RELAZIONE PROGETTO

Corso SAW

Maroz Vera

Silvestri Fabrizio

Beltrame Stefano

Sommario

Idea generale 4
Problema 4
EventAround 4
Funzionalità Visitatore 5
Informazioni sul sito 5
Ricerca eventi 5
Registrazione al sito 6
Funzionalità Utente 7
Profilo 7
Ricerca plus 7
Eventi seguiti 8
Visualizzare profilo utenti 8
Area messaggi 9
Aggiungere eventi 9
Gestione dei propri eventi 10
Funzionalità amministratore 11
Accesso 11
Dashboard 11
Funzionalità utenti 11
Funzionalità eventi 12
Funzionalità amministratore 12
Struttura database 13
Admin 13
Events 13
Users 13
Profiles 13
Followed 13
Signaled 13
Messages 13
Specifiche tecniche 14
Sicurezza 14
Gestione accesso 14

Validazione input 14
Database 15
Interazioni 15
Gestione concorrenza 16
Passaggio informazioni 17
Librerie utilizzate 18
Bootstrap 3.0.3 18
Google Api 18
Mappa 18
Conversione indirizzo in coordinate 18
Geolocalizzazione 19

Idea generale

Problema

Nel corso dei nostri anni nella struttura universitaria, ci siamo ritrovati spesso ad analizzare i volantini sulle bacheche alla ricerca di eventi interessanti, incontri o anche solo seminari riguardanti il nostro corso di studi. I volantini spesso appaiono disordinati e rischiano di strapparsi in breve tempo o essere coperti da nuove affissioni prima che abbiano ricevuto la giusta attenzione. L'attività stessa di affiggere volantini su più bacheche, in diverse strutture, comporta anche un discreto dispendio di risorse cartacee oltre che finanziarie.

Possiamo fare qualcosa per ottenere una visuale più pratica sugli eventi e gli incontri che ci circondano? Da questa domanda nasce il nostro progetto.

EventAround

Il nostro applicativo web si pone l'obbiettivo di semplificare la ricerca agli utenti di attività formative e ludiche.

Sfruttando informazioni come data e luogo delle attività si possono fare ricerche mirate non solo definendo una determinata posizione ma anche una distanza massima che si vuole percorrere e un intervallo di giorni dalla data odierna.

Come per qualsiasi bacheca pubblica il monopolio dell'affissione dei volantini non sarà riservato solamente agli amministratori del sistema ma un qualsiasi utente registrato potrà pubblicare la propria proposta e questa sarà subito visibile da tutti.

Ogni utente registrato nel sistema potrà gestire gli eventi che gli interessano attraverso un'area personale, potrà modificare i propri eventi pubblicati o semplicemente decidere quali informazioni rendere pubbliche della propria persona agli altri utenti.

Per favorire la crescita di una comunità che si auto gestisca sarà presente anche un sistema di messaggistica interna per permettere un dialogo tra gli utenti.

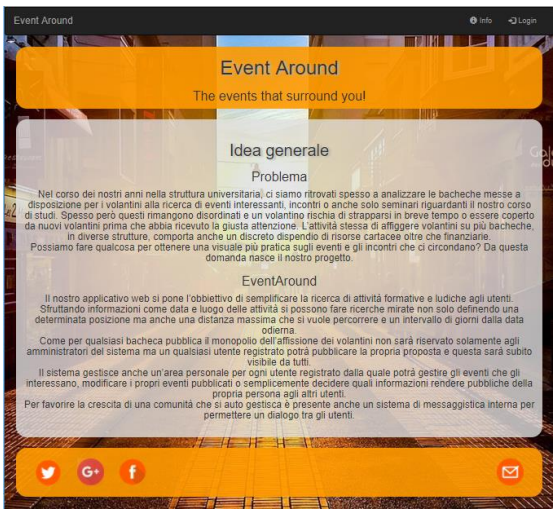
Link

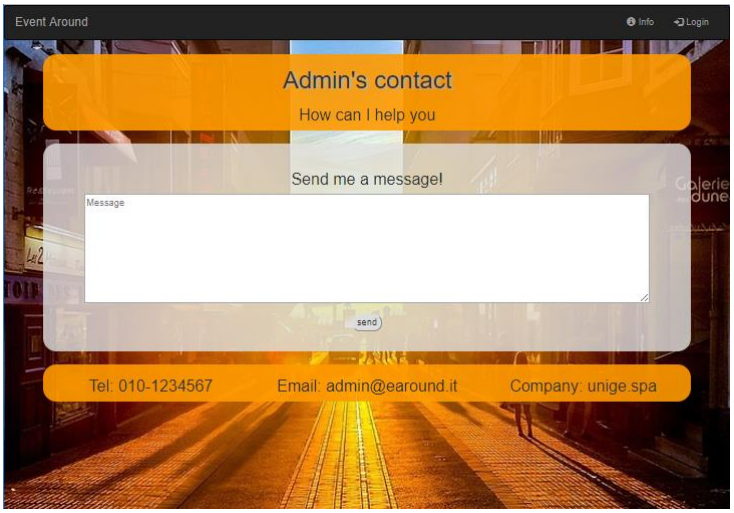
https://webdev.dibris.unige.it/~S4116422

Funzionalità Visitatore

Informazioni sul sito

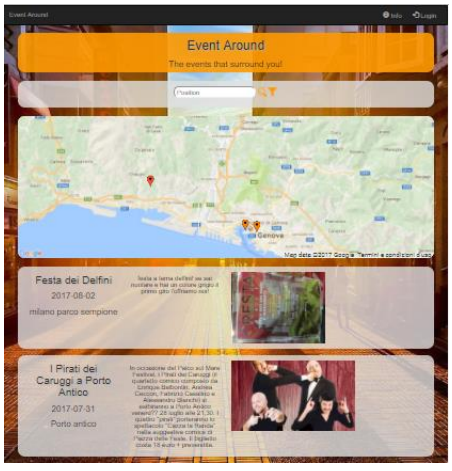
Per introdurre lo scopo dell'applicativo ad un utente al primo accesso abbiamo strutturato una pagina contenente le informazioni generali delle funzionalità disponibile e che descrivono brevemente l'intero sito. Per favorire un'esperienza utente più coinvolgente abbiamo inserito i collegamenti ai vari portali di comunicazione come facebook, twitter e google plus e una pagina che permetta di contattare l'amministratore per qualsiasi problematica riscontrata, accessibili in qualsiasi momento della navigazione.

