Progetto relativo al corso di Transactional System & Data Warehouse

Unimate: il tuo compagno universitario

Obiettivo

Realizzazione di un'applicazione mobile multi-platform di ateneo, concepita sul modello social, in grado di mettere in comunicazione gli utenti, fornendo un supporto per la ricerca di eventi ed attività.

Rossi Riccardo – 3689831
De Luca Jacopo – 3675883

INDICE

Progetto relativo al corso di
Transactional System & Data Warehouse
(App studenti – Progetto 3)

Corso tenuto dal:
Prof. Boccalatte Antonio

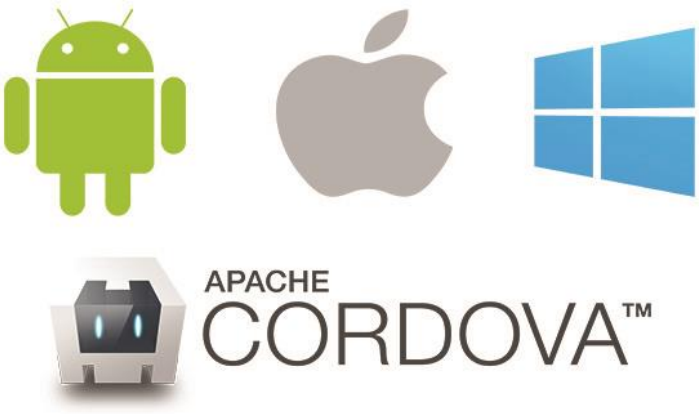
Studenti:
Rossi Riccardo – 3689831
De Luca Jacopo – 3675883

Professori di riferimento al progetto:
Prof. Sacile Roberto
Prof. Parodi Francesco

OBIETTIVO
Realizzare un'applicazione mobile multi-platform di ateneo. Tale applicazione, da concepirsi sul modello social, dovrà essere in grado di mettere in comunicazione Studenti, Professori ed Aziende, fornendo un supporto per la ricerca di eventi ed attività per lo studente.

STRUMENTI
Per realizzare tale applicazione sono stati usati:

- ***Altervista**, spazio web gratuito*
- ***MySQL**, database relazionale open source*
- ***PHPMyAdmin**, applicazione web fornita da Altervista per amministrare il database MySQL*
- ***FileZilla Client**, per il trasferimento dei file in rete tramite protocollo FTP*
- ***Notepad++**, editor di testo open source che supporta la sintassi di vari linguaggi*
- ***HTML5**, utilizzato per costruire lo "scheletro" delle pagine*
- ***CSS3**, fogli di stile per definire la formattazione dei documenti HTML*
- ***Javascript**, linguaggio di scripting lato client per la gestione dinamica delle pagine*
- ***jQuery**, framework sviluppato per rendere il codice più sintetico, nonché di implementare funzionalità ajax*
- ***PHP**, linguaggio di scripting lato server usato come intermediario tra gli script lato client e il server (in quanto non interpretato da Apache Cordova)*
- ***Apache Cordova**, framework per lo sviluppo di applicazioni mobile: permette di creare app mobile usando HTML5, CSS3 e Javascript anziché affidarsi ad API specifiche delle varie piattaforme. Il framework incapsula il codice "traducendolo" in ibrido (tra nativo e web), permettendo ai più comuni OS (Android, iOS, Windows Phone, ecc.) la sua interpretazione, rendendo quindi la app in questione multi-platform*



PREMESSA

Per la creazione dell'applicazione sono stati utilizzati metodi e linguaggi mai incontrati lungo il nostro percorso di studi, pertanto, affrontando ogni volta diverse difficoltà, le nostre conoscenze si sono sviluppate durante la realizzazione del progetto rendendo la leggibilità e riutilizzabilità del codice poco chiara da un punto di vista professionale e tecnico, lasciando probabilmente qualche imprecisione, ma rafforzando le nostre competenze e permettendoci di mettere in pratica altrettanti insegnamenti.

I. DATAMANAGE

Prima di esaminare l'applicazione vera e propria, è importante cominciare da uno strumento sviluppato per simulare il database universitario.

http://datamanage.altervista.org/datamanage/

La pagina sopra è un *form* di registrazione (chiamato *Datamanage*), con il quale l'utente può inserire i propri dati all'interno del database. Questi dati saranno poi recuperati dall'applicazione mobile: è evidente il ruolo di *Datamanage* che dissimula il ruolo dell'università in tutti i servizi online (dal portale *Unige* ad *Aulaweb*), in quanto per accedere all'app saranno necessarie le credenziali (ovvero la *UnigePASS* composta da matricola e password) e l'utente non dovrà iscriversi alla nostra applicazione, bensì dovrà essere un utente già registrato.
Datamanage, quindi, è da considerarsi una "fonte esterna", come richiesto dal progetto, e come tale non ha alcuna funzionalità all'interno dell'applicazione oltre a fornire i dati dell'utente, come sopra descritto.

STRUTTURA DATABASE

Altervista fornisce uno spazio gratuito limitato, permettendo di creare (gratuitamente) un solo database per dominio registrato. Pertanto abbiamo deciso creare tutte le tabelle di *Datamanage* e dell'applicazione nello stesso database, tuttavia dal punto di vista concettuale intendiamo le due basi di dati separate.

Il database di *Datamanage* fa uso di tre tabelle, una per categoria di utente:

- *Studente*

id	nome	cognome	matricola	password	universita	facolta	corso	indirizzo	civ	citta	cap	mail	phone
2	Riccardo	Rossi	s3689831	aaaaaaaa	Unige	Ingegneria	Ingegneria Informatica	Via Ugolini	18	Chiavari	16043	red.riccardo.91@gmail.com	3463088334
3	Jacopo	De Luca	s3675883	bbbbbb	Unige	Ingegneria	Ingegneria Informatica	Corso Valparaiso	90	Chiavari	16043	jacopodeluca.private@gmail.com	3403257414

- *Docente*

id	nome	cognome	matricola	password	universita	indirizzo	civ	citta	cap	mail	phone
5	Andrea	Bianchi	p1234567	aaaaaa	Unige	Corso Europa	3	Genova	16121	andrea.bianchi@unige.it	3498511327

- *Azienda*

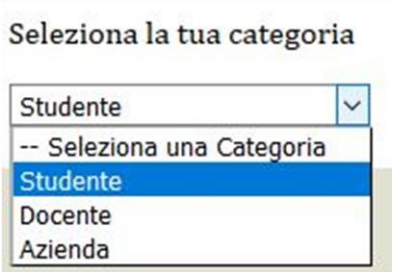
id	nome	piva	password	indirizzo	civ	citta	cap	mail	phone	sitoweb
9	StarBytes	12345678912	wwwwww	Via roma	24	Accettura	14589	strabytes@gmail.com	3408924857	www.star-bytes.com

Per quanto riguarda *Studente* e *Docente*, la chiave primaria corrisponde alla matricola (l'*id* crescente unico è stato inserito per completezza) **S***matricola* per studente e **P***matricola* per docente, mentre la chiave primaria delle *Azienda* corrisponde alla P.IVA di 11 cifre (*piva*).

E' evidente la mancanza di campi a *NULL*, poiché al momento della registrazione sarà necessario per l'utente inserire obbligatoriamente tutti i dati richiesti, a seconda della categoria.

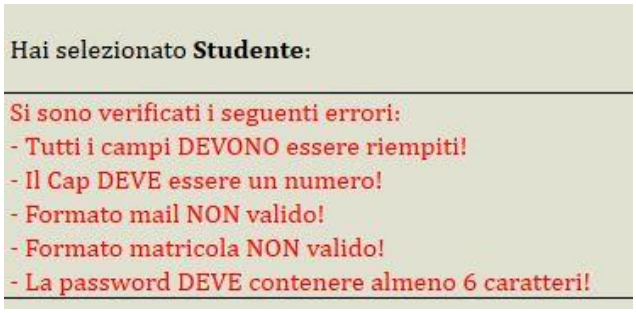
FUNZIONAMENTO

La pagina, molto semplice e intuitiva, richiede subito all'utente di specificare la propria categoria di appartenenza e alla scelta il *form* cambierà dinamicamente i campi da inserire.



