Assignment Specification: JavaScript/Profiles

In this next assignment you will write a simple resume database that support Create, Read, Update, and Delete operations (CRUD). You will also move user information into its own table and link entries between two tables using foreign keys. You will also add some in-browser JavaScript data validation.

Sample solution

You can explore a sample solution for this problem at http://www.wa4e.com/solutions/res-profile/

Resources

There are several resources you might find useful:

Recorded lectures, sample code and chapters from www.wa4e.com:

- Review the SQL language
- Using PDO in PHP
- JavaScript

How to validate a form in JavaScript on StackOverflow.

Documentation from www.php.net on how to use PDO to talk to a database.

General Specifications

Here are some general specifications for this assignment:

- You must use the PHP PDO database layer for this assignment.
- Your name must be in the title tag of the HTML for all of the pages for this assignment.
- All data that comes from the users must be properly escaped using the **htmlentities()** function in PHP. You do not need to escape text that is generated by your program.
- You must follow the POST-Redirect-GET pattern for all POST requests. This means when your program receives and processes a POST request, it must not generate any HTML as the HTTP response to that request. It must use the "header('Location: ...');" function and either "return;" or "exit();" to send the location header and redirect the browser to the same or a different page.

- All error messages must be "flash-style" messages where the message is passed from a POST to a GET using the SESSION.
- Please do not use HTML5 in-browser data validation (i.e. type="number") for the fields in this assignment as we want to make sure you can properly do server side data validation. And in general, even when you do client-side data validation, you should still validate data on the server in case the user is using a non-HTML5 browser.

Databases and Tables Required for the Assignment

You will need to have a users table as follows:

```
CREATE TABLE users (

  user_id INTEGER NOT NULL AUTO_INCREMENT,

  name VARCHAR(128),

  email VARCHAR(128),

  password VARCHAR(128),

  PRIMARY KEY(user_id)

) ENGINE = InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE users ADD INDEX(email);

ALTER TABLE users ADD INDEX(password);
```

You will also need to add a Profile table as follows:

```
CREATE TABLE Profile (

  profile_id INTEGER NOT NULL AUTO_INCREMENT,

  user_id INTEGER NOT NULL,

  first_name TEXT,

  last_name TEXT,

  email TEXT,

  headline TEXT,

  summary TEXT,
  PRIMARY KEY(profile_id),

  CONSTRAINT profile_ibfk_2

  FOREIGN KEY (user_id)

  REFERENCES users (user_id)

  ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This table has a foreign key to the **users** table.

The Screens for This Assignment

We are going to have a number of screens (files) for this assignment. Functionality will be moved around from the previous assignment, although much of the code from the previous assignment can be adapted with some refactoring.

- **index.php** Will present a list of all profiles in the system with a link to a detailed view with **view.php** whether or not you are logged in. If you are not logged in, you will be given a link to **login.php**. If you are logged in you will see a link to **add.php** add a new resume and links to delete or edit any resumes that are owned by the logged in user.
- **login.php** will present the user the login screen with an email address and password to get the user to log in. If there is an error, redirect the user back to the login page with a message. If the login is successful, redirect the user back to **index.php** after setting up the session. In this assignment, you will need to store the user's hashed password in the **users** table as described below.
- **logout.php** will log the user out by clearing data in the session and redirecting back to **index.php**. This file can be very short - similar to the following:

```
<?php // line 1 added to enable color highlight

session_start();

unset($_SESSION['name']);

unset($_SESSION['user_id']);

header('Location: index.php');
```

- **add.php** add a new Profile entry. Make sure to mark the entry with the foreign key **user_id** of the currently logged in user. (create) `<?php // line 1 added to enable color highlight`

```
$stmt = $pdo->prepare('INSERT INTO Profile
  (user_id, first_name, last_name, email, headline, summary)

  VALUES ( :uid, :fn, :ln, :em, :he, :su)');

$stmt->execute(array(

  ':uid' => $_SESSION['user_id'],

  ':fn' => $_POST['first_name'],

  ':ln' => $_POST['last_name'],

  ':em' => $_POST['email'],

  ':he' => $_POST['headline'],

  ':su' => $_POST['summary'])