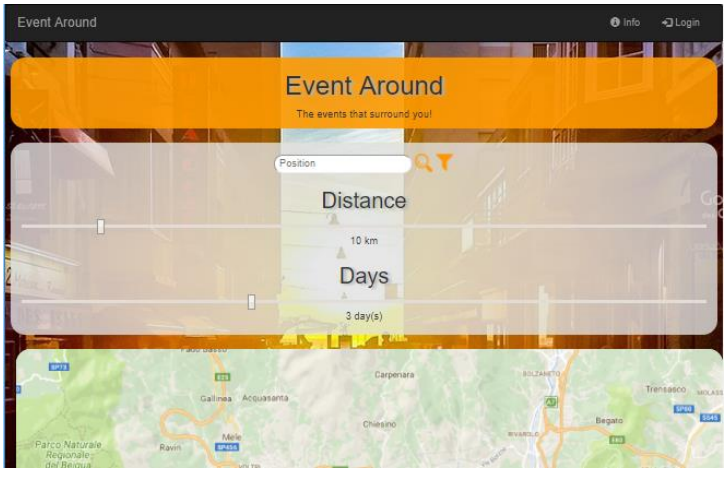




Ricerca eventi

Un utente ospite del sistema potrà usufruire solamente della ricerca degli eventi.
Al primo accesso nella homepage del sito saranno visualizzati i dieci eventi più seguiti indicati sia sulla cartina che nell'elenco sottostante.
In seguito le ricerche potranno essere effettuate utilizzando la propria posizione, se si dispone del servizio di geolocalizzazione sul dispositivo, o inserendo un qualsiasi indirizzo o nome di città.



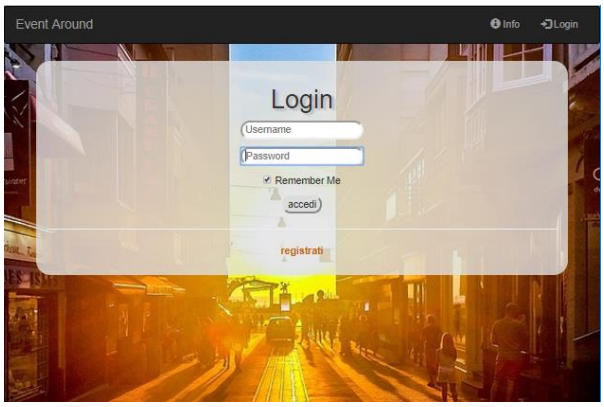


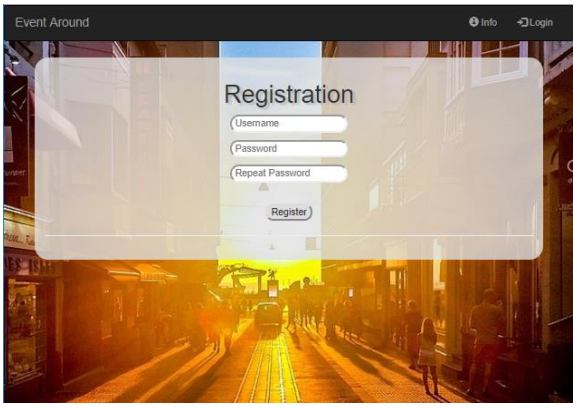
Utilizzando l'apposito pulsante si possono visualizzare i parametri della ricerca che potranno essere impostati secondo le preferenze dell'utente semplicemente trascinando il cursore sulla barra del relativo campo.

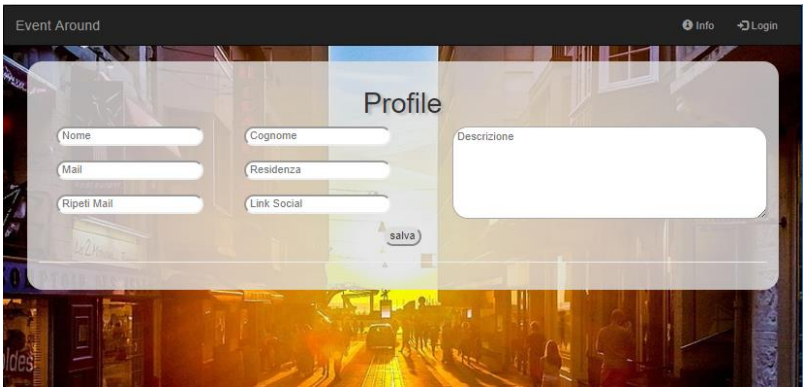
Registrazione al sito

Un utente può decidere di accedere a funzionalità aggiuntive registrandosi al sistema. Il form di registrazione è accessibile dalla pagina di login e viene strutturato in due fasi. Nella prima vengono definiti username e password mentre nella seconda viene inserita obbligatoriamente la email e come campi aggiuntivi nome, cognome, residenza, link al proprio social network o sito e una breve descrizione. Tutti i campi inseriti nella seconda fase potranno essere modificati in seguito ad eccezione dell'indirizzo email.

