Semplice registrazione

e login degli utenti

SOMMARIO

Una della richieste più frequenti degli utenti che si avvicinano al php è sicuramente lo sviluppo di un **sistema di login**.

Le funzionalità che dovrà svolgere tale applicativo saranno almeno tre:

1. registrazione degli utenti;
2. il login degli utenti;
3. impedire l'accesso a pagine o contenuti agli utenti non loggati.

Il semplice script che oggi vi propongo vi consentirà di implementare nei vostri siti tali funzionalità.

Il database

view plainprint?

```
1. CREATE TABLE IF NOT EXISTS `users` (
2.   `id` int(11) NOT NULL AUTO_INCREMENT,
3.   `username` varchar(100) NOT NULL,
4.   `pass` varchar(50) NOT NULL,
5.   `email` varchar(255) NOT NULL,
6.   `data_reg` datetime NOT NULL,
7.   PRIMARY KEY (`id`)
8. ) ENGINE=MyISAM  DEFAULT CHARSET=latin1
```

La classe

view plainprint?

```
1. <?php
2. // error_reporting(E_ALL | E_DEPRECATED | E_STRICT);
3.
4. Class Users{
5.     /*********************************
6.     SETTING
7.     *********************************/
8.     // le credenziali di accesso al database
9.     private $host_db = 'localhost';
10.    private $user_db = 'root';
11.    private $pass_db = '';
12.    private $name_db = 'test';
13.    // gli url che gestinranno le operazioni di login
14.    public $Urls = array(
15.                    'login_page'    => 'http://localhost/test/guida_semple/login.php',
16.                    'register_page' => 'http://localhost/test/guida_semple/registrazione.p
    hp',
17.                    'logout_page'   => 'http://localhost/test/guida_semple/logout.php'
18.                    );
19.
20.    /*****************************************
21.    se non sai ciò che fai non toccare più nulla
22.    *****************************************/
23.    /*risorse di connessione*/
24.    protected $conn;
```

```
25.    protected $selezione_db;
26.
27.    /*variabili di registrazione*/
28.    protected $reg_username;
29.    protected $reg_email;
30.    protected $reg_pass;
31.    protected $reg_confirm_pass;
32.    protected $reg_crypt_pass;
33.
34.    /*variabili di login*/
35.    protected $login_username;
36.    protected $login_password;
37.    protected $login_cryptpass;
38.    protected $login_iduser;
39.
40.    /*variabili per gestire gli errori*/
41.    public $messages = array(
42.                    1 => 'Il campo username è obbligatorio.',
43.                    2 => 'Il campo email è obbligatorio.',
44.                    3 => 'Il campo password è obbligatorio.',
45.                    4 => 'Le due password non coincidono.',
46.                    5 => 'Il campo username contiene caratteri non validi. Sono consentiti
   solo lettere, numeri il i seguenti simboli . _ -.',
47.                    6 => 'Inserisci una email con sitassi corretta.',
48.                    7 => 'La password scelta è eccessivamente breve. Scegli una passwor
   d di almeno 8 caratteri.',
49.                    8 => 'Esiste già un utente registrato con questo username.',
50.                    9 => 'Esiste già un utente registrato con questa email.',
51.                    10 => 'Registrazione effettuata con successo.',
52.                    11 => 'Login errato',
53.                    12 => 'Login eseguito con successo.',
54.                    13 => 'Logout eseguito con successo.',
55.                    14 => 'Per accedere a questa pagina occorre essere loggati.'
56.                    );
57.
58.    public $message_script;
59.
60.    // il costruttore attiva la connessione a mysql
61.    public function __construct(){
62.        $this->connessione();
63.        }
64.    /******************
65.    CONNESSIONE A MYSQL
66.    ******************/
67.    protected function connessione(){
```

```
68.         $this->conn = mysql_connect($this->host_db, $this->user_db, $this-
    >pass_db) or die(mysql_error());
69.         $this->selezione_db = mysql_select_db($this->name_db, $this->conn) or die(
    mysql_error());
70.         return TRUE;
71.         }
72.
73.     /************************************
74.     ALCUNI METODI PER ESEGUIRE VALIDAZIONI
75.     ************************************/
76.
77.     // verifica campo generico non vuoto (TRUE se non vuoto)
78.     public function empty_string($string){
79.         $string = trim($string);
80.         if($string==''){
81.             return TRUE;
82.             }
83.         else{
84.             return FALSE;
85.             }
86.         }
87.
88.     // verifica sintassi username
89.     public function is_username($username){
90.         $regex = '/^[a-z0-9\.\-_]{3,30}$/i';
91.         return preg_match($regex, $username);
92.         }
93.
94.     // verifica sintassi email (TRUE se ok)
95.     public function is_email($email){
96.         $regex = '/^[_a-zA-Z0-9-]+(\.[_a-zA-Z0-9-]+)*@[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)+
    $/';
97.         return preg_match($regex, $email);
98.         }
99.
100.    // verifica sintassi password (per semplicità solo lunghezza) (TRUE se ok)
101.    public function is_secure_password($password){
102.        if(strlen($password)>=8){
103.            return TRUE;
104.            }
105.        else{
106.            return FALSE;
107.            }
108.        }
```

```
109.  /*****************************************************
110.  METODI PER VERIFICARE ESISTENZA DI USERNAME E PASSWORD
111.  *****************************************************/
112.
113.  // verifica esistenza username (TRUE se esiste)
114.  public function isset_username($username){
115.     $query = "SELECT COUNT(username) AS count
116.              FROM users
117.              WHERE username='".mysql_real_escape_string($username)."'
118.              LIMIT 1";
119.     $result = mysql_query($query) or die(mysql_error());
120.     $row = mysql_fetch_array($result);
121.     if($row['count']==1){
122.        return TRUE;
123.        }
124.     else{
125.        return FALSE;
126.        }
127.     }
128.
129.  // verifica esistenza email (TRUE  se esiste)
130.  public function isset_email($email){
131.     $query = "SELECT COUNT(email) AS count
132.              FROM users
133.              WHERE email='".mysql_real_escape_string($email)."'
134.              LIMIT 1";
135.     $result = mysql_query($query) or die(mysql_error());
136.     $row = mysql_fetch_array($result);
137.     if($row['count']==1){
138.        return TRUE;
139.        }
140.     else{
141.        return FALSE;
142.        }
143.     }
144.
145.  /*****************************
146.  I FORM DI LOGIN E REGISTRAZIONE
147.  *****************************/
148.  public function get_login_form(){
149.     $html = '
150.        <form action="' .$this->Urls['login_page']. '" method="post" id="form_login"
   >
151.        <fieldset>
152.        <legend>Login<legend>
153.        <label for="login_user">Username</label>
```

```
154.            <input type="text" name="username" id="login_user" />
155.        <label for="login_pass">Password</label>
156.            <input type="password" name="pass" id="login_pass" />
157.        <input type="submit" name="login" value="login" id="login_submit"/>
158.        </fieldset>
159.        </form>';
160.      return $html;
161.      }
162.
163.  public function get_register_form(){
164.      $html = '
165.        <form action="' .$this->Urls['register_page']. '" method="post" id="form_reg
  ister">
166.        <fieldset>
167.        <legend>Registrazione<legend>
168.        <label for="reg_user">Username*</label>
169.            <input type="text" name="username" id="reg_user" />
170.        <label for="reg_email">Email*</label>
171.            <input type="text" name="email" id="reg_email" />
172.        <label for="reg_pass1">Password*</label>
173.            <input type="password" name="pass1" id="reg_pass1" />
174.        <label for="reg_pass2">Confirm Password*</label>
175.            <input type="password" name="pass2" id="reg_pass2" />
176.        <input type="submit" name="register" value="registra" id="reg_submit" />
177.        <input type="reset" name="reset" value="cancella" id="reg_reset" />
178.        </fieldset>
179.        </form>';
180.      return $html;
181.      }
182.
183.  /*****************************
184.  LINK LOGOUT
185.  *****************************/
186.  public function get_link_logout(){
187.      if($this->is_logged()){
188.          return '<a href="'.$this->Urls['logout_page'].'" class="logout">Logout</a>';
189.          }
190.      return '';
191.      }
192.
193.  /*****************************
194.  METODO PER CRIPTARE LE PASSWORD
195.  *****************************/
196.  public function crypt_pass($pass){
197.      return sha1($pass);
198.      }
```

```
199.        /*****************************
200.    ESECUZIONE DELLA REGISTRAZIONE
201.    *****************************/
202.    public function esegui_registrazione(){
203.        // se il form e i suoi input sono stati inviati
204.        if(isset($_POST['register']) AND
205.            isset($_POST['username']) AND
206.            isset($_POST['email']) AND
207.            isset($_POST['pass1']) AND
208.            isset($_POST['pass2'])){
209.            //valorizziamo alcune variabili
210.            $this->reg_username = trim($_POST['username']);
211.            $this->reg_email = trim($_POST['email']);
212.            $this->reg_pass = trim($_POST['pass1']);
213.            $this->reg_confirm_pass = trim($_POST['pass2']);
214.            // criptiamo la password
215.            $this->reg_crypt_pass = $this->crypt_pass($this->reg_pass);
216.            // eseguiamo la validazione degli input
217.            $valid_input = $this->check_input_registrazione();
218.            // se sono validi
219.            if($valid_input===TRUE){
220.                // inseriemo all'interno del database i dati
221.                $this->query_insert_registrazione();
222.                // settiamo il messaggio di successo della registrazione
223.                $this->message_script = 10;
224.                return TRUE;
225.                }
226.            }
227.        return FALSE;
228.        }
229.
230.    // verifica che gli input siano corretti
231.    protected function check_input_registrazione(){
232.        if($this->empty_string($this->reg_username)){
233.            $this->message_script = 1;
234.            return FALSE;
235.            }
236.        else if($this->empty_string($this->reg_email)){
237.            $this->message_script = 2;
238.            return FALSE;
239.            }
240.        else if($this->empty_string($this->reg_pass)){
241.            $this->message_script = 3;
242.            return FALSE;
243.            }
244.        else if($this->reg_pass != $this->reg_confirm_pass){
```

```
245.          $this->message_script = 4;
246.          return FALSE;
247.          }
248.       else if(!$this->is_username($this->reg_username)){
249.          $this->message_script = 5;
250.          return FALSE;
251.          }
252.       else if(!$this->is_email($this->reg_email)){
253.          $this->message_script = 6;
254.          return FALSE;
255.          }
256.       else if(!$this->is_secure_password($this->reg_pass)){
257.          $this->message_script = 7;
258.          return FALSE;
259.          }
260.       else if($this->isset_username($this->reg_username)==TRUE){
261.          $this->message_script = 8;
262.          return FALSE;
263.          }
264.       else if($this->isset_email($this->reg_email)==TRUE){
265.          $this->message_script = 9;
266.          return FALSE;
267.          }
268.       return TRUE;
269.       }
270.
271.    // esecuzione della query insert di registrazione
272.    protected function query_insert_registrazione(){
273.       $query = "
274.                INSERT INTO users
275.                SET
276.                   username='".mysql_real_escape_string($this->reg_username)."',
277.                   pass='".mysql_real_escape_string($this->reg_crypt_pass)."',
278.                   email='".mysql_real_escape_string($this->reg_email)."',
279.                   data_reg= NOW()";
280.       $result = mysql_query($query) or die(mysql_error());
281.       return mysql_insert_id();
282.       }
283.
284.    /*******************
285.    ESECUZIONE DEL LOGIN
286.    *******************/
287.    public function esegui_login(){
288.       // se il form di login e i sui tutti input sono stati inviati
289.       if(isset($_POST['login']) AND isset($_POST['username']) AND isset($_POST
      ['pass'])){
```

```
290.          // valorizziamo delle variabili
291.          $this->login_username = trim($_POST['username']);
292.          $this->login_password = trim($_POST['pass']);
293.          // criptiamo la password
294.          $this->login_cryptpass = $this->crypt_pass($this->login_password);
295.          // validiamo i dati (non devono essere vuoti)
296.          $not_empty_input = $this->check_input_login();
297.          // se la validazione è andata a buon fine
298.          if($not_empty_input===TRUE){
299.             // eseguiamo la query e verifichiamo se individua le credenziali
300.             if($this->query_select_login()==TRUE){
301.                // settiamo lo status di utente loggato
302.                $this->set_logged($this->login_iduser);
303.                // settiamo l'username
304.                $this->set_username($this->login_username);
305.                // settiamo il messaggio di successo del login
306.                $this->message_script = 12;
307.                return TRUE;
308.                }
309.             // se la query non ha trovat utenti con quelle credenziali
310.             else{
311.                //  settiamo un messaggio di insuccesso dell'operazone
312.                $this->message_script = 11;
313.                }
314.             }
315.          }
316.       return FALSE;
317.       }
318.
319.    // verifica che gli input del login non siano vuoti
320.    protected function check_input_login(){
321.       if($this->empty_string($this->login_username)){
322.          $this->message_script = 1;
323.          return FALSE;
324.          }
325.       else if($this->empty_string($this->login_password)){
326.          $this->message_script = 3;
327.          return FALSE;
328.          }
329.       return TRUE;
330.       }
331.
332.    // esecuzione della qeury per verificare il login
333.    protected function query_select_login(){
334.       $query = "
335.                SELECT id FROM users
```

```
336.                 WHERE
337.                     username='".mysql_real_escape_string($this->login_username)."'
    AND
338.                     pass='".mysql_real_escape_string($this->login_cryptpass)."'";
339.         $result = mysql_query($query) or die(mysql_error());
340.         // se individua l'utente
341.         if(mysql_num_rows($result)==1){
342.             $row = mysql_fetch_array($result);
343.             $this->login_iduser = $row['id'];
344.             return TRUE;
345.             }
346.         return FALSE;
347.         }
348.
349.     /**********************************
350.     VERIFICA DELLO STATO DI LOGIN UTENTE
351.     **********************************/
352.
353.     // verifica login
354.     public function is_logged(){
355.         return isset($_SESSION['auth']);
356.         }
357.
358.     // set login
359.     protected function set_logged($id_user){
360.         $_SESSION['auth'] = $id_user;
361.         return;
362.         }
363.
364.     // access denied
365.     public function access_denied(){
366.         if(!$this->is_logged()){
367.             header("location: ".$this->Urls['login_page']."?message=14");
368.             exit;
369.             }
370.         return;
371.         }
372.
373.     protected function set_username($username){
374.         $_SESSION['username_logged'] = $username;
375.         return;
376.         }
377.
378.     public function get_username(){
379.         return isset($_SESSION['username_logged']) ? $_SESSION['username_logg
    ed'] : '';
```

```
380.        }
381.
382.    // logout
383.    public function logout(){
384.        session_unset();
385.        session_destroy();
386.        setcookie(session_name(), '', time()-42000, '/');
387.        header("location: ".$this->Urls['login_page']."?message=13");
388.        return;
389.        }
390.
391.    /*****************************
392.    METODO PER OTTENERE I MESSAGGI
393.    *****************************/
394.    public function get_message(){
395.        if(isset($_GET['message'])){
396.            $this->message_script = $_GET['message'];
397.            }
398.        $key = intval($this->message_script);
399.        if(array_key_exists($key, $this->messages)){
400.            return $this->messages[$key];
401.            }
402.        return FALSE;
403.        }
404.    }
405.?>
```