L'applicazione è stata concepita per recuperare dati da più università, perciò in *Datamanage* è stato ideato un sistema di *select* dinamiche legate tra loro, le cui opzioni vengono acquisite da file di testo.
Il primo file di testo include i nomi delle università, la *select* delle facoltà legge il testo e crea le opzioni in base alla scelta leggendo un altro file di testo (con lo stesso nome dell'università) contenete le facoltà, in maniera analoga deve esistere un file di testo contenente i corsi (un file per ogni facoltà), in questo modo anche l'ultima *select* crea le opzioni in base alle scelte precedenti. Questo approccio rende possibile cambiare e inserire nuove università, facoltà e corsi di studi lavorando semplicemente sui file di testo.

Per questa pagina è stato implementato un controllo sull'inserimento dei campi.
Al momento del click sul pulsante *Registrati*, uno script controllerà se tutti i campi sono stati inseriti correttamente: innanzitutto verrà controllato se tutti i campi sono stati effettivamente riempiti, successivamente verranno esaminati i caratteri tramite l'uso di *espressioni regolari* per validare le stringhe. In caso di esito positivo, i dati saranno inseriti nel database, altrimenti apparirà all'attenzione dell'utente un messaggio di errore che specificherà i campi inseriti in maniera non corretta.



Ultima nota va al campo *città*: all'interno del server è presente un file di testo contenente il nome di tutti i comuni italiani; l'utente inserisce i caratteri e l'oggetto *datalist* legge dal file e suggerisce i comuni a seconda della stringa inserita.

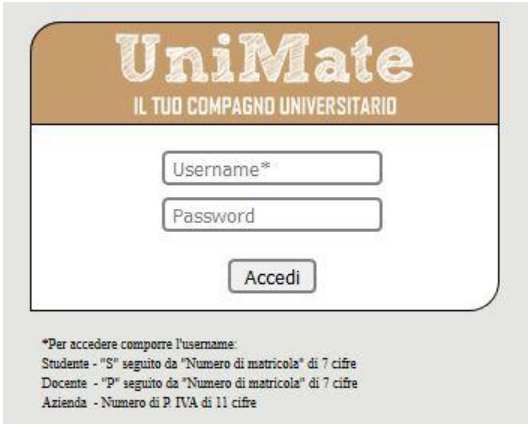
Come verrà mostrato in seguito, per inserire i dati nel database lo script effettuerà una chiamata *ajax* a un file PHP, nel quale si eseguiranno le query opportune.

II. UNIMATE

Vedremo successivamente come l'applicazione viene installata sul dispositivo, utilizzando i comandi di *Apache Cordova*. Per cominciare analizzeremo i funzionamenti dei vari servizi che l'applicazione propone.
Innanzitutto, una piccola nota per quanto riguarda il nome della nostra app. Il progetto nasce come software mobile di supporto alla vita universitaria, secondo un modello social, facilitando la connessione tra gli utenti con l'implementazione di gestione eventi, la ricerca di lavoro per gli studenti e viceversa lo "screening" di possibili dipendenti da parte di aziende. Quindi questa applicazione sarà una sorta di appoggio per gli utenti: per questo motivo abbiamo scelto di chiamarla *Unimate*, accennando a un "compagno universitario".

MODELLO SOCIAL

Per applicazione social si intende un software in grado di mettere in comunicazione le persone, quindi ogni utente condividerà le proprie informazioni con altri. Per poter accedere alla propria pagina personale, all'avvio dell'app si presenta una schermata di LogIn con il compito di riconoscere l'utente e salvaguardare le sue informazioni personali.
Ad accesso eseguito, come in ogni social, si presenta la propria *Home Page*, contenente le informazioni personali, quali Nome, Cognome, Università, Facoltà, Corso di Studi, Città, immagine di profilo ecc. Ovviamente a seconda del tipo di utente (*Studente, Docente, Azienda*), le informazioni mostrate saranno differenti.



In questa pagina si visualizza la propria "storia", ovvero un elenco scorrevole di post relativi alle azioni compiute, ognuno con una tipica icona identificativa, timestamp e una descrizione.
Un aspetto fondamentale dell'approccio social è la creazione di interconnessioni tra utenti: nel nostro modello non si creano amicizie (stile *Facebook*), bensì l'utente sceglie di "seguire" un altro utente in maniera univoca (stile *Instagram*), in tal caso il secondo verrà aggiunto alla rete del follower.

STRUTTURA

In cima e in fondo nella *Home*, come in tutte le altre pagine, rimangono bloccate due barre con le principali funzionalità dell'applicazione con cui l'utente può interagire:

- *Header*, sull'estremità superiore, con i tasti *Home* (per tornare al proprio profilo), *Add Event* (per la creazione di eventi), *Ricerca Avanzata* (per aprire un form di ricerca filtrata), la barra di *Ricerca Generica* (con tasto *Search*) e *LogOut*.

- *Footer*, in fondo alla pagina, con i tasti *Rete* (per la visualizzazione della propria rete di amici), *Eventi* (al cui click si aprirà un pannello per la scelta degli eventi da visualizzare), *Messaggi* (per accedere all'area di messaggistica), *Notifiche* e *Mappa* (per la geolocalizzazione degli eventi).

Per alcune funzionalità (come la *Ricerca Avanzata)* sono state usate delle finestre di *Dialog,* fornite dalle librerie *JQuery UI* (basate su quelle *Javascript*).

III. DATABASE DI UNIMATE

Ovviamente il database costituisce il "cuore" dell'applicazione, governando quest'ultima le informazioni contenute in esso. Abbiamo deciso di utilizzare il database *MySQL*, gestito da *PHPMyAdmin*, entrambi forniti gratuitamente da *Altervista*.

STRUTTURA E ORGANIZZAZIONE

Per amministrare al meglio le funzionalità di *Unimate*, è stato necessario costruire un database composto da varie tabelle, ognuna delle quali immagazzina dati inseriti e richiesti dall'applicazione quando opportuno:

- ***Utenti***

Per gestire gli utenti, abbiamo creato una tabella *Utenti* contenente le relative informazioni. Al momento del primo login (è stato implementato un opportuno controllo), le informazioni esterne di *Datamanage* (delle tabelle *Studente*, *Docente*, *Azienda*) vengono recuperate e inserite nella suddetta tabella. In questo modo non è possibile iscriversi all'applicazione, bensì è necessario, idealmente, essere registrati nel database dell'università.

nome	cognome	matricola	password	universita	facolta	corso	indirizzo	civ	citta	cap	mail	phone	piva	sitoweb	categoria	foto
Riccardo	Rossi	s3689831	aaaaaaaa	Unige	Ingegneria	Ingegneria Informatica	Via Ugolini	18	Chiavari	16043	red.riccardo.91@gmail.com	3463088334	*NULL*	*NULL*	studente	38399.jpeg
Jacopo	De Luca	s3675883	bbbbbb	Unige	Ingegneria	Ingegneria Informatica	Corso Valparaiso	90	Chiavari	16043	jacopodeluca.private@gmail.com	3403257414	*NULL*	*NULL*	studente	78972.jpeg
Andrea	Bianchi	p1234567	aaaaaa	Unige	*NULL*	*NULL*	Corso Europa	3	Genova	16121	andrea.bianchi@unige.it	3498511327	*NULL*	*NULL*	docente	*NULL*
StarBytes	*NULL*	*NULL*	wwwwww	*NULL*	*NULL*	*NULL*	Via roma	24	Accettura	14589	strabytes@gmail.com	3408924857	12345678912	www.star-bytes.com	azienda	*NULL*

Le informazioni vengono immagazzinate in *Utenti* in maniera tale da settare a *NULL* i campi inesistenti (ad esempio Cognome per Azienda), in questo modo tutte le categorie di utente sono contenute in un'unica tabella.
I campi *matricola* e *piva* costituiscono una chiave, particolare la scelta relativa all'utilizzo di questi: uno studente o un docente avrà solo il dato *matricola* mentre *piva* sarà a *NULL*, viceversa per l'azienda che avrà il dato *piva* e *matricola* sarà a *NULL*. Per evitare che vengano inseriti utenti doppi, *matricola* e *piva* sono stati settati come *unici*, così non potranno essere aggiunti alla tabella (e quindi accedere all'app) utenti la cui *matricola* o *piva* sia già stata inserita.