I campi inseriti dall'utente dovranno rispettare determinati parametri per poter garantire una maggiore sicurezza e rispettare i formati richiesti quali email e link.



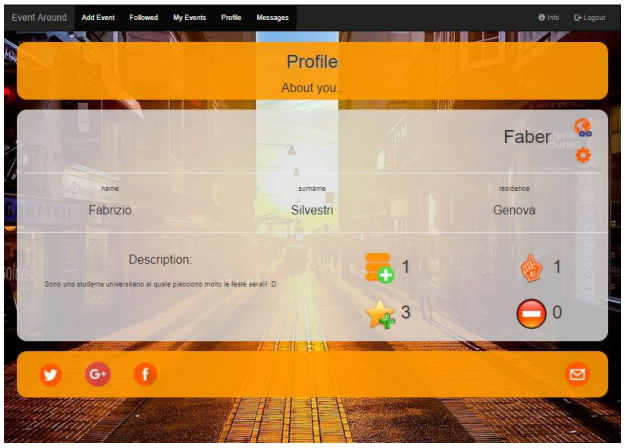


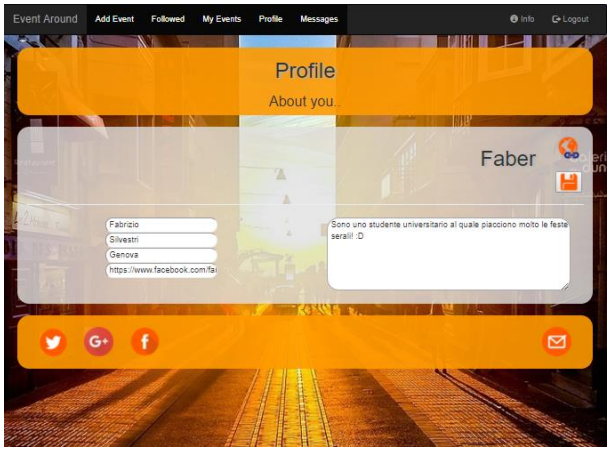


Funzionalità Utente

Profilo

In seguito alla registrazione ogni utente avrà a disposizione una pagina profilo nel quale saranno raccolte tutte le informazioni personali e le statistiche delle attività sul sistema. Le prime saranno modificabili in qualsiasi momento tramite l'apposito pulsante che attiverà un form per aggiornarli, mentre le ultime sono sintetizzate da quattro icone che rappresentano gli eventi inseriti, gli eventi seguiti, gli utenti che seguono i propri eventi e il numero di segnalazioni. Questi dati saranno visibili da tutti gli utenti registrati.





Ricerca plus

Un utente ha a disposizione funzionalità aggiuntive sugli eventi risultanti da una ricerca nella pagina principale oltre a ottenere le informazioni base.

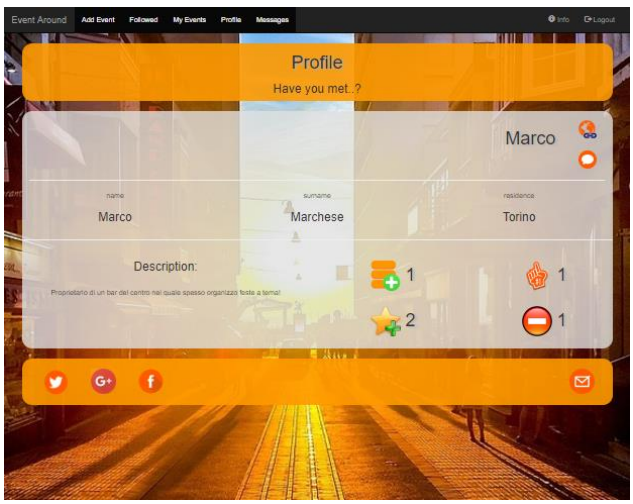
Utilizzando la stella blu sulla sinistra dell'evento avrà la possibilità di mostrare il proprio interesse per quell'attività che rimarrà salvata nell'area degli eventi seguiti e quindi più facilmente accessibile in futuro. Potrà visionare il profilo dell'utente che ha creato l'evento per controllare l'affidabilità tramite le sue statistiche. Infine avrà la possibilità di segnalare all'amministratore gli eventi che rappresentano un contenuto inadatto al sistema utilizzando il pulsante segnala sulla destra.

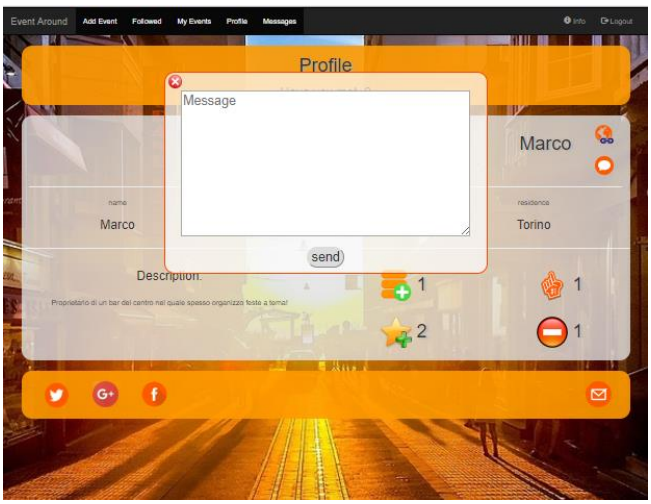
Eventi seguiti

Ogni evento per cui un utente mostrerà interesse verrà conservato nell'area "*Followed*" per semplificarne il ritrovamento in seguito senza dover ripetere la ricerca. In questa pagina gli eventi hanno lo stesso formato di quelli nella Ricerca Plus e l'utente potrà decidere se continuare a seguirli o rimuoverli dai preferiti sfruttando nuovamente la stella blu.