La pagina di registrazione

view plainprint?

```
1. <?php
2. session_start();
3. require_once('lib/Users.class.php');
4. $login = New Users;
5. if($login->esegui_registrazione()==TRUE){
6.     header("location: ".$login->Urls['login_page']."?message=".$login-
   >message_script);
7.     exit;
8.     }
9. ?>
10.<html>
11.<head>
12.<title>Register page</title>
13.<link rel="stylesheet" href="css/style.css" type="text/css" />
```

```
14.<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/j
   query.min.js"></script>
15.<script type="text/javascript" src="http://ajax.microsoft.com/ajax/jquery.validate/1.7/j
   query.validate.js"></script>
16.<script type="text/javascript" src="js/validation_reg.js"></script>
17.</head>
18.<body>
19.<div id="content">
20.<?php if($login->get_message()) : ?>
21.       <div class="message"><p><?php echo $login->get_message(); ?></p></div>
22.<?php endif; ?>
23.<?php echo $login->get_register_form(); ?>
24.</div>
25.</body>
26.</html>
```

La pagina di login

view plainprint?

```
1. <?php
2. session_start();
3. require_once('lib/Users.class.php');
4. $login = New Users;
5. $login->esegui_login();
6. ?>
7. <html>
8. <head>
9. <title>Login page</title>
10.<link rel="stylesheet" href="css/style.css" type="text/css" />
11.<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/j
   query.min.js"></script>
12.<script type="text/javascript" src="http://ajax.microsoft.com/ajax/jquery.validate/1.7/j
   query.validate.js"></script>
13.<script type="text/javascript" src="js/validation_login.js"></script>
14.</head>
15.<body>
16.<div id="content">
17.
18.<?php if($login->get_message()) : ?>
19.       <div class="message"><p><?php echo $login->get_message(); ?></p></div>
20.<?php endif; ?>
21.
22.
23.<?php if(!$login->is_logged()) : ?>
```

```
24.    <?php echo $login->get_login_form(); ?>
25.
26.
27.<?php else: ?>
28.    <p>Benvenuto <strong><?php echo $login->get_username(); ?></strong></p>
29.    <?php echo $login->get_link_logout(); ?>
30.<?php endif; ?>
31.</div>
32.</body>
33.</html>
```

Pagina di logout

```
1. <?php
2. session_start();
3. require_once('lib/Users.class.php');
4. $login = New Users;
5. $login->logout();
6. ?>
```

Pagina protetta

```
1. <?php
2. session_start();
3. require_once('lib/Users.class.php');
4. $login = New Users;
5. $login->access_denied();
6. ?>
7. <html>
8. <head>
9. <title>Pagina protetta</title>
10.<link rel="stylesheet" href="css/style.css" type="text/css" />
11.</head>
12.<body>
13.<body>
14.<h1>Questa è una pagina protetta</h1>
15.<p>Potrai accedere a questa pagina solo hai eseguito il login.</p>
16.</body>
17.</html>
```

Pagina con contenuti protetti

```
1. <?php
2. session_start();
3. require_once('lib/Users.class.php');
4. $login = New Users;
5. ?>
6. <html>
7. <head>
8. <title>Pagina con contenuti protetti</title>
9. <link rel="stylesheet" href="css/style.css" type="text/css" />
10.</head>
11.<body>
12.<body>
13.<h1>Questa è una pagina contiene contenuti protetti.</h1>
14.<p>Questi sono contenuti leggibili a tutti.</p>
15.
16.<?php if($login->is_logged()) : ?>
17.
18.    <div id="contenuti_protetti">
19.    <p>Il testo contenuto in questo div sarà visualizzabile solo agli utenti loggati.</p>

20.    </div>
21.
22.<?php endif; ?>
23.
24.</body>
25.</html>
```

Il setting e il costruttore della classe

Gli unici settaggi che la nostra classe richiede saranno i **parametri di connessione a mysql** e gli **URL delle nostre pagine di registrazione, login e logout.**

Il costruttore si occuperà di eseguire il metodo **connessione()** che (come è facile intuire) attiva la connessione al nostro database.