- ***Amici***

All'interno della tabella *Amici* vengono inserite le informazioni relative alle "amicizie": come già detto sarà un utente che segue un altro. La chiave primaria corrisponde all'*id* identificativo dell'amicizia, mentre i campi *user1* e *user2* indicano rispettivamente *matricola* o *piva* del follower e dell'utente seguito.

id	user1	user2
299	s3675883	12345678912
300	S3675883	p1234567
298	s3675883	s3689831
297	s3689831	s3675883

Nel caso in cui si dovessero recuperare informazioni degli utenti da questa tabella, sarà necessario costruire una query alla tabella *Utenti* utilizzando *matricola* o *piva* corrispondenti.
Per quanto riguarda una cancellazione di following, la riga relativa sarà eliminata dalla tabella.

- ***Eventi***

Un aspetto importante dell'applicazione è l'interazione con gli eventi, è stato quindi necessaria la creazione di una tabella apposita.

id	nome	categoria	lat	lon	data	orario	descrizione	proprietario	indirizzo	citta	num_votanti	media_voto
37	Giro Pizza	Ritrovo/Festa	44.315048	9.326684999999998	30/11/2016	21:00	Vi aspettiamo numerosi	s3689831	Via Nino Bixio 6	Chiavari 16043	1	3
38	Ricevimento Studenti	Ricevimento	44.39972239999999	8.96121960000005	9/1/2017	15:30	Gli studenti sono invitati a partecipare in caso d...	p1234567	Via Montallegro 1	Genova 16121	0	0
39	Prova	Offerta Lavoro	44.40223899999999	8.96114799999998	Non Specificata	Non Specificato	Prova	P1234567	Via opera pia 13	Genova 16145	1	4
36	Programmatore Web	Offerta Lavoro	44.314921	9.329655000000002	Non Specificata	Non Specificato	Siamo alla ricerca di un programmatore Web. Compet...	12345678912	Piazza Roma 10	Chiavari 16043	1	4

La chiave primaria è costituita dall'identificativo *id*. All'interno di *Eventi* vengono immagazzinate tutte le informazioni (Nome, Categoria, Proprietario, Indirizzo, Città ecc.); di particolare importanza le coordinate (*lat* e *lon* utili per mappare l'evento), *data* e *orario* (possibili a *Non*

specificata/o nel caso di eventi ricorrenti, come i servizi universitari e lezioni), *descrizione* (importate per incentivare l'interesse degli utenti) e le colonne sulle votazioni (*num_votanti* e *media_voto*).

- ***Messaggi***

Per quanto riguarda la messaggistica, i dati relativi ai messaggi tra utenti sono immagazzinati nella tabella *Messaggi*.
Come chiave primaria è stato utilizzato il campo *id*; le altre informazioni sono *mittente*, *destinatario*, *testo* del messaggio e *stamp*. Molto importante il campo *visualizzato*, che verrà usato per la gestione delle relative notifiche (vedremo successivamente).

id	mittente	destinatario	testo	visualizzato	stamp
57	s3689831	S3675883	ciao	si	2016-12-22 10:26:46
56	S3675883	s3689831	Ciao	si	2016-12-22 10:26:16
54	s3675883	s3689831	Tutto bene. Grazie.	si	2016-12-21 16:44:17
55	s3689831	p1234567	Buongiorno. Vorrei sapere quando Ã¯ disponibile pe...	si	2016-12-21 17:17:12
53	s3689831	s3675883	Ciao Jacopo. Come stai?	si	2016-12-21 16:43:42

- ***Notifiche***

Un aspetto fondamentale di *Unimate* sono le notifiche.
Questa tabella tiene traccia di alcune azioni che svolgono gli utenti: in particolare, quando un utente compie un'azione interfacciandosi con un altro (amicizia, partecipazione ad un evento, voto evento), si crea una *tupla* con le relative informazioni. I campi di cui è composta la tabella sono: *id*, chiave primaria; *user*, utente che riceve la notifica (quindi oggetto dell'azione); *categoria*, per specificare il tipo di notifica; *id_evento* e *id_amicizia* (mutuamente esclusivi, con possibilità di valori a *NULL*), utilizzati per recuperare le informazioni da altre tabelle; *visualizzato* e *ricevuto*, per gestire la notifica (tratteremo in seguito l'argomento).

id	user	categoria	visualizzato	id_evento	id_amicizia	ricevuto
417	P1234567	voto_event	si	39	*NULL*	si
416	P1234567	part_event	si	39	*NULL*	si
415	p1234567	amicizia	si	*NULL*	300	si
414	12345678912	voto_event	si	36	*NULL*	si
413	p1234567	part_event	si	38	*NULL*	si
412	12345678912	amicizia	si	*NULL*	299	si
411	s3689831	voto_event	si	37	*NULL*	si

- ***Log***

Log è una tabella contenente tutte le azioni compiute dagli utenti.
La tabella è costruita analogamente a *Notifiche*: *id*, chiave primaria; *user*, utente soggetto dell'azione; *tipo*, tipologia dell'azione; *id_amicizia*, *id_evento* e *id_messaggio* (mutuamente esclusivi, con possibilità di valori a *NULL*), per recuperare le informazioni delle attività dalle relative tabelle; *stamp*, per tenere traccia del momento in cui l'azione è stata effettuata.

id	user	tipo	id_amicizia	id_evento	id_messaggio	stamp
209	s3689831	send_msg	*NULL*	*NULL*	55	2016-12-21 17:16:50
218	s3689831	send_msg	*NULL*	*NULL*	57	2016-12-22 10:26:24
217	S3675883	send_msg	*NULL*	*NULL*	56	2016-12-22 10:26:10
216	s3689831	voto_event	*NULL*	39	*NULL*	2016-12-22 10:22:42
215	s3689831	part_event	*NULL*	39	*NULL*	2016-12-22 10:22:12
214	P1234567	create_event	*NULL*	39	*NULL*	2016-12-22 10:20:45
213	S3675883	amicizia	300	*NULL*	*NULL*	2016-12-22 10:15:53

LEGAMI TRA TABELLE

Le tabelle di *Unimate* sono legate fra loro con lo scopo di evitare la ridondanza dei dati, in questo modo per recuperare alcune informazioni è necessario richiamarne altre opportune.

Di seguito saranno trattati i vari legami tra tabelle.

- ***Utenti → DB Datamanage, Amici, Eventi, Messaggi, Notifiche, Log***

La tabella *Utenti*, come già accennato, contiene i dati degli utenti che utilizzano l'applicazione: è facile capire che questa tabella è fondamentale e viene richiamata spesso durante la ricerca di alcune informazioni.

Per primo consideriamo il rapporto tra *Utenti* e il database di *Datamanage.*
Le informazioni immagazzinate in *Utenti* vengono recuperate dalle tabelle *Studente*, *Docente* e *Azienda*, a seconda della categoria dell'utente. Gli utenti di *Datamanage* non sono contenuti a priori nel database di *Unimate*, ma sarà necessario accedere almeno una volta all'applicazione: al primo accesso, i dati corrispondenti all'username del login vengono cercati nel database di *Datamanage* e, nel caso in cui la ricerca abbia esito positivo e l'utente non sia già stato inserito (cioè è già stato effettuato un accesso), vanno a popolare i campi di *Utenti*.

Per quanto riguarda le tabelle di *Unimate*, in ognuna di esse è presente almeno un campo contenente *matricola/piva* per recuperare le informazioni dalla tabella *Utenti*.

- ***Notifiche → Amici, Eventi, Log***

Quando un utente compie un'azione, l'utente oggetto riceverà una notifica: è evidente che la tabella *Notifiche* abbia la necessità di poter recuperare informazioni riguardanti le attività. Ci sono tre tipologie di notifica (dovute alle azioni), specificate nel campo *categoria* della tabella in questione, che sono: *amicizia* (quando un utente ne segue un altro), *part_event* (relativa alla partecipazione ad un evento) e *voto_event* (quando un utente vota un evento).
Per sapere a quale amicizia o evento si fa riferimento, nella tabella *Notifiche* sono presenti i campi di identificazione *id_amicizia* e *id_evento* contenenti rispettivamente gli *id* delle tabelle *Amici* e *Eventi*.
Eventualmente da *Amici* si può recuperare l'utente soggetto dell'azione sfruttando il legame con la tabella *Utenti*, mentre per *Eventi* il processo di ricerca dell'utente soggetto è diverso: infatti in *Eventi* è presente solo il campo *proprietario*, che riceverà la notifica, quindi è necessario cercare l'azione compiuta dall'utente in questione nella tabella *Log* (che, come detto, tiene traccia di tutte le attività effettuate da ogni utente).