Visualizzare profilo utenti

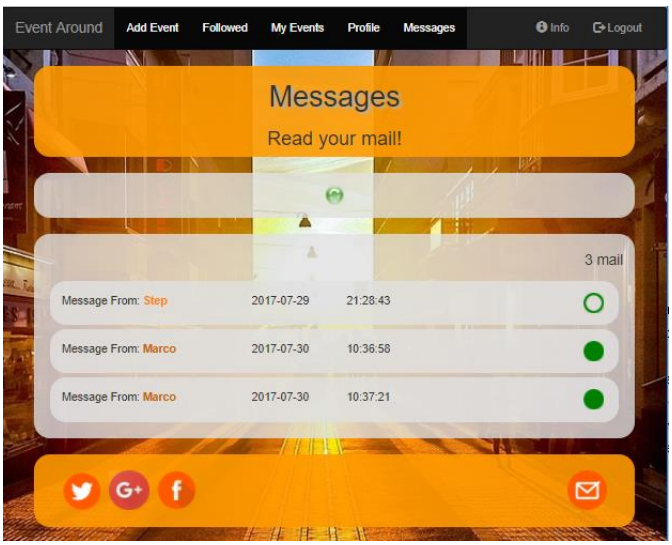
Come anticipato in precedenza, durante la ricerca si può accedere al profilo dell'utenti proprietario di un evento. La pagina visualizzata conterrà tutte le informazioni personali che l'utente avrà inserito e i dati statistici nello stesso formato della propria pagina profilo. In aggiunta ci sarà la possibilità di contattare l'utente mandandogli un messaggio accedendo a questa funzionalità tramite il pulsante "*Message*".

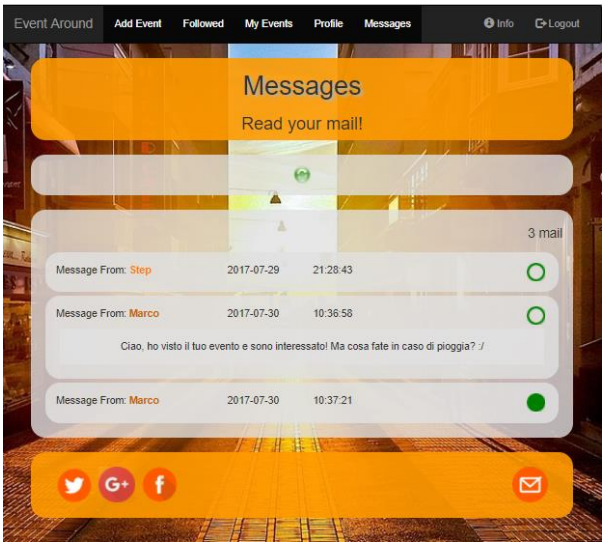




Area messaggi

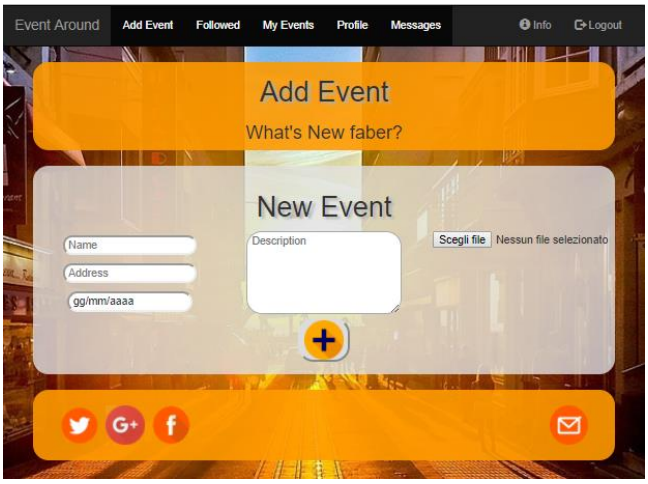
I messaggi inviati dagli altri utenti saranno raccolti nell'area "*Messages*". In questa pagina sarà possibile visualizzare il numero totale di messaggi ricevuti e distinguere quelli non ancora letti da quelli letti tramite un pallino verde affianco ai dati del messaggio, questo sarà pieno nel primo caso e vuoto nel secondo. Inizialmente si visualizzano solamente il mittente, la data e l'ora del messaggio. Per poter visualizzare il contenuto bisogna premere su queste informazioni perché l'area si espandi e mostri il testo.





Aggiungere eventi

Ogni utente avrà la possibilità di aggiungere il proprio volantino nel sistema. Recandosi nell'area "Add Event" potrà compilare tutti i dati necessari per salvare l'evento con l'unica condizione di non poterne inserire in un luogo già impegnato lo stesso giorno. In seguito all'inserimento con successo dell'evento nel sistema, questo sarà immediatamente visibile da tutti gli altri utenti.



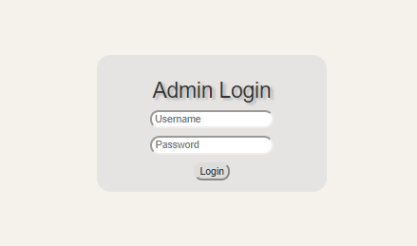
Gestione dei propri eventi

Gli eventi inserite saranno raccolti nell'area "My Events" per semplificarne la gestione da parte del proprietario. Dopo aver aggiunto un evento nel sistema si potrà modificare la sua descrizione per aggiungere ulteriori informazioni e specificare eventuali cambiamenti di programma oppure cancellare l'evento. Tutte queste operazioni saranno intuitive grazie alle icone presenti sul lato degli eventi.