La registrazione

La registrazione degli utenti altro non è che l'inserimento all'interno del tabella users dei dati immessi dall'utente all'interno di un form. Quindi, abbiamo anzitutto bisogno di modulo di registrazione.

Per ottenere tale il form registrazione utilizzeremo il metodo **get_register_form()** mentre l'operazione di registrazione vera e propria verrà gestita attraverso il metodo **esegui_registrazione()**.

L'inserimento all'interno della tabella users avverrà con il metodo **query_insert_registrazione()** la quale prima di essere eseguita richiede, oltre che il criptaggio della password con il metodo **crypt_pass()**, il controllo degli input immessi dall'utente.

Infatti, l'operazione più delicata da eseguire è la verifica che i dei dati immessi siano "corretti" ed in particolare:

- username:
    1. obbligatorio;
    2. con una lunghezza minima di 3 caratteri;
    3. verifica della sintassi;
    4. verifica che non si tratti di un utente già registrato.
- email:
    1. obbligatoria;
    2. verifica della sintassi;
    3. verifica che non si tratti di un'email già registrata.
- Password
    1. obbligatoria;
    2. con una lunghezza minima di 8 caratteri.
- Conferma password
    1. Corrispondenza tra le due password

La validazione degli input

Per eseguire le operazioni di validazione all'interno della nostra classe ci serviremo di alcuni metodi ed in particolare:

- **empty_string()** riceve come parametro una stringa e se vuota restituisce TRUE;
- **is_username()** verifica se la stringa ricevuta come parametro ha una sintassi corretta per l'username; in particolare verificherà che questo abbia una lunghezza compresa fra 3 e 30 caratteri; saranno consentite a-z, A-Z, 0-9 e i simboli . _ - e ritorna un valore bolenano, TRUE se ha una sintassi corretta.
- **is_email()** verifica se la stringa ricevuta come parametro ha una sintassi corretta per un indirizzo email; ritorna un valore bolenano, TRUE se ha una sintassi corretta.

- **is_secure_password()** verifica se una determinata stringa può essere una password; al fine di semplificare il codice verrà semplicemente verificato che abbia una lunghezza di almeno 8 caratteri.
- **isset_username()** verifica se un determinato username risulta essere già registrato; ritorna un valore bolenano, TRUE se già registrato;
- **isset_email()** verifica se un determinato indirizzo email risulta essere già registrato; ritorna un valore bolenano, TRUE se già registrato;

Il controllo in sede di registrazione verranno cumulativamete eseguito per tutti i campi con il metodo **check_input_registrazione()** che ritornerà un valore boleano TRUE se gli input sono tutti correti. Analogamente avverrà in fase di login con il metodo **check_input_login()**.

Queste operazioni di validazione potranno essere effettuate anche lato client con javascript e ajax. Nel file da scaricare le validazioni sono state eseguire con jquery validation che è stato oggetto del precedente articolo in cui si spiegava come implementarlo in un form di registrazione del tutto identico a quello qui presentato.

Il login

In termini banali, l'utente inserendo le corrette credenziali diventerà loggato. Ci occorrerà anzitutto un fom per il login che sarà prodotto dal metodo **get_login_form()**.

L'esecuzione del login avverrà tramite il metodo **esegui_login()**.

Quest'ultimo quando il form verrà inviato cripta la password con il medoto **crypt_pass()** e verifica che i campi non siano vuoti con il metodo **check_input_login()**.

Se la validazione è andata a buon fine eseguiamo la query su database.

A questo punto, se la query individua l'utente occorrerà loggarlo. Ma cosa significa tecnicamente loggare un utente?

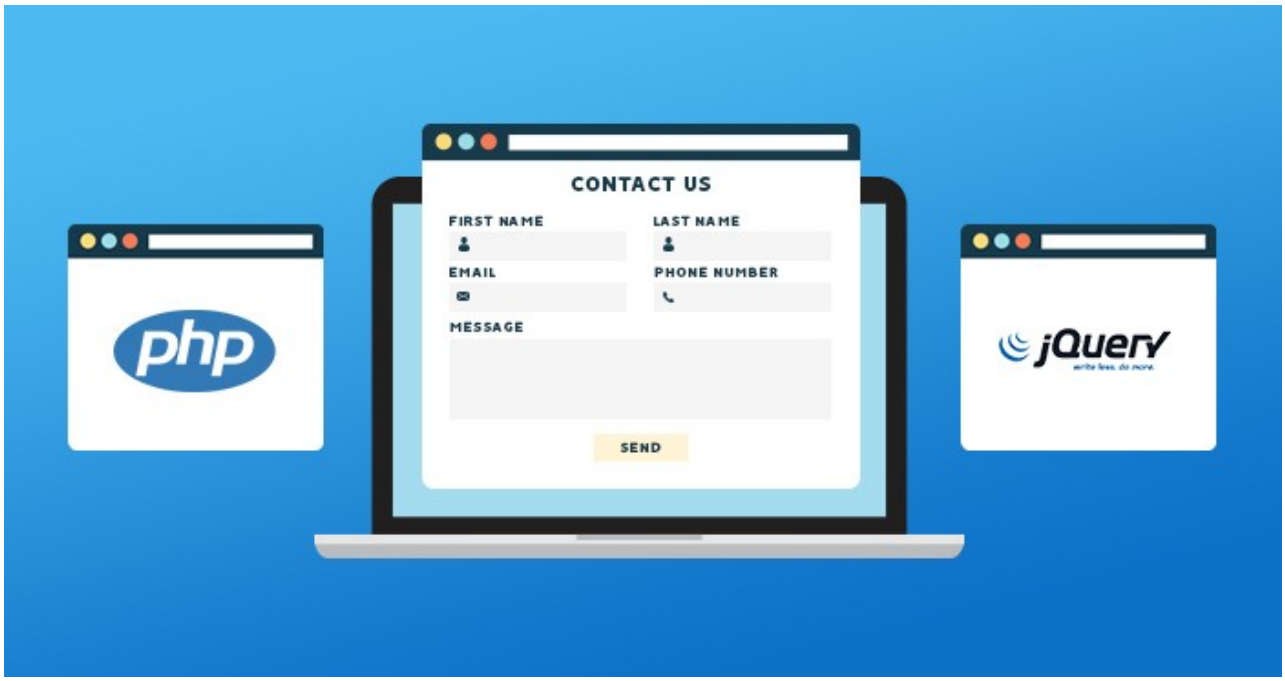
Dal punto di vista tecnico lo status di "*utente loggato*" è determinato dalla esistenza della variabile di sessione che attesta l'avvenuta autenticazione; nel nostro script tale variabile sarà **$_SESSION['auth']**.

Detto ciò ne deriva che:

- verificare che un utente è loggato significherà verificare l'esistenza della variabile $_SESSION['auth']; si veda a tale scopo il metodo **is_logged()**;
- loggare l'utente equivale a creargli la variabile di sessione $_SESSION['auth']; si veda a tale il metodo scopo **set_logged()**;

- proteggere una pagina web consiste nel vietare l'accesso a chi non "possiede" la variabile di sessione $_SESSION['auth']; si veda a tale scopo il metodo **access_denied()**;
- eseguire il logout significa cancellare le variabili di sessioni tra di esse $_SESSION['auth'] e il relativo cookie di sessione; si veda a tale scopo il metodo **logout()**;

How to Create a PHP Contact Form With MySQL & HTML5 Validation



A PHP contact form allows users to communicate with website administrators. It allows them to send queries to the site owners about relevant services or features. Using the contact form, web administrators are able to manage their business emails. Once there is an active contact form available, it can generate queries. It easily gets connected with the database, thus providing complete record and details about the users who are willing to contact and send their queries to website administrators.

Table of Content

9. Form Handler Library
10. Final Words

Prerequisites

To create a simple PHP contact form with MySQL, I assume that you have a PHP application installed on a web server. My setup is:

- PHP 7.1
- MySQL
- Jquery/Ajax

To make sure that that I don't get side-tracked by server level issues, I decided to host PHP application on Cloudways managed servers because the platform offers a powerful PHP optimized environment. In addition, I don't have to deal with server management hassles and thus focus on the core idea of this tutorial.

Create the Contact Form HTML

Create the contact form HTML as shown below with validation and save it with .php extension. Value which will be written between the double quotes in attribute Name like name="u_name" in input tags work as a variable name. These attributes will contain the data from the form that we will use to save in our database . There are two methods to send your form data to your PHP page: GET and POST. I will be using POST as it hides the user data and there is no limit to send data. If you don't have the time to dive deep in technicalities, you can use online forms that are pre-designed according to professional form design standards.