- ***Log → Amici, Eventi, Messaggi***

Quando un utente compie un'azione, questa viene inserita nella tabella *Log*, quindi, in maniera analoga a *Notifiche,* è necessario poter recuperare le informazioni dalle tabelle a cui è legata.
Le tipologie di log sono: *amicizia*, *part_event* e *voto_event* (come in *Notifiche*), *create_event* (quando un utente crea un evento) e *send_msg* (quando un utente invia un messaggio). Per sapere a quale attività specifica si fa riferimento, anche in *Log* sono presenti i campi di identificazione *id_amicizia*, *id_evento* e *id_messaggio* contenenti rispettivamente gli *id* delle tabelle *Amici*, *Eventi* e *Messaggi*.
Da quest'ultime si può recuperare l'utente oggetto dell'azione sfruttando il solito legame con la tabella *Utenti*: da *Amici* si usa *user2*, da *Eventi* si utilizza *proprietario* e da *Messaggi* si sfrutta *destinatario*.

CHIAMATE AL DATABASE DALL'APP

Per la comunicazione dell'applicazione con il database è stato necessario utilizzare un linguaggio di programmazione lato server: come accennato abbiamo scelto PHP.
È evidente la necessità dell'uso del linguaggio PHP (lato server) che poteva essere utilizzato come unico linguaggio di scripting, evitando l'utilizzo di Javascript (lato client), tuttavia il framework *Cordova* non lo supporta, inoltre è uso comune utilizzare Javascript per lo scripting delle pagine a cui l'utente si interfaccia.
Nel momento in cui è necessaria una chiamata al database, il processo di comunicazione con esso consiste in due fasi: da Javascritp a PHP e da PHP al database.

Nella prima fase, all'interno di un file Javascript viene effettuata una chiamata asincrona *Ajax*, in jQuery, a un file PHP.

```
$.ajax({

        type: "POST",

        url: "http://datamanage.altervista.org/check_login.php",

        data: "username_login=" + username_login + "&password_login=" + password_login,

          success: function(msg)
          {
              if(msg == "0")
                  alert("Login errato");
              else
                  location.href="home.html";
          },

        error: function()
        {
            alert("Si è verificato un errore!");
        }
});
```

La funzione *Ajax* ammette vari parametri. Noi all'interno di *Unimate* abbiamo utilizzato i più comuni ed essenziali: *type* indica il metodo in cui vengono passati i dati, può assumere valore GET (i dati vengono passati all'interno dell'indirizzo) oppure POST (i dati vengono passati tramite opportuni form); *url* denota il percorso del file a cui eseguire la chiamata (in questo caso un file PHP esterno all'applicazione); *data* contiene la coppia nome-valore da inviare, in caso di più valori è necessaria la loro concatenazione (mediante "&"). Nel caso in cui la chiamata vada a buon fine, viene considerato *success*, in caso contrario *error*: entrambi i parametri contengono una funzione a cui viene eventualmente passato il messaggio di risposta dal file PHP (in questo caso *msg* che contiene il valore di *echo* del relativo file PHP).
È buon uso utilizzare il parametro *dataType*, che specifica il tipo di dato ricevuto in risposta, anche se jQuery cerca di capirlo autonomamente.

```
dataType: "html",
```

Mentre il parametro *data* può essere omesso nel caso in cui non ci sia la necessità di inviare dei dati al file PHP, per esempio nel caso di una chiamata per recuperare alcuni valori dal database e non per inserirli.

Nella seconda fase, il file PHP (a cui è stata effettuata la chiamata *Ajax* precedente) riceve gli eventuali valori ed esegue delle richieste direttamente al database.
In primis è necessaria la connessione al database.

```
<?php
$host = "localhost";
$user = "datamanage";
$password = "";
$db = "my_datamanage";

$connessione = new mysqli($host, $user, $password, $db);

if ($connessione->connect_errno) {
    echo "Connessione fallita: ". $connessione->connect_error . ".";
    exit();
}
?>
```

Il file di connessione è incluso in tutti gli altri file PHP.
Nell'immagine successiva è possibile osservare come in PHP vengano recuperati gli eventuali dati ricevuti dal relativo file Javascript, immagazzinandoli in variabili. Come detto, tutti i file PHP hanno il compito di comunicare con il database, quindi al loro interno vengono costruite le opportune query: si crea una variabile e le si assegna la stringa (cioè la query). Successivamente si effettua la chiamata (tramite la funzione *mysqli_query();*) e si tiene in memoria il risultato, restituito come tabella.

```php
<?php
session_start();
include "connect_db.php";

$username_login = $_POST['username_login'];
$password_login = $_POST['password_login'];

$control = false;
$first = substr($username_login, 0, 1);

if($first == "s" or $first == "S"){

$sql = "SELECT * FROM Studente WHERE matricola = '$username_login' AND password = '$password_login'";
$con = mysqli_connect($host, $user, $password, $db);
$result = mysqli_query($con, $sql);
$row = mysqli_fetch_array($result);
```

Tale risultato viene suddiviso in righe, per l'esattezza in array (tramite la funzione *mysqli_fetch_array()*), per poi scorrerle ad ogni iterazione di un ciclo (generalmente un ciclo *while*), nel caso in cui il risultato sia maggiore di uno. Nel nostro esempio, trattandosi di un login, la *SELECT* produrrà al massimo un solo risultato, quindi non è stato necessario un ciclo, bensì un costrutto *if*, per controllare che esista effettivamente un risultato (*$row > 0*), e il corrispondente *else*, in cui è presente l'*echo* a cui si accennava in precedenza (tale testo viene recuperato nella funzione *Ajax* che ha effettuato la chiamata). Per concludere si chiude la connessione al database.

```php
if($row > 0){
else{
echo "0";
}
```

IV. EVENTI

Un aspetto fondamentale di *Unimate* risiede nell'opportunità di potersi interfacciare con eventi. Come abbiamo già visto, nel database esiste una tabella relativa agli eventi, nella quale sono conservate tutte le opportune informazioni. Un utente, quindi, con l'aiuto dell'applicazione, può decidere di creare eventi, ricercarli, confermare la propria partecipazione ed esprimere un giudizio. L'applicazione propone diverse tipologie di evento riguardanti vari aspetti della vita universitaria:

- *Lezione*
- *Ricevimento professore*
- *Servizio Università*
- *Seminario*
- *Laboratorio*
- *Ritrovo/Festa*
- *Offerta di Lavoro*

Ogni evento è caratterizzato da alcuni parametri, quali nome dell'evento, indirizzo, data, orario, descrizione e proprietario.

CREAZIONE DELL'EVENTO

Ogni utente ha la possibilità di creare eventi, tuttavia, a seconda della categoria dell'utente, è possibile scegliere di creare solo alcune tipologie (tramite opportuno controllo). In particolare:

Studente	Docente	Azienda
Ritrovo/Festa	*Lezione*	*Offerta di Lavoro*
	Ricevimento	
	Servizio Università	
	Seminario	
	Laboratorio	

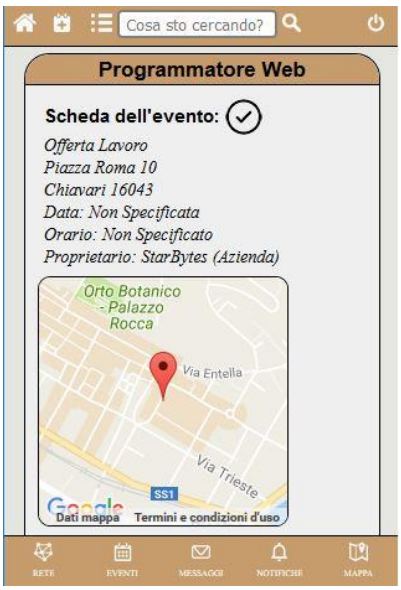
In questo modo, si evita che uno studente possa creare eventi che non gli "appartengono", ad esempio un servizio universitario; analogo funzionamento per quanto riguarda docenti e aziende.