Funzionalità amministratore

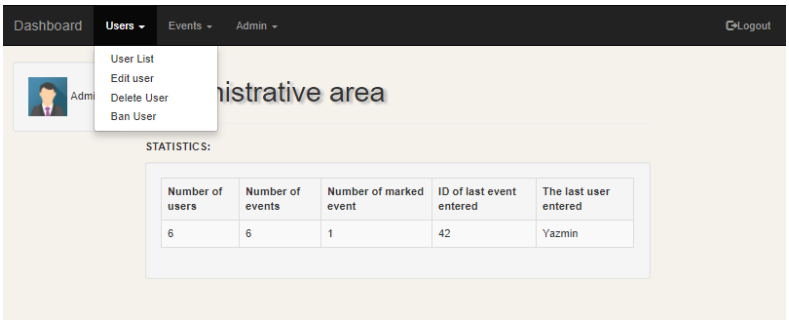
Accesso

L'area amministrativa del sito è gestita da un login che ne controllerà gli accessi. Tramite dei controlli nelle singole pagine, un qualsiasi accesso tramite url alle risorse contenute in quest'area senza avere la corretta autorizzazione sarà reindirizzato in questa pagina.



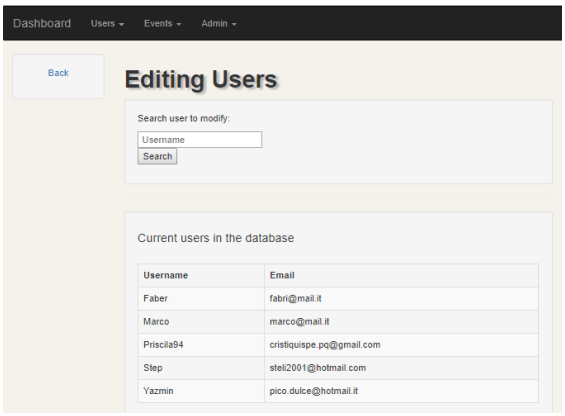
Dashboard

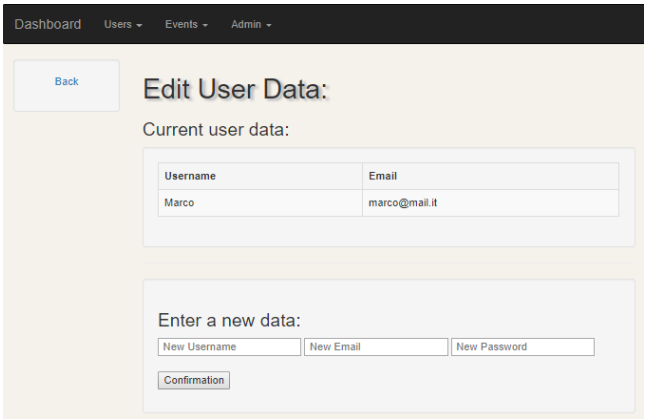
Quando un amministratore verificherà correttamente le proprie credenziali accederà all'area "*Dashboard*" nella quale saranno disponibili alcuni dati statistici e gli accessi a tutte le funzionalità per gestire i dati di utenti ed eventi tramite il menu iniziale. Sarà presente anche un'area per poter aggiungere nuovi amministratori al sistema.



Funzionalità utenti

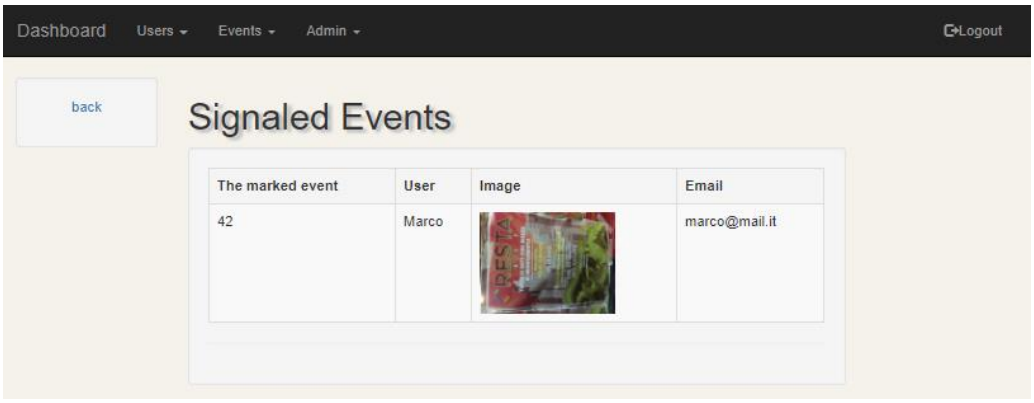
In questa sezione dell'area amministrativa si possono svolgere operazioni quali visionare le informazioni degli utenti iscritti al sistema ed eventualmente modificarle su richiesta, gestire la cancellazione di utenti inattivi o creati erroneamente ed infine sospendere un utente in seguito ad azioni scorrette o su segnalazione di altri utenti. Essendo funzionalità con interfaccia molto simile prendiamo come esempio delle funzionalità descritte le seguenti immagini della modifica degli utenti.