Note: For styling you can use your own **CSS** and also use **Bootstrap Classes** for better styling.

```
1. !DOCTYPE html>
2. <html>
3. <head>
4. <!-- Latest compiled and minified CSS -->
5. <link rel="stylesheet"
   href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
   integrity="sha384-
   BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4
   u" crossorigin="anonymous">
6. <style>
7. #loading-img{
8. display:none;
9. }
```

```
..response_msg{
margin-top:10px;
font-size:13px;
background:#E5D669;
color:#ffffff;
width:250px;
padding:3px;
display:none;
}
</style>
</head>
<body>
<div class="container">
<div class="row">
<div class="col-md-8">
<h1><img src="Inquiry.png" width="80px">Easy Contact Form With Ajax MySQL</h1>
<form name="contact-form" action="" method="post" id="contact-form">
<div class="form-group">
<label for="Name">Name</label>
<input type="text" class="form-control" name="your_name" placeholder="Name" required>
</div>
<div class="form-group">
<label for="exampleInputEmail1">Email address</label>
<input type="email" class="form-control" name="your_email" placeholder="Email" required>
</div>
<div class="form-group">
<label for="Phone">Phone</label>
<input type="text" class="form-control" name="your_phone" placeholder="Phone" required>
</div>
<div class="form-group">
<label for="comments">Comments</label>
<textarea name="comments" class="form-control" rows="3" cols="28" rows="5" placeholder="Comments"></textarea>
</div>
<button type="submit" class="btn btn-primary" name="submit" value="Submit" id="submit_form">Submit</button>
<img src="img/loading.gif" id="loading-img">
</form>
<div class="response_msg"></div>
</div>
</div>
</div>
```

```
<script
   src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
$("#contact-form").on("submit",function(e){
e.preventDefault();
if($("#contact-form [name='your_name']").val() === '')
{
$("#contact-form [name='your_name']").css("border","1px solid red");
}
else if ($("#contact-form [name='your_email']").val() === '')
{
$("#contact-form [name='your_email']").css("border","1px solid red");
}
else
{
$("#loading-img").css("display","block");
var sendData = $( this ).serialize();
$.ajax({
type: "POST",
url: "get_response.php",
data: sendData,
success: function(data){
$("#loading-img").css("display","none");
$(".response_msg").text(data);
$(".response_msg").slideDown().fadeOut(3000);
$("#contact-form").find("input[type=text], input[type=email], textarea").val("");
}
});
}
});
$("#contact-form input").blur(function(){
var checkValue = $(this).val();
if(checkValue != '')
{
$(this).css("border","1px solid #eeeeee");
}
});
});
</script>
</body>
</html>
```

Configure the MySQL Database

The next step is to setup and configure the MySQL database. For this, fire up the Cloudways Database manager and create a table '**contact_form_info',** with the fields **id** , **name , email , phone,comments**.

Next, create **config.php** that will be used to set up the connection between the PHP app and the database. Once the file has been created, open it and paste the following code in it:

```
<?php
$host = "localhost";
$userName = "fyrhp";
$password = "RTDE";
$dbName = "fyrhp";
// Create database connection
$conn = new mysqli($host, $userName, $password, $dbName);
// Check connection
if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error);
}
?>
```

Create the PHP Contact Form Script

Now let's create a file **get_response.php** and paste the following code in it:

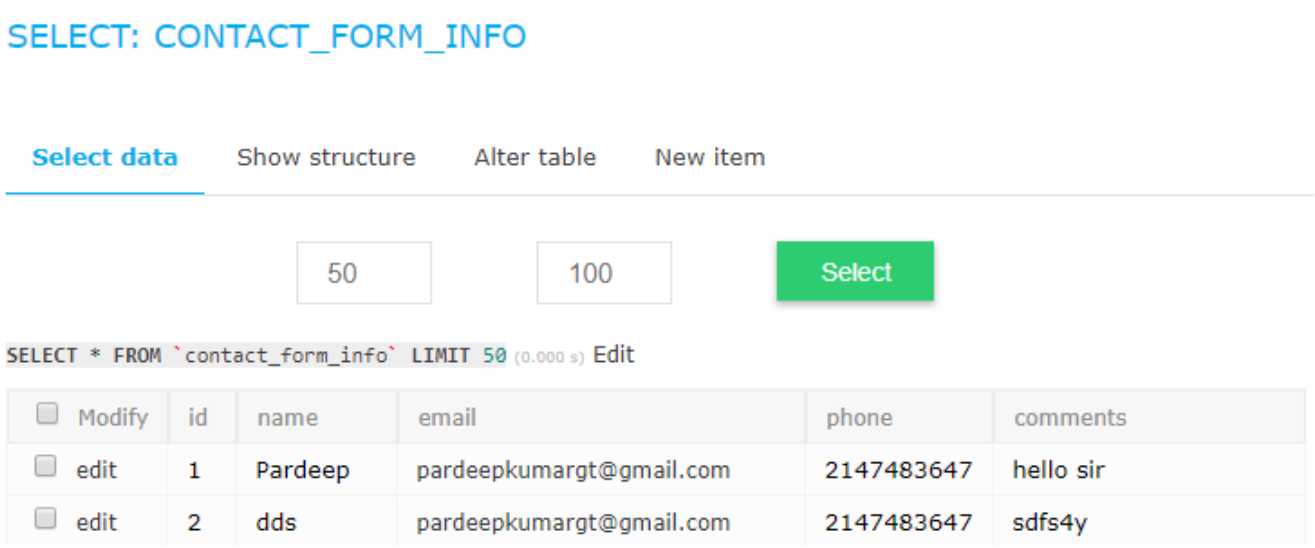
```
<?php
require_once("config.php");
if((isset($_POST['your_name'])&& $_POST['your_name'] !='') &&
(isset($_POST['your_email'])&& $_POST['your_email'] !=''))
{
require_once("contact_mail.php
```

```
$yourName = $conn->real_escape_string($_POST['your_name']);
$yourEmail = $conn->real_escape_string($_POST['your_email']);
$yourPhone = $conn->real_escape_string($_POST['your_phone']);
$comments = $conn->real_escape_string($_POST['comments']);
$sql="INSERT INTO contact_form_info (name, email, phone, comments) VALUES
('".$yourName."','".$yourEmail."', '".$yourPhone."', '".$comments."')";
if(!$result = $conn->query($sql)){
die('There was an error running the query [' . $conn->error . ']');
}
else
```

```
10.{
11.echo "Thank you! We will contact you soon";
12.}
13.}
14.else
15.{
16.echo "Please fill Name and Email";
17.}
18.?>
```

Since, I have used the POST method for submitting the form data to the server, I will use two global methods, $_REQUEST and $_POST to retrieve and save the data in the server local variable. The difference between these two is that $_REQUEST can retrieve data from both methods i.e. GET and POST. However, $_POST can only receive data from the POST method.

Here is what the PHP contact form script looks in action:



Mail Method

I also create a file **contact_mail.php** for mail in which send your contact form data on your mail easily.

```
1. <?php
2. $toEmail = "pardeepkumargt@gmail.com";
3. $mailHeaders = "From: " . $_POST["your_name"] . "<".
   $_POST["your_email"] .">\r\n";
4. if(mail($toEmail, $_POST["comments"], $_POST["your_phone"], $mailHeaders)) {
5. echo"<p class='success'>Contact Mail Sent.</p>";
6. } else {
```

```
7. echo"<p class='Error'>Problem in Sending Mail.</p>";
8. }
9. ?>
```

Form Captcha

You can use Captcha code in a form to ensure that the form is submitted with manual intervention without using any tools.

PHP Contact Form with Captcha

To develop contact form with captcha, let's start with the following HTML code. In this code, I will put a PHP file link into the image tag with the name of captcha.php.

```
1. <div>
2. <label>Captcha</label>
3. <span id="captcha-info" class="info"></span><br/>
4. <input type="text" name="captcha" id="captcha" class="demoInputBox"><br>
5. </div>
6. <div>
7. <img id="captcha_code" src="captcha.php" />
8. <button name="submit" class="btnRefresh" onClick="refreshCaptcha();">Refresh
   Captcha</button>
9. </div>
```

Contact Form Captcha Validation

For Captcha field validation, you can paste the following code in <script> tag. Using this code, you can set jQuery validation for captcha form validation.

```
1. if(!$("#captcha").val()) {
2. $("#captcha-info").html("(required)");
3. $("#captcha").css('background-color','#FFFFDF');
4. valid = false;
5. }
```

Captcha Refresh

This simple jQuery script will refresh PHP captcha code, and recreate the image with the new code. This new image will be set as captcha image source.

```
1. function refreshCaptcha() {
2. $("#captcha_code").attr('src','captcha.php');
3. }
```

PHP Captcha Image

PHP uses a function called PHP rand() to generate a random number. Using md5(), you can encrypt the number and split it into 6-character captcha code. The code is not only

added to the PHP session but also added as a source of captcha image using PHP GD function.

```
1. session_start();
2. $random_alpha = md5(rand());
3. $captcha_code = substr($random_alpha, 0, 6);
4. $_SESSION["captcha_code"] = $captcha_code;
5. $target_layer = imagecreatetruecolor(70,30);
6. $captcha_background = imagecolorallocate($target_layer, 255, 160, 119);
7. imagefill($target_layer,0,0,$captcha_background);
8. $captcha_text_color = imagecolorallocate($target_layer, 0, 0, 0);
9. imagestring($target_layer, 5, 5, 5, $captcha_code, $captcha_text_color);
10.header("Content-type: image/jpeg");
11.imagejpeg($target_layer);
```

Form Handler Library

You can also perform captcha and form validation by using Form Handler library. Check the following example:

```
1. use FormGuide\Handlx\FormHandler;
2. $demo = new FormHandler();
3. $validator = $demo->getValidator();
4. $validator->fields(['name','email'])->areRequired()->maxLength(50);
5. $validator->field('email')->isEmail();
6. $validator->field('message')->maxLength(6000)
7. $demo->requireCaptcha();
```

Final Words

In this tutorial, I demonstrated creating a PHP contact form using HTML, AJAX, jQuery and MySQL. Here is a live demo of the PHP contact form code in action. I've also given a brief overview about captcha form, and why you must use it in contact form. The article not only demonstrates how to setup captcha in a contact form, but also defines how to setup its validation using the script tag. Furthermore, you can also use Form Handler library to integrate captcha validation within the form.

Still, if you have some more questions regarding captcha integration in the contact form, or want to share some of your tips on the topic, feel free to write down your suggestions in the comments section below.

Come creare un form PHP per l'invio di dati

In questo caso il form invierà : Nome, Età, Genere, Messaggio

Come metodo in questo caso utilizziamo POST.