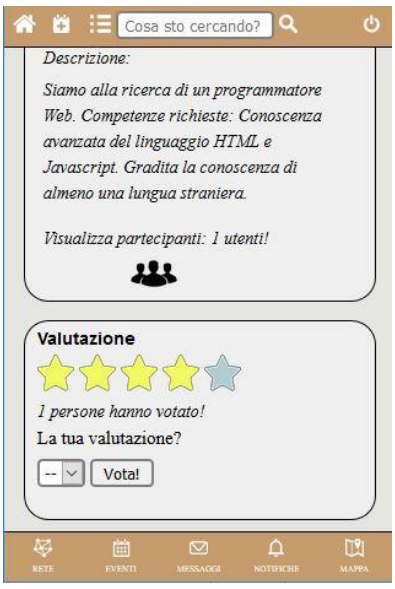
Cliccando sul pulsante *Add Event* nell'*Header*, si accede alla pagina di creazione dell'evento nella quale si presenta un classico form per l'inserimento delle relative informazioni. Per quanto riguarda gli eventi ricorrenti (ad esempio le *lezioni*), abbiamo scelto una particolare politica: tutti i campi sono obbligatori, eccetto *Data* e *Orario*. In questo modo si può evitare di specificare il giorno e l'ora stabilite per l'evento, ma è consigliato (buona norma) precisare le ricorrenze nella descrizione (come i giorni di lezione). Una volta riempiti i campi necessari, l'utente può aggiungere l'evento cliccando sul pulsante *Aggiungi Evento*, in fondo alla pagina.

VISUALIZZAZIONE DEGLI EVENTI

Una volta creato un evento, questo viene immagazzinato nel database, quindi tutte le relative informazioni sono a disposizione degli utenti.
Per raggiungere la pagina di un evento, *Unimate* propone due alternative: la ricerca (trattata in seguito) e un menu dedicato.
Cliccando sul pulsante *Eventi* del *Footer* (in qualsiasi pagina), comparirà un piccolo menu con alcune voci: *I tuoi eventi* (per visualizzare gli eventi di cui l'utente è proprietario), *Eventi a cui partecipi* e *Guarda eventi* (per visualizzare tutti gli eventi). Tratteremo più avanti (nella sezione dedicata alla Ricerca) i menù-lista dei risultati, con l'elenco degli eventi.

Esaminiamo ora la pagina dell'evento.
In questa l'utente può trovare tutte le informazioni pertinenti (opportunamente recuperate dal database) ed eventualmente può decidere di partecipare con un semplice click: così facendo crescerà il numero dei partecipanti, i quali sono visualizzabili premendo sul relativo pulsante tramite una *dialog*. Si noti che attraverso l'immagine dell'utente partecipante, è possibile visitarne il profilo.





Vedremo successivamente come sono state costruite le *dialog* usando jQuery.

Particolare la modalità con cui *Unimate* calcola la posizione dell'evento: al momento della creazione, l'utente inserisce l'indirizzo composto da *Indirizzo*, *Civ.*, *Città* e *CAP*, informazioni che vengono concatenate e utilizzate come "chiave" per ottenere le relative coordinate. Infatti per inserire il marker nella mappa sono necessarie latitudine e longitudine.

```
242  var geocoder = new google.maps.Geocoder();
243  geocoder.geocode( { address: stringa_mappa }, function(results, status) {
244  if (status == google.maps.GeocoderStatus.OK) {
245      var latitudine_evento = results[0].geometry.location.lat();
246      var longitudine_evento = results[0].geometry.location.lng();
247      get_valori(latitudine_evento, longitudine_evento);
248  }
249  else {
250      alert("Indirizzo non trovato!");
251  }
252  });
```

Mediante l'oggetto *Geocoder* è possibile richiamare la funzione *geocode* alla quale viene passato come parametro *address* contenete la suddetta "chiave", cioè la stringa composta dall'indirizzo concatenato completo. Le coordinate vengono calcolate (a partire dall' indirizzo "chiave") autonomamente da *geocode* e, tramite il suo secondo parametro (una funzione), si esamina il risultato: al suo interno consideriamo i casi in cui l'esito è positivo (coordinate calcolate), quindi passiamo latitudine e longitudine all'opportuna funzione che li inserirà nel database insieme alle altre informazioni (richiamando un file PHP come precedentemente spiegato); oppure negativo (coordinate non trovate e non calcolate), in questo caso viene mostrato un messaggio di errore.

PARTECIPAZIONE E FEEDBACK

Abbiamo già accennato alla possibilità di poter partecipare agli eventi.
L'utente può decidere di partecipare cliccando semplicemente sull'icona disponibile e, nel caso in cui fosse già un partecipante, comparirà l'icona per annullarla. Se eventualmente l'utente è proprietario di tale evento, non potrà esprimere la propria partecipazione.
Dal punto di vista del database, il click all'icona *partecipa* crea una tupla di categoria *part_event* all'interno delle tabelle *Notifiche* e *Log*, inoltre il proprietario riceverà una notifica di nuova partecipazione (trattate successivamente).
Come precedentemente illustrato, è possibile visualizzare una *dialog* (tramite l'opportuna icona) con l'elenco dei partecipanti: questo elenco viene creato recuperando dalla tabella *Log* gli utenti che hanno *part_event* per l'evento in questione.

Il progetto richiedeva un sistema di feedback per gli eventi: abbiamo pensato che il modo più semplice per affrontare questo tema fosse esprimere una valutazione, quindi *Unimate* permette agli utenti di dare un voto agli eventi.
Nella tabella *Eventi* del database sono presenti i campi *num_votanti* e *media_voto*: questi due valori si aggiornano ogni volta che un utente esprime il proprio giudizio e, dopo aver opportunamente calcolato la media dei voti (per l'esattezza viene calcolato il range), il risultato viene mostrato mediante "stelle" (classico effetto grafico relativo alle votazioni).
Come per le partecipazioni, il proprietario non può votare un proprio evento, inoltre in caso un utente abbia già votato, non potrà farlo nuovamente.

V. RICERCA E SESSIONI

Come ogni applicazione social che si rispetti, anche *Unimate* rende possibile la ricerca.
La nostra app supporta due tipi di ricerca: *Ricerca Avanzata* e *Ricerca Generica*, dedite a trovare utenti e/o eventi.

RICERCA AVANZATA

Per *Ricerca Avanzata* si intende una ricerca suddivisa in parametri, cioè all'utente viene fornito un form nel quale possono essere inseriti i dati da cercare, in questo modo nel database vengono effettuate delle richieste specifiche ad opportune tabelle.
Cliccando sull'icona *Ricerca Avanzata* dell'*Header,* compare una *dialog* contenente il suddetto form. L'utente ha la possibilità di cercare altri utenti o eventi.

Nel caso in cui la scelta ricada sull'utente, è possibile specificare la categoria (ovvero *Studente*, *Docente*, *Azienda* e *Tutti*) e, in base a questa, i campi di ricerca cambiano:

Studente	Docente	Azienda
Nome	*Nome*	*Nome*
Cognome	*Cognome*	*P.IVA*
Matricola	*Matricola*	
Università	*Università*	
Facoltà		
Corso di studi		

Se l'utente decidesse di cercare nella tipologia *Tutti*, la ricerca comprenderebbe tutte le categorie.

È possibile ricercare gli eventi in maniera analoga alla ricerca utente, scegliendo la categoria (le opzioni sono costituite da tutte le tipologie di evento, già descritte, e *Tutti*): in questo caso sarà possibile riempire i campi *Nome* e *Città/CAP* (per ottenere una ricerca più precisa abbiamo deciso di collegare questi due dati).

RICERCA GENERICA

Attraverso la *Ricerca Generica*, è possibile effettuare una ricerca in tutte le tabelle del database: l'utente digita l'input (può essere anche incompleto) nella barra dell'*Header* e cliccando sul pulsante *Search* si potrà visualizzare il risultato.
A differenza della *Ricerca Avanzata*, nella quale è necessario l'inserimento completo dei campi da cercare ed è obbligata la suddivisione tra Utente/Evento e tra categorie, nella *Ricerca Generica* l'utente ha la libertà di scegliere qualunque tipo di dato esso stia cercando e ha la possibilità di immettere nella barra informazioni incomplete: sarà *Unimate* a cercare l'input della ricerca in tutte le sue tabelle.

QUERY DINAMICHE

In entrambe le ricerche è stato sviluppato un meccanismo di query dinamiche, ovvero le richieste al database (le query vere e proprie) vengono costruite a seconda dell'input di ricerca.

Nella *Ricerca Avanzata*, l'utente può decidere di cercare solo alcuni dati, quindi può inserire uno o più campi del form: l'applicazione deciderà cosa ricercare nelle opportune tabelle.
Per sviluppare le query dinamiche è stato necessario considerare la struttura delle query:

"SELECT campi_da_selezionare FROM nome_tabella WHERE condizioni"
(1) (2) (3)

1) In primo luogo bisogna decidere l'azione (nelle ricerche è sempre una SELECT) e il campo da ricercare (solitamente tutti *), quindi questa parte della query non è dinamica.
2) A seconda della categoria selezionata (Utente o Evento), *Unimate* effettua la ricerca nella relativa tabella. In realtà i pulsanti *Cerca* della *dialog* di ricerca sono distinti e conducono a due file Javascript differenti: uno per gli eventi (in cui sarà "FROM Eventi") e uno per utenti ("FROM Utenti").