);
```

- **view.php** show the detail for a particular entry. This works even is the user is not logged in. (read)
- **edit.php** edit an exsiting entry in the database. Make sure the user is logged in, that the entry actually exists, and that the current logged in user owns the entry in the database. (update)
- **delete.php** delete an entry from the database. Do not do the delete in a GET - you must put up a verification screen and do the actual delete in a POST request, after which you redirect back to index.php with a success message. Before you do the delete, make sure the user is logged in, that the entry actually exists, and that the current logged in user owns the entry in the database. (delete)

If the user goes to an add, edit, or delete script without being logged in, die with a message of "Not logged in".

You might notice that there are several common operations across these files. You might want to build a set of utility functions to avoid copying and pasting the same code over and over across several files.

Storing Users and Hashed Password in the Database

In this assignment, we are going to allow for more than one user to log into our system so we will switch from storing the account and hashed password in PHP strings to storing them in the database. The salt value will remain in the PHP code.

Once you create the **users** table above, you will need to insert a single user record into the "users" table using this SQL:

INSERT INTO users (name,email,password)

```
VALUES ('UMSI','umsi@umich.edu','1a52e17fa899cf40fb04cfc42e6352f1');
```

The above password is the salted MD5 hash of 'php123' using a salt of 'XyZzy12*_'. You will need this user in the database to pass the assignment. You can add other users to the database is you like.

The salt value remains in the PHP code while the stored hash moves into the database. There should be **no stored hash** in your PHP code. You can compute the salted hash of any password / salt combination using this PHP code:

Salt-O-Matic 2000

Since the email address and salted hash are stored in the database, we must use a different approach than in the previous assignment to check to see if the email and password match using the following approach:

```
$check = hash('md5', $salt.$_POST['pass']);

$stmt = $pdo->prepare('SELECT user_id, name FROM users

WHERE email = :em AND password = :pw');

$stmt->execute(array( ':em' => $_POST['email'], ':pw' => $check));

$row = $stmt->fetch(PDO::FETCH_ASSOC);
```

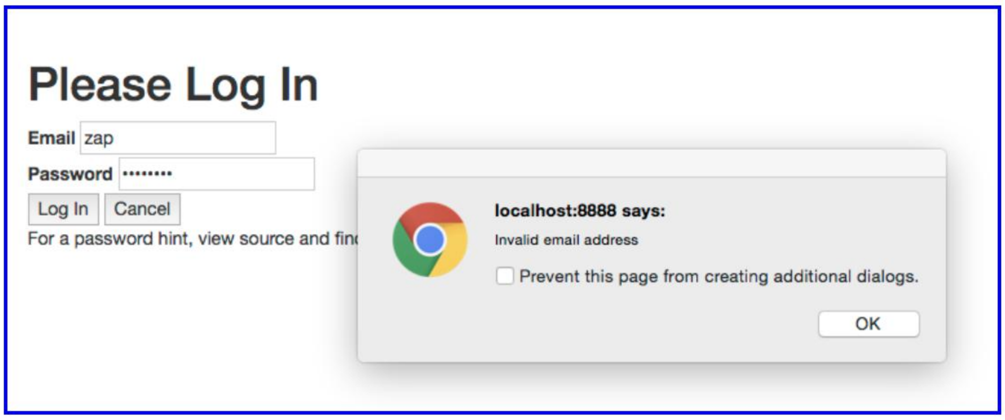
Since we are checking if the stored hashed password matches the hash computation of the user-provided password, If we get a row, then the password matches, if we don't get a row (i.e. $row is false) then the password did not match. If he password matches, put the user_id value for the user's row into session as well as the user's name:

```
if ( $row !== false ) {

$_SESSION['name'] = $row['name'];

$_SESSION['user_id'] = $row['user_id'];

// Redirect the browser to index.php

header("Location: index.php");

return;

...
```

Make sure to redirect back to **login.php** with an error message when there is no row selected.

Login Data Validation in JavaScript

In addition to the PHP data validation in the previous assignment, you need to add JavaScript based data validation on the **login.php** screen that pops up an alert() dialog if either field is blank or the email address is missing.


Please Log In

Email zap

Password ••••••••

For a password hint, view source and fin

localhost:8888 says:

Invalid email address

Prevent this page from creating additional dialogs.

This is done using an **onclick** event on the form submit button that calls a JavaScript function that checks the data, puts up an alert box if there is a problem and then returns true or false depending on the validity of the data.

```
...