Ecco il codice HTML da scrivere per la pagina index.php

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

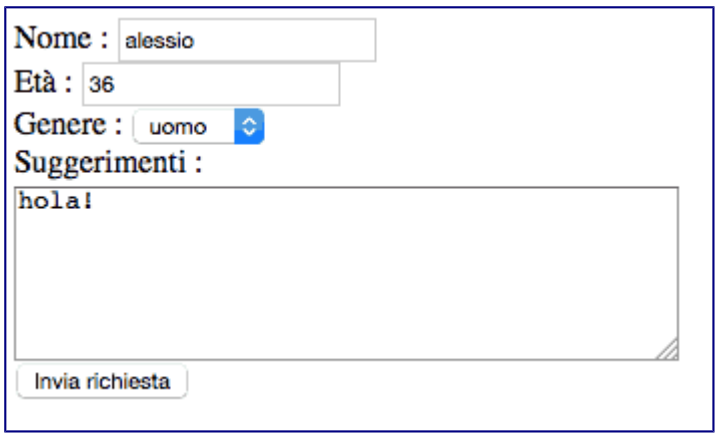
<head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<title>Invio i dati</title>

</head>

<body> <form action="ricevitore.php" method="post" name="datiUtenti"> Nome : <input type="text" name="nome" /></br> Età : <input type="text" name="eta" /></br> Genere : <select name="genere"> <option value="uomo" selected="selected">uomo</option> <option value="donna">donna</option> </option> </select> </br> Suggerimenti :</br> <textarea name="testo" cols="40" rows="5"></textarea></br> <input type="submit" /> </form> </body> </html>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Invio i dati</title>
</head>
<body>
<form action="ricevitore.php" method="post" name="datiUtenti">
Nome : <input type="text" name="nome" /></br>
Età : <input type="text" name="eta" /></br>
Genere : <select name="genere">
<option value="uomo" selected="selected">uomo</option>
<option value="donna">donna</option>
</option>
</select>
</br>
Suggerimenti :</br>
<textarea name="testo" cols="40" rows="5"></textarea></br>
<input type="submit" />
</form>
</body>
</html>
```



In realtà la pagina index.php l'ho chiamata in questo modo solo per convenzione, potete utilizzare un qualunque altro nome purchè abbia estensione .php .

Come si può notare ogni campo input è formito del tag <name> che verrà utilizzato successivamente come variabile da passare al file che riceve i dati, ossia quello che ho chiamato ricevitore.php, il tag <value> viene invece utilizzato per valorizzare una select (è il comportamento medesimo del tag name).

L'utilizzo del metodo POST mi permette di non visualizzare i dati nella URL, che altrimenti sarebbero visibili e non criptati in questo caso.

Premendo il pulsante submit (che come valore riporterà "invia richiesta") invierà i dati al file ricevitore, ossia la pagina ricevitore.php, andiamo quindi a creare la pagina.

Come creare una pagina PHP per la ricezione di dati di un form con delle variabili

Contenuto pagina ricevitore.php

PHP

<?php $var1= $_POST["nome"]; $var2= $_POST["eta"]; $var3= $_POST["genere"]; $var4= $_POST["testo"]; if ($var3 == "uomo"){ $articolo = "un" ; } else{ $articolo = "una"; } echo "Ciao $var1 hai $var2, sei $articolo $var3 e hai detto </br> $var4"; ?>

```php
<?php

$var1= $_POST["nome"];
$var2= $_POST["eta"];
$var3= $_POST["genere"];
$var4= $_POST["testo"];

if ($var3 == "uomo"){
    $articolo = "un" ;
    }
else{
    $articolo = "una";
    }
echo "Ciao $var1 hai $var2, sei $articolo $var3 e hai detto </br> $var4";


?>
```

Ciao alessio hai 36, sei un uomo e hai detto
hola!

Analizziamo cosa fa questo codice PHP di preciso.

$var* = il dollaro indica sempre una variabile

$_POST[“*”] = indica il valore contenuto nel tag <name> del file precedente, ossia la pagina (index.php nel nostro caso) che è quella che invia i dati

if = implica una condizione

else = è proprio la mera traduzione inglese di “altrimenti”

echo = viene utilizzato per stampare a video qualcosa, può contenere codice HTML, CSS e ovviamente anche delle variabili come in questo caso

In questa pagina in sostanza stiamo raccogliendo i dati del form e li stiamo inserendo in delle variabili, vedi $var1, $var2, gli stiamo però dicendo che le variabile variano (perdonate il gioco di parole), per ovvi motivi, perchè non possiamo sapere cosa verrò scritto nei campi del form.

La condizione if verifica semplicemente se il sesso è un uomo o una donna, nel primo caso inserirà un articolo maschile, nel secondo caso invece metterà quello femminile.

Questo guida è molto semplice ed intuitiva ed è consigliata per chi è al primo approccio con PHP, avendo iniziato anche io così, posso dirvi che una volta capito come funzionano le variabili poi vi potete sbizzarrire allegramente per inserire dati in un database o creare anche un piccolo motore di ricerca.

PHP: inserire dati su database tramite form, senza ricaricare la pagina

Ebbene sì, anche a me è capitato di dover creare una pagina in **PHP** con un form per salvare dei dati su un database dovendo evitare di cambiare pagina o ricaricare la stessa (via POST).

Cercando in giro per il web ho trovato un metodo piuttosto semplice e veloce da implementare: **utilizzare chiamate AJAX**

Come funziona? Tramite javascript si intercetta il clic sul bottone “Esegui” del form che abbiamo creato e si effettua una semplice chiamata “POST” alla pagina nella quale effettuiamo veramente l’ ” **insert** ” su database.

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

 //al click sul bottone del form
 $("#esegui").click(function(){

 //associo variabili
 var nome = $("#nome_cerca").val();  //  nome_cerca è l'ID della input del Form
 var cognome = $("#cognome_cerca").val();
 var sesso = $("#sesso_cerca").val();


 //chiamata ajax
 $.ajax({
```

```
//imposto il tipo di invio dati (GET O POST)
type: "POST",

//pagina da chiamare dove è presente la query
url: "./modifica.php",

//dati da inviare
data: "nome=" + nome + "&cognome=" + cognome + "&sesso=" + sesso,
dataType: "html",


success: function(msg)
{
$("#risultato-ricerca").html(msg); // aggiunti valori al db
},
error: function()
{
alert("Chiamata fallita, si prega di riprovare..."); //callback in caso di errore
}
});
});
});
</script>
```

Gestiamo quindi il risultato (valore di ritorno) della chiamata per mostra il buono od il cattivo esito della chiamata (“#risultato-ricerca”)

Ecco invece cosa inserire nella pagina modifica.php:

```
// catturiamo i dati in POST

$nome = $_POST['nome'];
$cognome = $_POST['cognome'];
$sesso = $_POST['sesso'];

// facciamo la query

$strsql="update paziente set nome = '$nome' , cognome = '$cognome'";
 $strsql.="where id ='$id_paz'";

 $result = mysql_db_query($database_quale,$strsql);
 if (!$result) {
 die('Invalid query: ' . mysql_error());}
 $numrec=mysql_affected_rows();

// gestiamo il ritorno  ( le classi si riferiscono a foundation 6 )
```

```
 if($numrec!= 0){
 //echo '<p>Paziente modificato correttamente</p>'; } //Messaggio che apparirà sotto il form di aggiunta tramite msg
 ?>
 <div class="success callout" data-closable>
 <strong>OK! Paziente <?php echo "$id_paz"; ?> modificato! </strong>
 <button class="close-button" aria-label="Dismiss alert" type="button" data-close>
 <span aria-hidden="true">&times;</span>
 </button>
 </div>
 <?php }
 else {

 ?>
 <div class="warning callout" data-closable>
 <strong><?php echo 'Si e verificato un errore ' . mysql_error() .''; ?></strong> P
 <button class="close-button" aria-label="Dismiss alert" type="button" data-close>
 <span aria-hidden="true">&times;</span>
 </button>
 </div>

 <?php

 }
```

Use PHP to Insert Information Into a MySQL/MariaDB Database From an HTML Form

Introduction

A PHP script is a convenient way to accept information submitted from a website's HTML form and insert it into a MySQL/MariaDB database. This tutorial will cover how to create a basic PHP script for inserting data, and an HTML form to take user input from a webpage and pass it to the PHP script.

Requirements

- A Cloud Server running Linux (any distribution)
- Apache, MySQL/MariaDB, and PHP installed and running.

Note: Apache, MySQL/MariaDB, and PHP are installed and running on a Standard installation by default. If your server was created with a Minimal installation, you will need to install and configure Apache, MySQL/MariaDB, and PHP before you proceed.

Create the MySQL/MariaDB Database and User

For this tutorial we will create a web page for an imaginary restaurant. The web page will allow customers to submit reviews of the restaurant.

Log in to the command line MySQL/MariaDB client:

```
mysql -u root -p
```

mixed

Create a database for the reviews:

```
CREATE DATABASE reviews;
```

mixed

Switch to that database:

```
USE reviews;
```

mixed

For this example, we will only create one table. It will have three fields:

- An ID field: This will be set to auto-increment.
- The reviewer's name: A text field with a 100-character limit.
- A star rating: A numeric rating of 1-5 TINYINT
- Review details: A text field with a limit of approximately 500 words. VARCHAR(4000)

Create the table:

```
CREATE TABLE user_review (
  id MEDIUMINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  reviewer_name CHAR(100),
  star_rating TINYINT,
  details VARCHAR(4000)
  );
```

mixed

For security reasons, it is always best to create a unique user for each database, particularly when that database will be accessed from a website.