3) Le condizioni costituiscono la parte dinamica vera e propria. Uno script analizza l'input dell'utente: ogni *textbox* è associata ad un controllo e nel caso in cui il campo non sia vuoto viene creata una stringa. Vengono presi i campi effettivamente inseriti e, costruendo la stringa, vengono messi in AND tra loro in modo da creare la condizione della query.

```
53  if(select_check == "tutti"){
54  var query = "SELECT * FROM Utenti " + " WHERE " + record + "";
55  }
56  else{
57  var query = "SELECT * FROM Utenti " + " WHERE " + record + " AND categoria=" + "'" + select_check + "'";
58  }
```

È evidente che concatenando (1), (2) e (3), si ottiene la query completa da inviare al file PHP.

La *Ricerca Generica* segue un procedimento analogo, con qualche differenza: la ricerca avviene in entrambe le tabelle *Utenti* ed *Eventi* e la condizione dinamica viene costruita utilizzando *LIKE* e OR.
LIKE permette di ricercare all'interno della stringa, ovvero non è necessario che l'input sia identico al record (se si digita, per esempio, "ric", saranno trovati tutti gli utenti che contengono tale sottostringa, come "riccardo").
Per rendere la ricerca più generica possibile, ogni parola dell'input viene considerata come stringa a sé stante e ricercata in tutti i campi delle tabelle: tale procedimento è possibile mediante l'uso di OR.

```
47  for(var i = 0; i < stringa.length; i++){
48      condizione = condizione + "(nome LIKE '%" + stringa[i] + "%' OR cognome LIKE '%" + stringa[i] + "%' OR matricola LIKE '%" + stringa[i] + "%' OR piva LIKE
        '%" + stringa[i] + "%') AND ";
49  }
```

Il tutto viene messo in AND, come nel caso della *Ricerca Avanzata*, per creare la condizione completa della query.

SESSIONI

Come accennato più volte, ogni ricerca conduce a una pagina contenente i risultati in un elenco (a cui ci siamo già riferiti come menu-lista), anche per quanto riguarda il menu degli eventi: deve esserci, quindi, uno scambio di informazioni tra le pagine. Questo procedimento è realizzabile tramite l'utilizzo di *Sessioni*, ovvero un meccanismo di memorizzazione di informazioni che opera sul server (in linguaggio PHP).

Unimate fa uso di quattro diverse *Sessioni*:
1) Il primo incontro avviene già al login, infatti viene creata una sessione contenente l'username: in questo modo si può navigare in *Unimate*, senza doversi ogni volta autenticare. Le informazioni di questa sessione (ovvero matricola/piva dell'utente) vengono utilizzate più volte all'interno dell'app.
2) Una "friend-session" per immagazzinare i dati di un utente, ovvero matricola/piva della persona di cui si vuole visualizzare il profilo.
3) Una sessione per gli eventi, in cui viene memorizzato l'id dell'evento
4) Infine una sessione per le ricerche. Questa viene utilizzata per mostrare i menu-lista (qui di fianco) dovuti a ricerche, ma anche per *Rete* e per le voci del menu *Eventi* (essendo anche queste implicitamente ricerche nel database).

Per descrivere il funzionamento delle sessioni per il passaggio di dati tra pagine di *Unimate*, analizzeremo il caso delle ricerche.

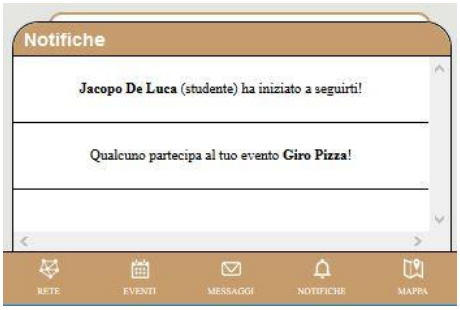
Come già descritto, i dati immessi nel form di ricerca, dopo aver completato la query al database, devono produrre un risultato espresso in un'altra pagina: il passaggio di questi dati (un elenco di utenti/eventi con le relative caratteristiche) è ovviamente possibile grazie all'utilizzo della sessione specifica. I dati di cui sopra vengono immagazzinati in un oggetto di tipo *JSON*, dato

inviato al file PHP della sessione (sovrascrivendo quella precedente). A questo punto la pagina-risultato recupera i dati dalla sessione appena sovrascritta e li visualizza come menu-lista.

```php
<?php
session_start();
include "connect_db.php";

$query = $_POST['ricerca'];
$nome = $_SESSION['nome'];

$sql = "$query";
$con = mysqli_connect($host, $user, $password, $db);
$result = mysqli_query($con, $sql);
$response = array();

while($row = mysqli_fetch_array($result)){
    if(!(($nome == $row[2]) || ($nome == $row[13]))){
    array_push($response, array("nome"=>$row[0], "cognome"=>$row[1],
    "matricola"=>$row[2], "password"=>$row[3], "universita"=>$row[4],
    "facolta"=>$row[5], "corso"=>$row[6], "indirizzo"=>$row[7],
    "civ"=>$row[8], "citta"=>$row[9], "cap"=>$row[10], "mail"=>$row[11],
    "phone"=>$row[12], "piva"=>$row[13], "sitoweb"=>$row[14],
    "categoria"=>$row[15], "foto"=>$row[16]));
    }
}

echo json_encode(array("server_response"=> $response));
mysqli_close($con);
?>
```

VI. NOTIFICHE

Unimate prevede un sistema di notifiche in tempo reale. Ogni qualvolta che un utente interagisce con alcune funzionalità dell'app, i profili interessati ricevono una notifica. In particolare l'utente può decidere di seguirne un altro, di partecipare ad eventi e votarli: queste attività vengono poi segnalate al profilo interessato (quello seguito o quello proprietario dell'evento).



FUNZIONAMENTO

Quando viene compiuta una delle suddette azioni, a livello di database vengono aggiornate le tabelle *Amici*, *Eventi* e *Log* mentre vengono inserite le informazioni nella tabella *Notifiche*: per quanto riguarda gli eventi, le informazioni della notifica vengono recuperate dalla tabella *Log* per risalire all'utente soggetto (in quanto in *Eventi* esiste solo il campo *proprietario*); analogamente, le notifiche di amicizia recuperano le informazioni dalla tabella corrispondente (*Amici*).
L'utente destinatario della notifica controlla periodicamente nella tabella *Notifiche*, verificando se è stata inserita una nuova riga contenente nel campo *user* la propria matricola/piva. In

particolare la funzione di controllo *check()* viene passata come parametro al metodo *setInterval()* insieme all'intervallo in millisecondi: scegliendone uno molto piccolo (1s nel nostro caso), si può parlare di notifica in tempo reale.

```
25  <body id="body" onload="setInterval('check()', 1000)">
```

Per essere più precisi, *check()* non controlla solo le notifiche, ma tutte le funzioni richiedenti un costante monitoraggio, come il controllo sulla ricezione di nuovi messaggi (trattata in seguito).

I CAMPI 'RICEVUTO' E 'VISUALIZZATO'

Se durante tali controlli viene trovata una nuova notifica, nel *Footer*, la corrispondente icona cambia, mostrando la classica spunta rossa: cliccandoci apparirà il box-notifiche (immagine a inizio capitolo), contenente quelle appena ricevute.
Per gestire il nostro sistema di creazione notifiche, nella relativa tabella ci siamo serviti dei campi *ricevuto* e *visualizzato*. Vediamo il funzionamento:

	ricevuto	*visualizzato*
1)	*NULL*	*NULL*
2)	*si*	*NULL*
3)	*si*	*si*

inizialmente, alla creazione della riga nella tabella *Notifiche*, tali campi sono settati a *NULL*, come descritto nel caso (1); quando la notifica viene ricevuta, dopo ogni controllo della corrispondente funzione in *setInterval()*, il campo *ricevuto* viene aggiornato, come nel caso (2); infine il caso (3) rappresenta i valori dei campi in questione dopo aver aperto il box-notifiche.
L'utilizzo di due campi distinti, permette di risolvere due problemi riscontrati durante lo sviluppo di questo sistema: ad ogni periodo di *setInterval()* vengono prima caricate le notifiche del caso (1), cioè quelle nuove appena create nel database, e solo successivamente *ricevuto* è settato a *si*, questo permette di non caricare ogni volta le stesse notifiche in loop; ogni volta in cui viene caricata una pagina, vengono caricate per prime le notifiche con i campi *ricevuto* e *visualizzato* rispettivamente a *si* e *NULL* (solo dopo aver aperto il box-notifiche si arriverà al caso (3)), in questo modo al primo accesso e al passaggio in un'altra pagina si evita di perdere notifiche ancora non visualizzate.