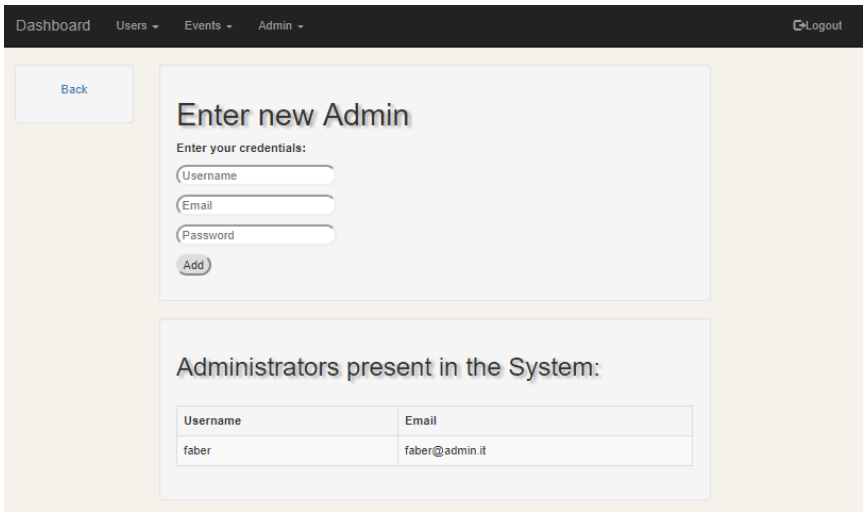
Funzionalità eventi

Analogamente alla sezione dedicata alla gestione utenti, in quest'area saranno gestite le funzionalità riguardanti gli eventi. Un amministratore potrà visionare l'elenco degli eventi inseriti nel sistema e cancellare quelli passati o inadatti al sito. Avrà anche a disposizione un'area specifica per visionare tutti gli eventi segnalati dagli utenti nella quale saranno presenti i contatti del proprietario. Riportiamo per esempio l'interfaccia dell'area "*Signaled Events*".



Funzionalità amministratore

Ogni amministratore avrà la possibilità di visionare l'elenco degli amministratori presenti e delegare il proprio ruolo ad altre persone inserendone le credenziali nell'area "*Add Admin*".



Struttura database

Il database utilizzato dall'applicativo è composto da sette tabelle, cinque di queste contengono dati mentre le restanti rappresentano la relazione tra gli eventi e gli utenti come le segnalazioni e i preferiti.

Admin

- **Username**
- *Email*
- Password

Events

- **Id**
- Name
- Description
- *Day*
- Address
- *Lat*
- *Lon*
- Owner

Users

- **Username**
- Password
- Banned

Profiles

- Username
- Name
- Surname
- *Email*
- Residence
- Link
- Description

Followed

- Id
- Username

Signaled

- Id
- Username

Messages

- **Id**
- Receiver
- Sender
- Day
- Time
- Text
- Isread

Specifiche tecniche

Sicurezza

Gestione accesso

La gestione degli accessi alle aree protette viene controllata sfruttando la sessione dell'utente. In seguito ad un login corretto o ad una corretta registrazione, verrà generata una variabile "*EAauthorized*" nel caso del sito principale ed "*EAadmin*" nel caso dell'area amministrativa. Questa verrà controllata in ogni file che gestisce le funzionalità protette, dalla visualizzazione del profilo agli script per comunicare con il database, e tramite una variabile *logged* verrà gestita la visualizzazione di sole parti pubbliche della pagina o il re indirizzamento alla homepage. In seguito un esempio della porzione di codice che inizializza la variabile durante il login dell'utente e la sua verifica.

```php
if(password_verify($password, $result["password"])) {
    session_start();
    $_SESSION["EAauthorized"] = 1;
    $_SESSION["EAusername"] = $username;
    if ($keep === "keep") {
        $cookie_username = "EAusernameC";
        setcookie($cookie_username, $username, expire: time() + 86400, path: "/");
    }
    header( string: "Location: ../pageHomepage.php");
} else {
    throw new Exception();
}
```

```php
if(!empty($_SESSION["EAauthorized"]) && $_SESSION["EAauthorized"] == 1) {
    if(!empty($_SESSION["EAusername"])) {
        $logged = 1;
        $username = $_SESSION["EAusername"];
    }

}
```

Riportiamo anche un utilizzo della variabile *logged* per controllare l'accesso all'area "*MyEvents*".

```php
if(!isset($logged)||!$logged)
    header( string: "Location: pageHomepage.php");
```

Validazione input

Per garantire una protezione del sistema da eventuali input dannosi inseriti erroneamente o volontariamente da un utente eseguiamo verifiche in diversi livelli. Come primo controllo utilizziamo la validazione degli input con alcune funzionalità messe a disposizione da HTML5, in seguito utilizziamo funzioni Javascript lato client ed infine eseguiamo le verifiche con PHP lato server. Oltre ai controlli di campi vuoti o con dimensioni diverse da quelle richieste, abbiamo

deciso di validare gli input gestiti dagli utenti con espressioni regolari per garantirci un ulteriore sicurezza. In seguito riportiamo l'esempio dei controlli eseguiti sulla password inserita nella fase di registrazione.

Controllo a livello HTML5:

```
<p>
    <input type="password" name="password1" id="password1" placeholder="Password" required>
</p>
<p>
    <input type="password" name="password2" id="password2" placeholder="Repeat Password" required>
</p>
```

Controllo a livello JS:

```
var pass_regexp = new RegExp("/^(?=.*\d)(?=.*[!@#$%^&*])(?=.*[a-z])(?=.*[A-Z])[0-9a-zA-Z]{8,}$/");
if(pass_regexp.test(pass1)) {
    document.getElementById("password1").focus();
    document.getElementById("formError").innerText = "Password not valid";
    return false;
}
if(pass1 !== pass2) {
    document.getElementById("password2").focus();
    document.getElementById("formError").innerText = "Password not equals";
    return false;
}
return true;
```