<input type="password" name="pass" id="id_1723">

<input type="submit" onclick="return doValidate();" value="Log In">

...
```

This is a partial implementation of the doValidate() function that only checks the **password** field.

```
function doValidate() {

console.log('Validating...');

try {

pw = document.getElementById('id_1723').value;

console.log("Validating pw="+pw);
if (pw == null || pw == "") {

alert("Both fields must be filled out");

return false;

}

return true;

} catch(e) {

return false;

}

return false;

}
```

Make sure to retain the PHP data validation checks as well given that any in-browser checks can be bypassed by a determinied end-user.

Profile Data validation

When you are reading profile data in **add.php** or **edit.php**, do the following data validation:

- All fields are required. If one of the fields is left blank put out a message like

All fields are required

and redirect back to the same page.

- The email address must include an @ sign in the text. If there is no @ sign in the email address issue a message of the form:

Email address must contain @

and redirect back to the same page.

To redirect back to the same page when the page requires a GET parameter (i.e. like edit), you need to add the GET parameter to the URL that you put in the location header using a technique similar to the following:

```
header("Location: edit.php?profile_id=" . $_POST["profile_id"]);
```

You may need to change **profile_id** to match the GET parameter your code is expecting and the name of the hidden parameter in your form (and $_POST) data.

Submitting Your Assignment

As a reminder, your code must meet all the specifications (including the general specifications) above. Just having good screen shots is not enough - we will look at your code to see if you made coding errors. For this assignment you will hand in:

1. A screen shot (including the URL) of your index.php with the user logged in three resumes in the list.
2. A screen shot (including the URL) of your edit.php showing the error message for a bad email address
3. A screen shot (including the URL) of your delete.php showing that you are showing the confirmation screen before the delete is being done.
4. A screen shot of your Profile database table showing three rows
5. Source code of index.php
6. Source code of login.php
7. Source code of edit.php
8. Source code of delete.php

Optional Challenges

This section is entirely optional and is here in case you want to explore a bit more deeply and test your code skillz.

Here are some possible improvements:

- Add an optional URL field to your tables and user interface. Validate the URL to make sure it starts with "http://" or "https://". It is OK for this to be blank. If this is non-blank show the image in the table view in index.php and in the view.php file.
- Medium Difficulty: Use the PHP cURL library to do a GET to the image URL from within PHP and if the URL does not exist, issue an error message to the user and do not add the profile.
- This is a bit tricky so please don't try if it feels confusing. Change the program so it supports multiple users and each user can only edit or delete profiles that match their user_id. Insert a second row into the users table with the same or a diffferent hashed password. This way you can log in with one user name, add some profiles, logout and log in as another user, add some profiles and then logout and log back in as the original user and the Edit/Delete buttons will only appear for the profiles owned by the user.
- Advanced: Change the index.php so that it has a search field. Use the LIKE operator in the WHERE clause. You can use a LIKE operator on any column (including numbers) and you can use the LIKE column on all of the columns as well.

- Super Advanced: If there are more than 10 profiles, only show 10 at a time and put up Next and Back buttons as appropriate. Use count query to determine number of rows and a a LIMIT clause in your tables query to return the correct range of rows.

Database Setup Detail

If you have been doing all the previous assignments, you should have a database set up. Here is the SQL if you are just starting on this assignment:

```
CREATE DATABASE misc DEFAULT CHARACTER SET utf8 ;

GRANT ALL ON misc.* TO 'fred'@'localhost' IDENTIFIED BY 'zap';

GRANT ALL ON misc.* TO 'fred'@'127.0.0.1' IDENTIFIED BY 'zap';

USE misc; (If in the command line)

CREATE TABLE users (

user_id INTEGER NOT NULL AUTO_INCREMENT,

name VARCHAR(128),

email VARCHAR(128),

password VARCHAR(128),

PRIMARY KEY(user_id),

INDEX(email)

) ENGINE=InnoDB CHARSET=utf8;

ALTER TABLE users ADD INDEX(email);

ALTER TABLE users ADD INDEX(password);

INSERT INTO users (name,email,password)

VALUES ('Chuck','csev@umich.edu','1a52e17fa899cf40fb04cfc42e6352f1');

INSERT INTO users (name,email,password)

VALUES ('UMSI','umsi@umich.edu','1a52e17fa899cf40fb04cfc42e6352f1');
```