VII. MESSAGGI

Essendo un social, *Unimate* fornisce la possibilità di poter inviare e ricevere messaggi, gestendoli da database (come già visto). Tale sistema è stato concepito sul modello "mail": ogni utente può inviare un messaggio ed eventualmente ricevere risposta all'interno di un altro messaggio, pertanto non si parlerà di conversazioni (tipo chat), bensì di un semplice scambio di testi, archiviati nel database e visualizzabili nella apposita sezione dell'app.

FUNZIONAMENTO

L'invio di un messaggio è possibile in due modi differenti: quando si visita il profilo di un utente (non per forza bisogna "seguirlo"), cliccando sull'icona opportuna, oppure dalla sezione *Messaggi* raggiungibile dal *Footer*.
Al click si apre una *dialog* con la casella di testo dedicata e il pulsante di invio. Il destinatario viene identificato dalla sessione *friend*, ciò implica il fatto che, quando viene premuto il bottone *Invia messaggio*, prima questa vada sovrascritta con la

matricola/piva corretta, dopodiché viene creata nella tabella *Messaggi* del database una nuova tupla contenente le relative informazioni.

NOTIFICA DI MESSAGGIO RICEVUTO

Similmente alle notifiche, quando si riceve un nuovo messaggio, *Unimate* ci avvisa prontamente in tempo reale.
Quando nel database ne viene aggiunto uno nuovo (nella tabella *Messaggi*), il corrispondente campo *visualizzato* viene settato a *NULL* di default. La stessa funzione *check()* considerata per le notifiche, passata come parametro a *setInterval()*, contiene all'interno il controllo di nuovi messaggi: ad ogni periodo quindi si verificano eventuali nuovi messaggi ricevuti e nel caso in cui l'esito della ricerca sia positivo, sull'icona *Messaggi* del *Footer* comparirà la solita spunta rossa. Cliccandoci, l'utente verrà indirizzato nella sezione dedicata dalla quale è possibile scorrere tutti quelli inviati e ricevuti, con le relative informazioni.

VIII. MAPPA

Nel capitolo relativo agli eventi è stato già accennato l'utilizzo delle mappe di Google. *Unimate*, infatti, dispone di una sezione ad esse dedicata, nella quale l'utente può riconoscere la propria posizione e quella degli eventi circostanti.

INTEGRAZIONE DELLE GOOGLE MAPS E FUNZIONAMENTO

Per poter utilizzare le mappe, è stato necessario ottenere la relativa API key: è stato necessario registrare un account Google e fare una richiesta ai loro server mediante apposito form.

```
369 <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDoXzqjiYpWXx8y0yfspSXqDD5tRqJTiaU&callback=initMap" async defer></script>
```

Particolarmente importante la funzione *initMap()*, la quale viene richiamata ogni volta che la mappa è caricata: al suo interno sono presenti tutti gli oggetti e i metodi fondamentali per l'utilizzo e la visualizzazione (inizializzazione, marker, InfoWindow, ecc).
L'inizializzazione della mappa è per merito di *Map()*, il quale riceve i vari parametri da impostare.

```
251 var map = new google.maps.Map(document.getElementById('map'), {
252   disableDefaultUI: true,
253   zoom: 14
254 });
```

Sulla mappa che si sta analizzando sono visibili graficamente i marker relativi alla nostra posizione e quella degli eventi nei dintorni.
Per calcolare la nostra posizione, si sfrutta la funzione *getCurrentPosition()* che restituisce, in caso di successo, le coordinate.

```
307 var watchID = navigator.geolocation.getCurrentPosition(onSuccess, onError, options);
```

Quest'ultime vengono usante dagli oggetti *Marker*, i quali vengono creati e "disegnati" sulla mappa. Nel caso della nostra posizione è facile notare che le coordinate vengono recuperate

come sopra descritto (tramite *getCurrentPosition()*), mentre per quanto riguarda gli eventi, è necessario richiamarle dal database.

```
272         if(partecipa == "si"){
273             marker[i] = new google.maps.Marker({
274             position: posizione_marker,
275             icon: image_si,
276             map: map
277             });
278         }
279         else if(partecipa == "no"){
280             marker[i] = new google.maps.Marker({
281             position: posizione_marker,
282             icon: image_no,
283             map: map
284             });
285         }
```

VISUALIZZAZIONE DELLA MAPPA

Come ogni funzionalità di *Unimate*, la mappa è raggiungibile tramite l'opportuna icona del *Footer*: cliccandoci, vengono mostrati i marker della nostra posizione e quelli degli eventi nelle vicinanze. Ovviamente per renderla disponibile, è necessario autorizzare la localizzazione sul proprio dispositivo, in caso contrario la mappa non verrà visualizzata.
L'utente ha la possibilità di scorrere la Google map e di zoomare, rendendo visibili anche altri eventi, poiché tutti sono mappati.
Abbiamo scelto di assegnare tre differenti colori ai marker per identificarli:

1) Blu: la nostra posizione
2) Verde: eventi di nostra proprietà e a cui si partecipa
3) Rosso: altri eventi (nessun interesse specificato)

Ad ogni marker è stato associato un oggetto *InfoWindow*, che si apre al click su di esso, mostrante alcune informazioni dell'evento e un link per raggiungerne la pagina.

IX. FOLLOWING

Come già accennato, in *Unimate* non si creano amicizie (stile *Facebook*), bensì l'utente sceglie di seguirne un altro in maniera univoca (stile *Instagram*), in tal caso il secondo verrà aggiunto alla rete del follower.
Visitando il profilo di un utente, in cui si può scorrere tra i vari post, è facile notare l'icona relativa all'amicizia: sarà possibile aggiungerlo alla propria rete oppure, nel caso fosse già seguito, comparirà l'icona per rimuoverlo (come nel caso dell'immagine affianco). Da *Rete*, sempre nel *Footer*, è possibile visualizzare la propria rete di contatti all'interno del solito menu-lista, già incontrato per altre funzionalità dell'app. Dal punto di vista del database un nuovo following implica la creazione di una nuova riga nella tabella *Amici* (oltre che in *Notifiche* e *Log*), contenente le matricole/piva degli utenti in questione; eliminare un contatto dalla propria rete comporta la cancellazione della riga corrispondente.

X. JQUERY

jQuery è una libreria di Javascript per applicazioni web, con l'obiettivo di semplificare il linguaggio e implementare le chiamate *AJAX*.
Per il suo utilizzo è necessario includere il file js nell'intestazione dei documenti HTML.

```
<script src="https://code.jquery.com/jquery-1.12.4.js"></script>
```

Noi abbiamo scelto di recuperarlo direttamente dai server di jQuery, piuttosto che scaricare il documento nel nostro (ovviamente è necessario sempre includerlo).
In *Unimate*, jQuery è stato principalmente per le chiamate *AJAX* (già analizzate nel capitolo relativo al database) e per semplificare il codice Javascript.

Inoltre, è stata utilizzata la libreria di plug-in, ovvero jQuery UI.

```
13  <link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
14  <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
```

Anche in questo caso è necessario includere i file js e css (per l'interfaccia grafica) nell'intestazione. Tale libreria fornisce interazioni e animazioni, effetti avanzati e widget: il suo utilizzo in *Unimate* si limita alle *dialog* incontrate per messaggi e ricerca.

```
1  $( function() {
2      $( ".dialog" ).dialog({
3          autoOpen: false,
4          title: "Ricerca Avanzata",
5          draggable: false,
6          resizable: false,
7          modal: true
8      });
9  } );
```

Dall'immagine si capisce che jQuery metta a disposizione, come detto, dei widget, quali la dialog, e rende possibile settare tutte le opportune caratteristiche in maniera semplice ed efficiente, rendendo una completa personalizzazione degli oggetti utilizzati.