Controllo a livello PHP:

```
$password = trim($_POST["password1"]);
if(empty($password)){
    $message = "Password Errror";
    throw new Exception();
}
if (!preg_match( pattern: "/^(?=.{8,})(?=.*[a-z])(?=.*[A-Z])(?=.*[\d])(?=.*[\W]).*$/", $password)){
    $message = "Password not valid";
    throw new Exception();
}
$passwordDB = password_hash($password,  algo: PASSWORD_BCRYPT);
```

Database

Interazioni

Per comunicare con il database abbiamo utilizzato l'estensione PDO di PHP che definisce un'interfaccia leggera e coerente per l'accesso ai database. Ogni script che richiede le informazioni del database è strutturato in tre parti. La prima consiste nella validazione degli input e in caso di errore interrompe l'esecuzione per segnalarlo all'utente. La seconda apre la connessione con la base di dati per eseguire le interrogazioni necessarie, interrompendosi in caso di errori. Infine l'ultima fase consiste nel restituire la risposta dell'esecuzione che può essere positiva o una segnalazione dell'errore riscontrato. Un esempio di questa procedura è lo script per restituire gli eventi seguiti da un utente.

```php
<?php
    require("accessControl.php");
    $eventDB = array(); $count = 0; $message = "";
    require("dbAccess.php");
    try {
        if(!isset($_SESSION["EAusername"])) {
            $message = "username non valido";
            throw new Exception();
        }
        $username = $_SESSION["EAusername"];
        $conn = new PDO( dsn: "mysql:host=$server;dbname=$dbName", $dbUser, $dbPass);
        $conn->setAttribute( attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION);
        $stmt = $conn->prepare(
             statement: "SELECT id, name, description, address, day, lat, lon FROM Events WHERE owner = :username");
        $stmt->bindParam( parameter: ":username", $username);
        $stmt->execute();
        $eventDB[$count++] = array();
        while ($row = $stmt->fetch( fetch_style: PDO::FETCH_ASSOC)) {
            $image_file = "../uploads/default.jpg";
            if (file_exists( filename: "../uploads/" . $row["id"] . ".jpg"))
                $image_file = "../uploads/" . $row["id"] . ".jpg";
            $eventDB[$count++] = array(
                "id" => $row["id"], "name" => $row["name"], "description" => $row["description"],
                "address" => $row["address"], "day" => $row["day"], "lat" => $row["lat"],
                "lon" => $row["lon"], "image" => $image_file
            );
        }
        echo json_encode($eventDB, options: JSON_PRETTY_PRINT);
        $conn = null;
    } catch(PDOException $e) { echo "[{}]; /*ERROR ".$e->getMessage()."*/";
    } catch(Exception $e) { echo "[{}]; /*".$message."*/"; }
?>
```

Gestione concorrenza

Per garantire l'assenza di errori durante l'accesso alla base di dati abbiamo utilizzato le transazioni. Queste ci permettono di eseguire letture dei dati con l'esecuzione di più interrogazioni senza che questi vengano modificati da altri utenti che accedono concorrentemente. Abbiamo adottato questa soluzione solamente per gli script che eseguono più di un'interrogazione in quanto negli altri casi risulterebbe superfluo. Prendiamo come esempio il codice che gestisce l'aggiunta di un evento al sistema dove viene controllata la disponibilità della data e del luogo e la presenza dell'utente che lo sta creando nel database prima di confermarne l'inserimento.

```php
try{

    $conn = new PDO( dsn: "mysql:host=$server;dbname=$dbName", $dbUser, $dbPass);
    $conn->setAttribute( attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION);
    $conn->beginTransaction();

    $stmt = $conn->prepare( statement: "SELECT id FROM Events WHERE day = :day AND lat = :lat AND lon = :lon");
    $stmt->bindParam( parameter: ":day",$day);
    $stmt->bindParam( parameter: ":lat",$lat);
    $stmt->bindParam( parameter: ":lon",$lon);
    $stmt->execute();
    if($stmt->rowCount() > 0) {
        $message="Day and place already used";
        throw new Exception();
    }
    $stmt = $conn->prepare( statement: "SELECT username FROM Users WHERE username = :username");
    $stmt->bindParam( parameter: ":username",$owner);
    $stmt->execute();
    if($stmt->rowCount() !== 1) {
        $message="User ".$owner." not valid";
        throw new Exception();
    }
```

```php
        $conn->commit();
    } catch(PDOException $e) {
        $conn->rollBack();
        $message = "Database error";
    } catch (Exception $e) {
        $conn->rollBack();
    }
    $conn = null;
    header( string: "Location: ../pageAddEvent.php?message=".$message);
```

Passaggio informazioni

All'interno della nostra applicazione web utilizziamo per l'interscambio di dati fra il server e il client il formato JSON. Questo viene utilizzato sia per generare una variabile array bidimensionale interrogabile durante la scrittura della pagina dal server, sia per inizializzare una variabile Javascript globale che verrà utilizzata per eseguire modifiche della pagina dinamicamente. Il primo utilizzo lo riserviamo alle pagine dinamiche dove scriviamo informazioni statiche per l'utente, un esempio è la pagina dell'area "*Messages*".

```
echo '<div class="row text-right">';
echo '    <div class="col-sm-12">';
echo '        <h4>' . count($messages) . ' mail</h4>';
echo '    </div>';
echo '</div>';
for ($j = 0; $j < count($messages); $j++) {
    $message = $messages[$j];

    echo '<div class=" marginMin liteBackground borderRadius" onclick="showMessage(\''.$message["id"].'\',\''.$message["read
    echo '<div class="row">';
    echo '    <div class="col-sm-4">Message From: <a href="pageOtherProfile.php?username='.$message["sender"].'" target="_bla
    echo '    <div class="col-sm-2">';
    echo '        <div class=row">'.$message["day"].'</div>';
    echo '    </div>';
    echo '    <div class="col-sm-2">';
    echo '        <div class=row">'.$message["time"].'</div>';
    echo '    </div>';
    echo '    <div class="col-sm-3"></div>';
    echo '    <div class="col-sm-1" id="read'.$message["id"].'">';
    if($message["read"] === '0') {
        echo '<img src="../media/unread.png" alt="unread">';
    } else {
        echo '<img src="../media/read.png" alt="read">';
    }
    echo '    </div>';
    echo '</div>';
    echo '  <div id="'.$message["id"].'"  class="message text-center marginMin liteBackground">'.$message["text"].'</div>';
    echo '</div>';
}
```