The following command will create a user review_site with password JxSLRkdutW and grant the user access to the reviews database:

```
GRANT ALL ON reviews.* to review_site@localhost IDENTIFIED BY 'JxSLRkdutW';
```

mixed

Create the Web Page HTML Form

Create the file reviews.html in your webspace and open it for editing. For example, to create the file in /var/www/html the command is:

```
sudo nano /var/www/html/reviews.html
```

mixed

Put the following content into this file:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <body>

 <p>Let us know how we're doing!</p>

 <form action="addreview.php" method="POST">

    Your name: <input type="text" name="reviewer_name"><br><br>

    How many stars would you give us?
     <select name="star_rating">
      <option value="1">1 star</option>
      <option value="2">2 stars</option>
      <option value="3">3 stars</option>
      <option value="4">4 stars</option>
      <option value="5">5 stars</option>
     </select><br><br>

    Your review: <br>
     <textarea name="details" rows="10" cols="30"></textarea><br><br>

    <input type="submit" value="Submit" name="submit">

 </form>

</body>
</html>
```

mixed

A few things to note about this basic HTML form:

- This form uses the POST method to pass data to the addreview.php PHP script.
- The name for each input field will be used as the PHP variable name in the next step. It is usually best to use the database table field name for these values.
- Never trust user input. In this example, we require a number from 1 to 5 for the star rating. If we asked users to manually enter their star rating, inevitably people would enter the wrong thing. When applicable, it's best to ask users to choose from a drop-down menu with the correct values already populated.

Create the PHP Script

Create a file addreview.php in your webspace and open it for editing. For example, to create the file in /var/www/html the command is:

sudo nano /var/www/html/addreview.php

mixed

Every PHP script must begin with the PHP opening tag:

<?php

mixed

Next, add a MySQL/MariaDB connection block with the server location (localhost), the database name, and the database username and password.

```
$hostname = "localhost";
$username = "review_site";
$password = "JxSLRkdutW";
$db = "reviews";
```

mixed

Then we add a section to connect to the database, and give an error if the connection fails:

```
$dbconnect=mysqli_connect($hostname,$username,$password,$db);

if ($dbconnect->connect_error) {
  die("Database connection failed: " . $dbconnect->connect_error);
}
```

mixed

Next, we "translate" the data from the HTML form into PHP variables:

```
if(isset($_POST['submit'])) {
  $reviewer_name=$_POST['reviewer_name'];
  $star_rating=$_POST['star_rating'];
  $details=$_POST['details'];
```

mixed

Then insert this information into the database:

```
  $query = "INSERT INTO user_review (reviewer_name, star_rating, details)
  VALUES ('$reviewer_name', '$star_rating', '$details')";
```

mixed

Add a nested if statement which will give an error message if the process fails, or thank the user for their review if it succeeds:

```
    if (!mysqli_query($dbconnect, $query)) {
        die('An error occurred when submitting your review.');
    } else {
      echo "Thanks for your review.";
    }
```

mixed

And finally, close the opening if statement and add a PHP closing tag:

```
}
?>
```

mixed

Note: If you get the error message "Database connection failed: Access denied for user 'review_site'@'localhost' (using password: YES)" you can test the login information by logging in to MySQL/MariaDB from the command line with the command:

```
mysql -u review_site -p reviews
```

mixed

To test the script, visit reviews.html in a browser and submit a review. Then log in to the reviews database with the command line MySQL/MariaDB client:

```
mysql -u root -p reviews
```

mixed

Use SELECT * FROM user_review to view the entire contents of the table:

```
MariaDB [reviews]> SELECT * FROM user_review;
+----+--------------+-------------+-------------------------------------------------------+
| id | reviewer_name | star_rating | details                                               |
+----+--------------+-------------+-------------------------------------------------------+
|  1 | Ben          |           5 | Love the calzone!                                     |
|  2 | Leslie       |           1 | Calzones are the worst.                               |
+----+--------------+-------------+-------------------------------------------------------+
2 rows in set (0.00 sec)
```

mixed

The full PHP script is:

```
<?php

$hostname = "localhost";
```

```
$username = "review_site";
$password = "JxSLRkdutW";
$db = "reviews";

$dbconnect=mysqli_connect($hostname,$username,$password,$db);

if ($dbconnect->connect_error) {
  die("Database connection failed: " . $dbconnect->connect_error);
}

if(isset($_POST['submit'])) {
  $reviewer_name=$_POST['reviewer_name'];
  $star_rating=$_POST['star_rating'];
  $details=$_POST['details'];

  $query = "INSERT INTO user_review (reviewer_name, star_rating, details)
  VALUES ('$reviewer_name', '$star_rating', '$details')";

    if (!mysqli_query($dbconnect, $query)) {
        die('An error occurred. Your review has not been submitted.');
    } else {
      echo "Thanks for your review.";
    }

}
?>
```

2

I need to connect the form to a database that I already have, but it is not connecting and I can't seem to find the problem. I the error that is showing up mostly is the "mysqli" error but I can't find the reason behind it

--order.php--

```
<!DOCTYPE html>
<html>
    <head>
        <title>Checkout</title>
        <link rel = "stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"/>
        <link rel="stylesheet" href="styles.css"/>
    </head>
    <body>
        <div class="col-sm-5 clearfix">
            <div class="bill-to">
                <h2>Bill To</h2>
                    <div class="form-one">
```

```
                <form action="order.php" method="POST">
                <input type="text" id="FirstName" name="FirstName" placeholder="First
Name" style="width:350px; background-color:pink;"><br>
                <input type="text" id="Surname" name="Surname" placeholder="Last Name"
style="width:350px;background-color:pink;"><br>
                <input type="text" id="HouseName" name="HouseName"
placeholder="House Name" style="width:350px;background-color:pink;"><br>
                <input type="text" id="StreetName" name="StreetName" placeholder="Street
Name" style="width:350px;background-color:pink;"><br>
                <input type="text" id="Locality" name="Locality" placeholder="Locality"
style="width:350px;background-color:pink;"><br>
                <input type="text" id="MobileNumber" name="MobileNumber"
placeholder="Mobile Number" style="width:350px;background-color:pink;"><br>
                <input type="text" id="Email" name="Email" placeholder="Email Address"
style="width:350px;background-color:pink;">
                <input type="submit" name="submit" value="Register"/>
                </form>


                <?php

                   if(isset($_POST['submit']))
                   {
                   $FirstName=$_POST['FirstName'];
                   $Surname=$_POST['Surname'];

                   $result="INSERT INTO users (FirstName,Surname) VALUES
('$FirstName','$Surname')";
                    mysqli_query($conn,$result);
                   }
                ?>
               </div>
          </div>
       </div>

   </body>
</html>
```

--connect.php-- This is the code that is in connect.php

```
<?php
   $user = 'root';
   $pass ='';
   $host = 'localhost';
   $db = 'webassignment';
```

```
    $conn = new mysqli($host, $user, $pass, $db);

  if ($conn -> connect_error) {
    die("Connection failed: " . $conn->connect_error);
  }
?>
```

improve this question
asked May 19 '18 at 14:48

Bleach101
1011 silver badge33 bronze badges

- Could you also tell what is the error coming ? – Siddharth Chaudhery May 19 '18 at 14:50
- 2
  Did you include connect.php? You are open to SQL injections. Please add what the the "mysqli" error is. – user3783243 May 19 '18 at 14:56

0

First, run only the connect.php file to check if the connection can be established successfully.

```
if ($conn -> connect_error) {
    die("Connection failed: " . $conn->connect_error);
  }
else
    echo 'Success';
```

In order.php I can't find anywhere that you have called the connect.php file. So, the $conn variable is of no use there.

Call it using include or require_once function.

improve this answer
answered May 19 '18 at 15:53

NewBee
36222 silver badges1313 bronze badges
add a comment

0

edit code in **connect.php** to be like this

```
$conn = mysqli_connect($host, $user, $pass, $db);

if(mysql_connect_errno()){
echo "can't connect" . mysqli_connect_error();
}
```

and in **order.php** put this code under <?php

```
require_once 'connect.php';
```

improve this answer
edited May 19 '18 at 19:04
answered May 19 '18 at 18:59

ferrysyahrinal
4155 bronze badges
add a comment

0

I've found the problem in the order.php the code is

```
 <?php
                    $conn = new mysqli("localhost","root","','webassignment',3306);

                   if (!$conn)
                   {
                      die('Could not connect: ' . mysql_error());
                   }

                    if(isset($_POST['submit']))
                    {
                    $FirstName=$_POST['FirstName'];
                    $Surname=$_POST['Surname'];
                    $HouseName=$_POST['HouseName'];
                    $StreetName=$_POST['StreetName'];
                    $Locality=$_POST['Locality'];
                    $MobileNumber=$_POST['MobileNumber'];
                    $Email=$_POST['Email'];
                    }

                    $result="INSERT INTO users
(FirstName,Surname,HouseName,StreetName,Locality,MobileNumber,Email) VALUES
('$FirstName','$Surname',
'$HouseName','$StreetName','$Locality','$MobileNumber','$Email')";
```

```
    if($conn->query($result))
    {
        //echo "worked";
    }else{
        //echo "Error: " .$result ."<br>". $conn->error;
    }

?>
```

Form d’invio e inserimento dei dati

La prima volta che accederemo alla nostra pagina PHP la tabella degli utenti sarà vuota, pertanto la prima azione utile da prendere in considerazione è l’inserimento di un nuovo record. Il link “aggiungi nuovo” ricaricherà la stessa pagina PHP passando come querystring la variabile $azione, lo switch che la gestisce entrerà nel case “form” e chiamerà la funzione crea_form().

Lo scopo di questa funzione è creare un form HTML per poi inserire i dati nel nostro database. Quindi, di fatto, inseriremo il codice HTML in una variabile che verrà restituita dalla funzione. Nella nostra pagina, ogni volta che una funzione restituisce un output in formato HTML lo memorizzeremo in una variabile $output che stamperemo.