XI. APACHE CORDOVA

Apache Cordova (ex *PhoneGap*) è un framework per lo sviluppo di applicativi per dispositivi mobili: questo permette agli sviluppatori di creare applicazioni mobili usando CSS3, HTML5 e Javascript anziché affidarsi alle API specifiche delle piattaforme Android, iOS e Windows Phone.
Il framework incapsula il codice "traducendolo" in ibrido (tra nativo e web), permettendo ai più comuni OS la sua interpretazione, rendendo quindi la app in questione multi-platform.

FUNZIONAMENTO

Innanzitutto è necessario installare Cordova sul proprio pc.

```
C:\Users\Riccardo>npm install -g cordova
```

Il secondo passo è creare il progetto dell'app, specificando quindi il percorso: in esso viene creata la cartella *www* (la quale a sua volta racchiude le cartelle CSS e JS) che deve contenere i file della nostra app, i quali poi saranno tradotti.

```
C:\Users\Riccardo>cordova create MyApp
```

Alla creazione del progetto, compare un file *index.js* di default contenente tutte le funzioni base necessarie per il corretto funzionamento dell'applicazione tradotta da Cordova.
Successivamente, entrati nella directory appena creata, è necessario scegliere le piattaforme a cui si è interessati: tramite l'apposito comando si aggiunge la piattaforma desiderata.

```
C:\Users\Riccardo>cordova platform add android
```

Infine per compilare, installare ed eseguire l'app sul dispositivo, è necessario lanciare il comando *run* di Cordova.

```
C:\Users\Riccardo>cordova run android
```

Cordova dispone di molti comandi (per esempio per la sola compilazione o installazione), facilmente reperibili nella documentazione ufficiale.
Ovviamente, per il corretto funzionamento dell'applicazione, sono necessari tutti gli strumenti, quindi SDK, relativi alla piattaforma che si vuole utilizzare.

```
BUILD SUCCESSFUL

Total time: 21.748 secs
Built the following apk(s):
        C:/Users/Riccardo/Documents/Cordova/hello/platforms/android/build/outputs/apk/android-debug.apk
ANDROID_HOME=C:\Users\Riccardo\AppData\Local\Android\sdk
JAVA_HOME=C:\Program Files\Java\jdk1.8.0_91
No target specified, deploying to device '015d4a8202600015'.
Skipping build...
Built the following apk(s):
        C:/Users/Riccardo/Documents/Cordova/hello/platforms/android/build/outputs/apk/android-debug.apk
Using apk: C:/Users/Riccardo/Documents/Cordova/hello/platforms/android/build/outputs/apk/android-debug.apk
Package name: com.example.hello
LAUNCH SUCCESS
```

PLUG-IN DI CORDOVA

Cordova mette a disposizione vari plug-in che devono essere installati nell'applicazione. Questi permettono di utilizzare varie funzionalità dei device: per creare l'app, ci si serve di una programmazione web, ciò significa che le tipiche funzionalità dei dispositivi non sono previste (ad esempio la fotocamera, il volume, la rubrica ecc); i plug-in di Cordova rendono possibile il loro utilizzo tramite Javascript.
L'installazione avviene mediante linea di comando, dopo essersi posizionati nella cartella del progetto. In *Unimate* sono stati utilizzati tre plug-in:

1) *camera*, necessario per scattare foto e selezionarle dalla galleria del dispositivo.
   ```
   C:\Users\Riccardo\Documents\Cordova\hello>cordova plugin add cordova-plugin-camera
   ```
2) *file-transfer*, necessario per il trasferimento di file al server.
   ```
   C:\Users\Riccardo\Documents\Cordova\hello>cordova plugin add cordova-plugin-file-transfer
   ```
3) *geolocation*, necessario per la localizzazione del device.
   ```
   C:\Users\Riccardo\Documents\Cordova\hello>cordova plugin add cordova-plugin-geolocation
   ```

La geolocalizzazione è stata utilizzata per la mappa, come già è stato analizzato in precedenza. Per quanto riguarda la fotocamera e il trasferimento file, essi sono stati utilizzati per l'unica funzionalità non ancora esposta: il cambio dell'immagine del profilo. Nei social è ormai fondamentale la possibilità di condividere, tra le proprie informazioni, anche la propria immagine, in modo tale da completare il biglietto da visita che incontreranno i nostri visitatori.

```
34  function cameraGetPicture() {
35      navigator.camera.getPicture(onSuccess, onFail, {
36          quality: 50,
37          targetWidth:500,
38          targetHeight:500,
39          destinationType: Camera.DestinationType.NATIVE_URI,
40          sourceType: Camera.PictureSourceType.PHOTOLIBRARY
41      });
42
```

Unimate fornisce tale possibilità: l'utente può decidere una propria foto da rendere disponibile nel proprio profilo (di default è l'immagine di *Unimate*). Questa funzionalità è stata possibile grazie ai plug-in di cui sopra. Di fianco, il codice utilizzato per la selezione della foto dalla galleria del device, in caso la funzione abbia successo, avviene l'upload nel server: l'immagine viene passata a un file PHP che si

```
1   <?php
2    if(isset($_FILES['FileUtente'])){
3     $tempPos = $_FILES['FileUtente']['tmp_name'];
4     $destPos = "./img/foto_profilo/".$_FILES['FileUtente']['name'];
5     move_uploaded_file($tempPos, $destPos);
6     echo "Foto profilo aggiornata con successo!";
7    }else{
8     echo "Si e' verificato un errore! Riprova.";
9    }
10  ?>
```

occupa di caricarla sul server. Successivamente viene aggiornato il campo *foto* della tabella *Utenti* del database (a *NULL* di default).

XII. CORREZIONI E POTENZIALITÁ

Già nella premessa è stato accennato che il lavoro per realizzare *Unimate* è sempre stato in crescendo: i primi periodi sono stati necessari per ottenere informazioni riguardo agli strumenti da utilizzare, linguaggi e altre nozioni fondamentali per il corretto funzionamento dell'app. Le nostre conoscenze, quindi, sono cresciute parallelamente allo sviluppo dell'applicazione, costringendoci ad affrontare ogni volta problemi diversi e mai incontrati nel nostro percorso di studi. Per quanto detto, solo a progetto completato ci siamo accorti di alcune imperfezioni che si sarebbero potute evitare.

1) La gestione della chiave primaria della tabella *Utenti* (due campi differenti matricola/piva) ha più volte appesantito codice e controlli: sarebbe stato molto più efficiente servirsi di un unico campo *user* contenente la matricola o la partita iva.
2) Lo scambio di informazioni tra file Javascript e PHP è stato gestito con formato di ritorno HTML (*dataType: html*) e ogni volta è stata necessaria una conversione al formato JSON per utilizzare i dati: sarebbe stato molto più semplice adoperare direttamente il formato JSON (*dataType: json*).
3) Invece di creare più sessioni relative a più funzionalità dell'app in più file, si sarebbe potuto creare un array di sessioni contenente tutte le informazioni necessarie in un unico file PHP.
4) Alcune pagine HTML e alcuni file CSS sono ridondanti: si sarebbe potuta usare la stessa pagina e lo stesso stile per più sezioni (ad esempio le pagine con i risultati delle varie ricerche), sfruttando la dinamicità del linguaggio Javascript.
5) La *Ricerca Generica* ha richiesto l'uso di due query differenti (una per gli utenti e una per gli eventi): si sarebbe potuta utilizzare una join tra le tabelle *Utenti* ed *Eventi* e ricercare nella tabella risultante.

Unimate è semplice ed immediata e ovviamente potrebbe essere soggetta a molti miglioramenti che la renderebbero più interessante, i quali non sono stati implementati per questioni di tempo. Di seguito le potenzialità di *Unimate* a cui abbiamo pensato:

1) Si potrebbe implementare una chat (molto più moderna) in sostituzione al sistema di messaggistica utilizzato.
2) *Unimate*, come *Datamanage*, è totalmente priva di sicurezza: con una semplice *query injection* si potrebbero compromettere tutte le informazioni del database, perciò sarebbe necessario un controllo accurato su tutti gli input (filtrare caratteri ecc).
3) Abbiamo gestito il lato server tramite pagine PHP, ma troppo tardi abbiamo scoperto la possibilità di usare Javascript (solitamente limitato al lato client) anche server-side mediante il framework *Node.js*, il quale è consigliato per gli sviluppatori che utilizzano Cordova, probabilmente perché questo non supporta il linguaggio PHP.

A cura di:
Rossi Riccardo
De Luca Jacopo