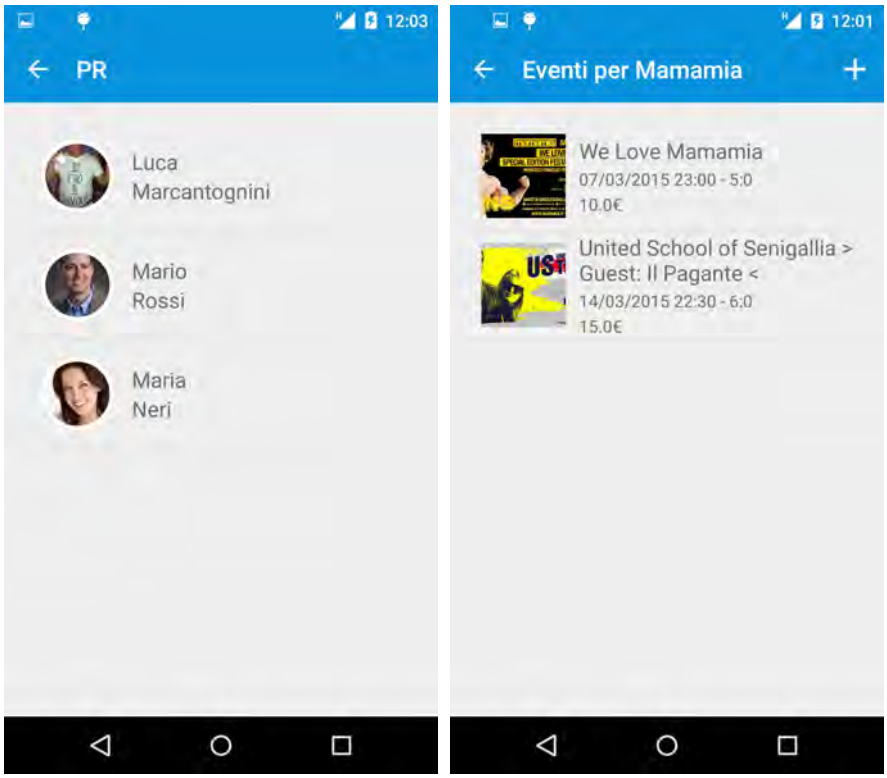


Figura 21: A sinistra: schermata con lista dei PR del locale. A destra: schermata con lista degli eventi del locale

3.2.11 Connessione e Json

Parte fondamentale del funzionamento dell'app è l'accesso ad internet e nello specifico lo scambio di informazioni con il server.

Per l'interazione con il server si sono sfruttate le classi messe a disposizione da Apache che sono già incluse nelle librerie di default.

Nello specifico si è costruita una funzione che prendesse in input un oggetto richiesta e ne restituisse la risposta come *Inputstream*.

A questo punto è bastato definire una funzione per per ogni chiamata al server che contenesse l'oggetto request appropriato e convertisse il file JSON ricevuto in risposta in oggetti Java tramite la libreria Gson con classi costruite ad hoc.

A questo punto è sufficente utilizzare questi oggetti come qualunque altro in Java.

4 Conclusioni

In questa tesi si è cercato di sviluppare un'sistema client-server che illustrasse come fosse possibile svilupparne uno partendo da zero.

Questo progetto mi ha permesso di approfondire tematiche legate allo sviluppo in ambito mobile e ai servizi web oltre ad incrementare le mie capacità di programmazione e la familiarità con i linguaggi utilizzati data la mole di codice prodotto.

Le performance attuali del servizio non sono tali da consentire un reale utilizzo dell'applicazione.
Ciò è dovuto sopratutto alla mancanza di un server dedicato al mantenimento del servizio in quanto attualmente il server viene ospitato su una macchina virtuale del mio computer, che risente molto della congestione di rete e non dispone di una connessione

stabile da consentire la totale continuità del servizio.

Le tempistiche di esecuzione della risposta da parte del server si mantengono su una soglia di qualche decina di ms in misurazioni effettuate in localhost confermando che non è da attribuire al server il tempo di attesa della risposta.

Altro aspetto sicuramente cruciale per una messa in produzione del servizio è la scalabilità che deve sicuramente essere migliorata.

Sul lato client le migliorie da apportare a mio avviso riguardano in larga parte la user experience che andrebbe modificata a seguito di uno studio mirato.

4.1 Sviluppi Futuri

Oltre alle modifiche di tipo perfettivo esposte sopra il servizio dovrebbe beneficiare di espansioni sul lato social che ben si legano al prodotto presentato.

Un integrazione con i social network permettendo in primo luogo il login tramite questi e la ricerca di amici come la possibilità di creare eventi e gestirne l'aggiornamento da entrambi i servizi.

Altre aggiunte potrebbero riguardare l'aggiunta di un sistema di commenti e la possibilità per utenti e locali di caricare foto e video degli eventi taggando gli utenti che hanno come amici.

Il potenziamento della funzione di ricerca per includere altri dati in modo da assecondare le preferenze dell'utente sarebbe un altro punto

da sviluppare come la presenza di un sistema di notifica su aggiornamenti degli eventi a cui si partecipa.

L'inclusione di un sistema di segnalazione per il controllo di comportamenti scorretti e fake di locali all'interno del servizio sarebbe ugualmente importante.

Questi riportati sono solo alcuni delle possibili aggiunte ad un servizio che risulta attualmente basilare e quindi aperto ad una moltitudine di possibili miglioramenti

5 Sitografia

http://w3techs.com/technologies/overview/programming_language/all

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

http://en.wikipedia.org/wiki/Great-circle_distance

http://developer.android.com/reference/packages.html

http://framework.zend.com/manual/current/en/index.html

php.net

http://blog.varonis.com/introduction-to-oauth/

http://en.wikipedia.org/wiki/Representational_state_transfer

http://rest.elkstein.org/