La modalità invece che coinvolge la variabile Javascript viene utilizzata per permettere una maggiore fluidità dell'esperienza utente durante la navigazione ed è riservata per le funzionalità dove avviene un frequente aggiornamento dei dati. Un esempio di questo utilizzo lo troviamo nella homepage del sito dove inizialmente salviamo gli eventi più seguiti nel sistema ed in seguito aggiorniamo i risultati ogni volta che l'utente esegue una ricerca. Questo ci permette una visualizzazione degli eventi senza dover ricaricare la pagina migliorando l'esperienza utente.

```
<script>
    var events = <?php require("script/mainEvents.php"); ?>;
</script>
```

```
function drawEventsList(check) {
    if(events.length <= 1) {
        text  = '<div class="row">';
        text += '    <div class="col-sm-12 marginMin text-center liteBackground borderRadius">';
        text += '        <h3>Nessun Evento</h3>';
        text += '    </div>';
        text += '</div>';
    }else{
        text = '';
        for(j=1;j<events.length;j++)
        {
            text += drawSingleEvent(j, check);
        }
    }
    if (text !== null)
        document.getElementById('eventsList').innerHTML = text;
}
```

Librerie utilizzate

Bootstrap 3.0.3

Per disegnare l'interfaccia molto minimale del nostro applicativo web ci siamo appoggiati al framework Bootstrap nella versione 3.0.3. Questo utilizzo ci ha permesso di ottenere più velocemente risultati nella parte grafica e poter investire più risorse per la gestione delle funzionalità. Nonostante il suo utilizzo però abbiamo comunque generato un nostro file di stile per poter gestire delle necessità più specifiche.

Google Api

Mappa

Per poter sfruttare la mappa di Google Maps all'interno del nostro sito ci siamo appoggiati ai servizi offerti da Google. Utilizzando una chiamata Javascript abbiamo quindi delegato la creazione della mappa per poi personalizzarla in seguito con la funzione richiamata nel campo "*callback*". Lo script che abbiamo utilizzato è il seguente.

```
<script src="https://maps.googleapis.com/maps/api/js?key=KEY&callback=myMap"></script>
```

Conversione indirizzo in coordinate

Per eseguire una ricerca bisogna inserire una determinata posizione espressa come indirizzo o nome di una città ma questo non sarebbe sufficiente per poter eseguire una ricerca efficiente nel sistema. Pertanto sia gli indirizzi utilizzati nelle ricerche che quelli per indicare la posizione degli eventi sul sistema saranno convertiti in latitudine e longitudine. Questa conversione viene eseguita appoggiandosi nuovamente ad un servizio di Google al quale fornito un indirizzo restituisce informazioni relative alla posizione tra cui la latitudine e la longitudine che noi utilizziamo.

```
$address = urlencode($position);
$request_url = "http://maps.googleapis.com/maps/api/geocode/xml?address=" . $address . "&sensor=true";
if (!($xml = simplexml_load_file($request_url))) {
    echo '[]';
    exit;
}
$status = $xml->status;
if ($status == "OK") {
    $lat = $xml->result->geometry->location->lat;
    $lon = $xml->result->geometry->location->lng;
} else {
    echo '[]';
    exit;
}
```

Geolocalizzazione

La ricerca della posizione del dispositivo utilizzato dall'utente viene eseguita interrogando l'oggetto navigator.geolocation ed utilizzando i dati da esso forniti per indicare l'area di ricerca senza che l'utente debba indicare il proprio indirizzo. Questa funzionalità però è garantita solamente se si accede al sito con il protocollo per il trasferimento delle pagine HTTPS. In seguito queste informazioni vengono utilizzate all'interno di una chiamata asincrona per aggiornare la pagina.

```
function searchEvents() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(setLatLon, err);
    } else {
        console.error("geolocalizetion not supported");
        sendSearch(null,null);
    }
}
```

```
function setLatLon(position) {
    lat = position.coords.latitude;
    lon = position.coords.longitude;
    sendSearch(lat, lon);
}
```

```
function sendSearch(lat, lon) {
    urlSimple = "script/eventsSearcher.php";

    position = document.getElementById('position').value;
    distance = document.getElementById('distance').value;
    days = document.getElementById('days').value;

    url = urlSimple+"?position="+position+"&distance="+distance+"&days="+days+"&lat="+lat+"&lon="+lon;
    xhr = getXMLHttpRequestObject();
    xhr.onreadystatechange = searchCallback;
    xhr.open('GET',url,true);
    xhr.send(null);
    if(!hider){
        showFilter();
    }
}
```

```
function searchCallback() {
    if (xhr.readyState === 4) {
        if (xhr.status === 200) {
            if (xhr.responseText !== "") {
                events = JSON.parse(xhr.responseText);
                showMap(events[0].lat, events[0].lon);
                drawEventsList();
                document.getElementById("searchForm").scrollIntoView();
            }
            else {
                console.error("Ajax error: no data received");
                events = [];
            }
        }
        else {
            console.error("Ajax error: " + xhr.responseText);
            events = [];
        }
    }
}
```