```
function form() {
    global $id;
    $out='';
    $out.='<form name="info" action="?" method="post">';
    $out.='<input type="hidden" id="id" name="id" value="'.$id.'">';
    $out.='<input type="hidden" id="azione" name="azione" value="salva">';
    $out.='<label for="nome">Nome</label><br /><input type="text" id="nome"
name="nome" required><br />';
    $out.='<label for="cognome">Cognome</label><br /><input type="text" id="cognome"
name="cognome" required><br />';
    $out.='<label for="email">Email</label><br /><input type="email" id="email"
name="email" required><br />';
    $out.='<input type="submit" class="btn btn-success" value="Salva">';
    $out.='</form>';
    return($out);
    }
```

Il form sarà modellato sulla base della tabella utenti, quindi prevedremo tre campi per l’inserimento di nome, cognome e email. Sfruttando HTML 5 potremmo completare

velocemente la fase dei controlli, inserendo l'attributo required all'interno dei campi di input che vogliamo rendere obbligatori.

Nel form, oltre ai campi da compilare, avremo anche due campi di servizio, ma li inseriremo come campi nascosti. Nel primo inseriremo l'azione, in modo che al submit del form venga innescato l'opportuno switch che ci dirotti sulla funzione di salvataggio. Nel secondo andremo invece a scrivere il valore di una variabile id.

Nel caso dell'inserimento questa variabile sarà valorizzata a 0; essa viene letta e settata all'inizio della pagina PHP, inizializzata a 0 e passata come variabile global alla funzione. Per sicurezza andremo a controllare il contenuto della variabile tramite le filter_var, nel caso specifico, faremo il *sanitize* dei numeri interi, ovvero il nostro $id sarà ripulito da tutto ciò che non è compatibile con il formato di un numero intero.

Inseriremo quindi il codice necessario nella parte superiore della pagina insieme alla lettura della variabile $azione:

```
$id = 0;
if(isset($_REQUEST['id'])) {$id =
filter_var($_REQUEST['id'],FILTER_SANITIZE_NUMBER_INT);}
```

Riepilogando: accedo alla pagina, essendo la tabella del database vuota l'unica azione utile è l'inserimento di un nuovo utente; attivo il link, ricarico la stessa pagina ma con querystring azione=form. A questo punto viene invocata crea_form() che stamperà il codice del form.

Una volta compilato il form, premendo sul pulsante submit spediremo i dati del form al server ricaricando ancora una volta la stessa pagina, ma questa volta con azione=salva. Il nostro switch provvederà a chiamare la funzione che si occuperà di scrivere i dati nel database:

```
function salva() {
	global $id, $connection;
	$nome = $_REQUEST["nome"];
	$cognome = $_REQUEST["cognome"];
	$email = $_REQUEST["email"];
	if($id==0){
		$sql='INSERT INTO utenti(nome, cognome, email) VALUES(:nome,
:cognome, :email)';
		$stmt = $connection->prepare($sql);
		$stmt->bindParam(':nome',$nome,PDO::PARAM_STR);
		$stmt->bindParam(':cognome',$cognome,PDO::PARAM_STR);
		$stmt->bindParam(':email',$email,PDO::PARAM_STR);
		$stmt->execute();
		}
	}
```

Il primo passaggio è semplice: si tratta di recuperare le informazioni necessarie, quindi l'id che arriverà dal form a 0 e la variabile contenente la connessione al database. Poi salveremo in tre variabili i dati provenienti dal form.

$id verrà usata anche per decidere se eseguire un INSERT o un UPDATE; se l'id è uguale a zero, il nostro caso, procederemo all'inserimento del un nuovo record. Prepariamo quindi la stringa SQL relativa all'inserimento con la definizione dei parametri: come da specifica PDO useremo parametri nominali, molto più comodi dei segnaposto anonimi di MySQLi.

Il passaggio successivo sarà il **bind dei parametri**, dove associare la variabile valorizzata in precedenza al segnaposto e specificare il tipo di dato della variabile stessa. Quest'operazione è raccomandata per evitare l'iniezione di codice malevolo ed è a tutti gli effetti uno dei maggiori punti di forza di PDO.

Arrivati a questo punto non resta che eseguire la query: stringa SQL e parametri verranno inviati al server separatamente e non sarà più possibile stampare la stringa "completa" a fini del debug.

Il record viene quindi inserito nel database e si passa all'esecuzione della funzione lista() che mostrerà l'elenco dei record presenti nella tabella utenti del database.

Form PHP: gestire i dati inseriti nei moduli del sito

Una delle operazioni più comuni di interazione tra un sito web ed i suoi utenti, consiste nella compilazione di **moduli** (o **form**). In questo articolo vedremo in che modo chiunque può implementare un **form con PHP** all'interno del proprio sito web per effettuare una serie di operazioni di vario tipo e natura come, ad esempio, l'invio di email o l'inserimento di dati in un database.

Introduzione: i moduli in HTML

Per chi non lo sapesse, un modulo è composto da un insieme di tag HTML il cui scopo è raccogliere gli *input* dell'utente: possiamo avere caselle di testo, strumenti di selezione, strumenti di scelta (alternativa e multipla), strumenti per l'upload di file, ecc. Attraverso tutti questi strumenti, l'utente inserisce dei dati ed esprime delle preferenze che potranno essere oggetto di archiviazione e/o di elaborazione da parte di uno script.

In questa sede non ritengo opportuno tornare su concetti propri del linguaggio di markup ed invito, pertanto, il lettore interessato a leggere questa lezione della nostra Guida ad HTML per una rinfrescata su <form> e sugli altri tag per la creazione di moduli.

Quello che a noi interessa, in questa sede, è di capire in che modo uno **script PHP** può **raccogliere ed elaborare i dati raccolti mediante un form**.

Trasmissione dei dati: i metodi GET e POST

I form possono inviare dati ad una pagina web mediante due diverse metodologie di trasporto che prendono il nome di **GET** e **POST**.

Il metodo utilizzato deve essere specificato dallo sviluppatore all'interno del codice HTML del form attraverso l'attributo *method* in questo modo:

```
<form method="post" action="mioscript.php">
...
...
...
</form>
```

Copia

Sulla base di quanto specificato nell'attributo *method* ("get" o "post") dovremo conformare il codice della pagina che andrà a processare la richiesta (ovvero il file specificato nell'attributo *action*).

Quando i dati vengono inviati col metodo GET, questi sono trasmessi sotto forma di *query-string*. In altre parole, quando il form verrà inviato dall'utente, il browser effettuerà un redirect sulla pagina specificata nel parametro *action* accodando - all'interno della URL - i valori dei diversi campi del modulo, in questo modo:

```
mioscript.php?campo1=valore1&campo2=valore2&campo3=valore3
```

Come potete vedere i valori sono trasmessi "in chiaro" alla pagina web ricevente sotto forma di coppie di parametri/valori, tra loro concatenate mediante il simbolo della "E commerciale" (&).

Viceversa, se il modulo viene inviato col metodo POST, tutti questi valori saranno trasmessi alla pagina web in modo "trasparente" per l'utente, nel senso che i parametri ed i rispettivi valori non saranno visibili nella URL della pagina, ma saranno inviati all'interno della "parte invisibile" della trasmissione HTTP che avviene tra il client ed il server.

La raccolta dei dati inviati tramite un modulo da parte di uno script PHP

Fino ad ora abbia fatto un semplice ripasso circa il funzionamento dei moduli HTML ed il modo in cui consentono di trasmettere dei dati ad una specifica pagina web (uno script). Vediamo ora in che modo PHP è in grado di raccogliere i dati che sono stati raccolti ed inviati tramite un form.

Per farlo PHP dispone di due variabili globali corrispondenti ai due metodi di invio visti poco sopra. Le due variabili in questione prendono il nome di **$_GET** e **$_POST**. Queste due variabili sono valorizzate in automatico da PHP e contengono, sotto forma di array, tutti i valori ricevuti tramite il metodo GET ed il metodo POST (per maggiori informazioni si faccia riferimento a questa lezione della nostra Guida PHP).

Il funzionamento di queste variabili è identico: essendo delle array, è possibile recuperare il valore di un dato parametro semplicemente specificandone il nome tra parentesi quadra, in questo modo:

```
// recupero il valore del parametro 'campo1' trasmesso mediante query-string (metodo GET)
$campo1 = $_GET['campo1'];

// recupero il valore del parametro 'campo1' trasmesso mediante il metodo POST
$campo1 = $_POST['campo1'];
```

Copia

In altre parole: il valore del campo, presente nel nostro modulo HTML, avente come nome "campo1" può essere recuperato da PHP sfruttando, appunto, le variabili globali $_GET o $_POST (a seconda del *method* specificato nel tag <form>).

Facciamo un semplicissimo esempio pratico. Supponiamo di avere un form che chiede all'utente di inserire il proprio indirizzo email:

```
<form method="post" action="mioscript.php">
  Inserisci il tuo indirizzo email<br>
  <input type="text" name="email"><br>
  <input type="submit" value="Invia Modulo">
</form>
```

Copia

Vediamo quindi il corrispondente script PHP che, al *submit* del form, raccoglie questa informazione e - molto banalmente - la stampa a video:

```
<?php
$email = $_POST['email'];
echo 'Il tuo indirizzo email è: ' . $email;
>
```

Copia

L'output prodotto sarà qualcosa di questo tipo:

Il tuo indirizzo email è: fake@email.com

Oltre alle variabili $_GET e $_POST, PHP offre anche una terza opzione: la variabile **$_REQUEST**. Questa variabile può essere utilizzata quando non si sa se il dato viene recapitato tramite il metodo GET o il metodo POST, essendo questa variabile una sorta di "mix" di entrambe:

```
// recupero il valore del parametro 'campo1' (sia GET che POST)
$campo1 = $_REQUEST['campo1'];
```

Copia

Si faccia attenzione però, in quanto questa variabile contiene - oltre ai valori trasmessi via GET e POST - anche i valori presenti nella variabile globale $_COOKIE. A titolo personale non ne consiglio l'utilizzo se non in casi di effettivo bisogno.

Creare un modulo di contatto in PHP

Una delle esigenze tipiche quando si crea un sito web consiste nella creazione di un **modulo di contatto** che consenta agli utenti di inviarci delle comunicazioni. Molto spesso mi viene chiesto "Come si crea un modulo di contatto in HTML?" La risposta è: "Non si può". HTML, infatti, è un linguaggio di markup - e non di scripting - e non offre pertanto la possibilità di gestire autonomamente questo tipo di esigenze.

E' in questa prima e comunissima esigenza che PHP rivela la sua potenza e versatilità nell'interazione coi moduli. Vediamo, quindi, come creare un semplicissimo modulo di contatto con PHP, partendo dal markup HTML:

```
<form method="post" action="formmail.php">
  Inserisci il tuo nome<br>
  <input type="text" name="nome"><br>
  Inserisci il tuo indirizzo email<br>
  <input type="text" name="email"><br>
  Scrivi un messaggio:<br>
  <textarea name="msg"></textarea><br>
  <input type="submit" value="Invia Modulo">
</form>
```

Copia

Vediamo ora il codice PHP del nostro file "formmail.php":

```
<?php
// Recupero i valori inseriti nel form
$nome = $_POST['nome'];
$email = $_POST['email'];
$msg = $_POST['msg'];

// compilo un messaggio combinando i dati recuperati dal form
$testo = "Nome: " . $nome . "\n"
      . "Email: " . $email . "\n"
      . "Messaggio:\n" . $msg;

// uso la funzione mail di PHP per inviare questi dati al mio indirizzo di posta
mail('mia@email.com', 'Messaggio dal mio sito web', $testo);

// Mostro un messaggio di conferma all'utente
echo 'Grazie per averci contattato!';
?>
```

Copia

Quello proposto è un esempio rudimentale che, tuttavia, assolve bene il compito di mostrare in che modo PHP riesca ad interagire coi moduli per generare, ad esempio, una email all'atto del loro submit. Per maggiori informazioni sulle caratteristiche e le potenzialità della funzione mail() di PHP vi invito a leggere questa lezione della nostra Guida a PHP.

Archiviazione dei dati all'interno di un database MySQL

Altra funzionalità tipica nella **gestione dei moduli con PHP**, consiste nell'archiviazione dei dati (inseriti dall'utente nei vari campi di un form) all'interno di una base dati. Per semplictà faremo riferimento al database MySQL essendo il più diffuso in combinazione con questo linguaggio di scripting lato server.

Ancora una volta partiamo dal markup HTML di un semplice modulo di raccolta dati anagrafici per la registrazione ad un dato servizio:

```
<form method="post" action="registrazione.php">
  Inserisci il tuo nome<br>
  <input type="text" name="nome"><br>
  Inserisci il tuo cognome<br>
  <input type="text" name="cognome"><br>
  Inserisci il tuo indirizzo email<br>
  <input type="text" name="email"><br>
  Scegli una password<br>
  <input type="text" name="pass"><br>
  <input type="submit" value="Registrati">
</form>
```

Vediamo ora il sorgente dello script che effettua materialmente la registrazione dell'utente salvandone i dati all'interno di un database:

```
<?php
// Recupero i valori inseriti nel form
$nome = $_POST['nome'];
$cognome = $_POST['cognome'];
$email = $_POST['email'];
$password = $_POST['pass'];

// dati di connessione al mio database MySQL
$db_host = 'localhost';
$db_user = '...';
$db_pass = '...';
$db_name = '...';

// connessione al DB utilizzando MySQLi
$cn = new mysqli($db_host, $db_user, $db_pass, $db_name);

// verifica su eventuali errori di connessione
if ($cn->connect_errno) {
    echo "Connessione fallita: ". $cn->connect_error . ".";
    exit();
}

// definisco la query di inserimento dati
$sql = "INSERT INTO utenti (nome, cognome, email, password) VALUES ("
     . "'" . $nome . "',"
     . "'" . $cognome . "',"
     . "'" . $email . "',"
```

```
    . "'" . $password . "')"

// esecuzione della query
if (!$cn->query($sql)) {
  echo "Errore della query: " . $cn->error . ".";
}else{
  echo "Registrazione effettuata correttamente.";
}

// chiusura della connessione
$cn->close();
```

Copia

Ancora una volta sembra tutto piuttosto semplice: PHP recupera i dati inseriti dall'utente all'interno del modulo, poi li utilizza all'interno di una query SQL che viene eseguita per effettuare l'inserimento dati all'interno del database (per maggiori informazioni sull'interazione tra PHP e MySQL consiglio la lettura di questa guida). Ancora una volta, però, quello proposto non è niente di più che un esempio scolastico la cui finalità non va oltre a quella puramente didattica: mettere in produzione questo script, infatti, sarebbe alquanto rischioso...

Il problema della sicurezza dei dati inviati coi moduli

Sino ad ora ci siamo limitati a mostrare in che modo è possibile gestire i form con PHP all'interno di "un mondo ideale" dove l'utente fa sempre e solamente quello che lo sviluppatore vorrebbe... peccato che nella realtà dei fatti non è sempre così.

Il buon sviluppatore, infatti, è ben cosciente dei rischi che si nascondono all'interno dei moduli e - più in generale - sa quanto sia importante effettuare un'accurata verifica e validazione di tutti i dati ricevuti in input dagli utenti.

Nell'esempio precedente, infatti, sarebbe piuttosto semplice, per un malintenzionato, attuare un attacco mediante la tecnica della SQL Injection: sarebbe sufficiente valorizzare *ad hoc* uno dei campi del nostro modulo di iscrizione, infatti, per compromettere la riservatezza e l'integrità del nostro database!

Quando si parla di **gestione dei form con PHP**, quindi, non si può non affrontare la tematica della **sicurezza dei moduli** al fine di prevenire attacchi di SQL Injection e Cross-Site Scripting (solo per citare le due minacce più diffuse).

A tal fine la raccomandazione è sempre la stessa: assicuratevi, prima di processare un modulo, che i dati inseriti al suo interno siano conformi ai valori che vi attendete da un utilizzatore corretto. Per fare un esempio: all'interno di un ipotetico campo "email" deve essere inserito un valido indirizzo email, così come all'interno di un ipotetico campo "età" deve essere scritto un valore di tipo numerico.

A tal fine PHP offre numerose funzioni *ad hoc*, vediamo di seguito un esempio di sequenza di controllo dei dati inseriti in un form:

```
// Recupero i valori inseriti nel form
$nome = $_POST['nome'];
$cognome = $_POST['cognome'];
$email = $_POST['email'];
$eta = $_POST['eta'];

// verifico che tutti i campi siano stati compilati
if (!$nome || !$cognome || !$email || !$eta) {
  echo 'Tutti i campi del modulo sono obbligatori!';
}
// verifico che il nome non contenga caratteri nocivi
elseif (!preg_match('/^[A-Za-z \'-]+$/i',$nome)) {
  echo 'Il nome contiene caratteri non ammessi';
}
// verifico che il cognome non contenga caratteri nocivi
elseif (!preg_match('/^[A-Za-z \'-]+$/i',$cognome)) {
  echo 'Il cognome contiene caratteri non ammessi';
}
// verifico se un indirizzo email è valido
elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  echo 'Indirizzo email non corretto';
}
// verifico che il campo età sia numerico, non sia inferiore di 1 e maggiore di 120
elseif (!is_numeric($eta) || $eta < 1 || $eta > 120) {
  echo 'Campo età non corretto';
}else{
  // A QUESTO PUNTO POSSO PROCESSARE IL FORM
  // ...
  // ...
  // ...
}
```

Altra utile precauzione consiste nell'utilizzo di funzioni di *escape dei caratteri speciali* prima dell'inserimento dei dati, ricevuti in input, all'interno di una query SQL: esempio tipico sono le funzioni *addslashes()* e, soprattutto, *mysql_real_escape_string()* e *mysqli_real_escape_string()*. In alternativa è possibile (e consigliabile) utilizzare la sintassi dei Prepared Statements di MySQLi che garantiscono una gestione sicura ed efficiente dei dati inseriti all'interno delle query SQL.

Altre funzioni molto utili quando si lavora coi form con PHP sono *htmlspecialchars()*, *htmlentities()* e *strip_tags()*: si tratta di tre funzioni native di PHP che è bene ricordare ed utilizzare nelle situazioni in cui è necessario prevenire attacchi di tipo XSS (Cross-Site Scripting) come, ad esempio, nei guestbook e nei forum o - più in generale - in tutte quelle situazioni in cui i dati passati in input da un utente vengono incorprati in una pagina visibile anche ad altri.

Conclusioni

PHP è un linguaggio estremamente potente e versatile grazie al quale non è particolarmente complicato **gestire i dati ricevuti in input dagli utenti tramite dei**

moduli HTML. Se la sintassi di base può apparire piuttosto semplice, tuttavia, è opportuno ricordarsi che una implementazione "basica" può prestare il fianco a numerosi inconvenienti tra cui, soprattutto, quello di cadere vittima di qualche attacco informatico.

Nella gestione dei form con PHP, quindi, è opportuno prendere le dovute precauzioni ed effettuare sempre i necessari controlli al fine di verificare che i dati inseriti siano conformi a quelli attesi.