# C1- Code & Go

C-WEB-150

# Piscine PHP

Day 8

CODING ACADEMY
BY EPITECH

# Exercise 01

1 Pt

**File to hand in:** pool_php_d08/ex_01/ex_01.php

Write a function my_very_secure_hash that takes as parameter a password in plain-text and returns it **hashed** in MD5.

The algorith used is MD5

# Exercise 02

3 Pts

**File to hand in:** pool_php_d08/ex_02/ex_02.php

The issue with basic cryptography is that if two input are identical then, the output will also be identical. Let's imagine that your database becomes compromised, in your database you have 2 users: "Didier" and "Wladimir", they both share the same password "potiron", but Wladimir also got his password compromised because of a key logger he got while surfing on some dark site.

Because the hash for Didier's and Wladimir's password will be identical it will be very easy for the people who got access to your database to now guess Didier's password. Now, instead of one stolen password, you actually got two. If Didier didn't follow basic secure good practice (like Wladimir) and also used this password for his mailbox, his mailbox also became compromised, and so on…

To counter that, we use something called "salt", it is something randomly generated that will be added to the password before being hashed/encrypted, thanks to that all output will be different, even if the input was identical. That's it for the introduction of this exercise, don't forget to applaud our 2 friends that volunteered for this introduction.

Write a function my_password_hash that takes as parameter a password in plain-text and returns an array containing the hashed password in MD5 as well as the "salt" used for hashing. The salt must be different on each call.

Then write a function my_password_verify that takes as parameter a password in plain-text, a salt, and a hashed password. This function must return true or false, depending on whether the hashed password corresponds to the plain-text password with the associated salt or not.

**Protype:** array my_password_hash(string $password)
**Protype:** bool my_password_verify(string $password, string $salt, string $hash)

The array return by the function my_password_hash should look like this:

```
$array = [
    "hash" => "XXXXX",
    "salt" => "12345"
];
```

# Exercise 03

3Pts

**File to hand in:** pool_php_d08/ex_03/db_pdo.php

**Restrictions:** You need to manage PDO exceptions. You will display the error content in the following way: "PDO ERROR: <$error> storage in <error_log_file>\n". <error_log_file> is a constant called ERROR_LOG_FILE which will represent the path to your log file (it can be whatever you want). You must write this error in the <error_log_file>.

Create a function that enables a connection to a MySQL database and returns the connection resource.

**Prototype:** mixed connect_db($host, $username, $passwd, $port, $db);

**Prototype:** mixed connect_db($host, $username, $passwd, $port, $db);

# Exercise 04

1Pt

**File to hand in:** pool_php_d08/ex_04/connect_db.php

Using the function of the previous exercise (you must copy your previous function in this file), create a script that retrieves the connection information in command line.

You must define the constant ERROR_LOG_FILE to "errors.log".

**Execution of the script:** php connect_db.php host username password port db

In case the connection is successful, your script will display on the standard output "Connection to DB successful" followed by a new line.

In case of failure, it will display the error message "Error connection to DB" followed by a new line.

In case of missing parameters, it displays "Bad params! Usage: php connect_db.php host username password port db" followed by a new line.

All errors must be displayed on the standard output and in the ERROR_LOG_FILE.

# Exercise 05

2Pts

**File to hand in:** pool_php_d08/ex_05/my_print_users.php

Write a function my_print_users that takes as parameter a PDO instance and a variable number of integers.

Each integer corresponds to an identifier "id" in the "users" table of the database given to you.

The function must display the user name ("name" field) which corresponds to each id passed as parameter followed by "\n".

The function must return true if some results have been found and false otherwise.

If a given parameter is not an integer, the function must raise the exception "Invalid id given".

**Prototype:** bool my_print_users (PDO $bdd [, int $id, ...])

# Exercise 06

1Pt

**File to hand in:** pool_php_d08/ex_06/my_password_change.php

Write a function my_password_change that takes as parameter a PDO instance, an email and a password, and modifies the user associated with the email (''email'' field) in the database by changing his password (''password'' field). You will have to use the function password_hash of PHP and let it manage the salt with the default algorithm. The password recorded in the database has to be the result of hashing and not the password in plain-text.

The password will be checked with the function PHP 5.5 password_verify();

Your function must raise an exception if the password passed as parameter is empty or if the user does not exist. Use Exception() without message.

**Prototype:** void my_password_change(PDO $bdd, string $login, string $new_password)

# Exercise 07

2Pts

**File to hand in:** pool_php_d08/ex_07/my_change_user.php
**Restriction:** Use the keyword ''finally''

Write a function my_change_user taking as parameter a PDO instance and a variable number of usernames (''name'' field) in the ''users'' table.

For each name passed as parameter, modify the corresponding name in the database so that it follows the same logic as the following example:

''toto'' => ''Toto42'' (first letter in uppercase, the rest in lowercase, and 42 added at the end).
If a user passed as parameter does not exist in the database, you must raise a PDOException with the message ''User not found''.

The function must return an array containing the modified names sorted by number of characters, and then by alphabetical order.

In all cases, the function must display ''Good luck with the user DB!'' followed by ''\n''.

**Prototype:** array my_change_user (PDO $bdd [, int $id, ...]) **Example:**

```
// With array("eeee", "abc", "aaa", "dddd");
// Result :

array(4) {
   [0]=>
   string(3) "aaa"
   [1]=>
   string(3) "abc"
   [2]=>
   string(4) "dddd"
   [3]=>
   string(4) "eeee"
}
```

# Exercise 08

2Pts

**File to hand in:** pool_php_d08/ex_08/my_show_db.php
**Restriction:** Use the keyword "yield"

Write a function my_show_db that takes as parameter a PDO instance and a string, and which returns each entry in the table whose name corresponds to the string passed as parameter. The function returns NULL when there are no entries or if an error occurred.

**Prototype:** array my_show_db (PDO $bdd , string $table)

**Example:**

```
$generator = my_show_db($bdd, "users");
foreach ($generator as $line) {
        echo $line;
}
```

The output will have to respect the following format:

```
[id]=>[1], [name]=>[toto], [password]=>[mypassword], [email]=>[toto@toto.com],
    [created_at]=>[11/22/2015], [is_admin]=>[1]
[id]=>[2], [name]=>[tata], [password]=>[mypassword], [email]=>[tata@tata.com],
    [created_at]=>[11/22/2015], [is_admin]=>[1]
```

# Exercise 09

3Pts

**File to hand in:** pool_php_d08/ex_09/add_user.php

Using your function created in exercise 03, create a script that lets you add a user to the 'users' table The table 'users' contains 5 fields:

- 'id' auto incrementing primary key
- 'login' VARCHAR 20
- 'password' VARCHAR 255
- 'is_admin' BOOLEAN
- 'is_active' TINYINT 4 with a default value of 1

The SQL file needed to create this table is given to you with this subject. To import it, log in your MySQL console and type:

```
Terminal
~/C-WEB-150> mysql -u root -p
mysql> source <path_to_your_users_sql_file>
```

**Execution of the script:** php add_user.php login password password_conf role

In case the connection is successful, your script will display on the standard output "Connection to DB successful" followed by a new line.

In case of failure, it will display the error message from exercise 03 followed by "Error connection to DB" followed by a new line then stop.

You must define the constant ERROR_LOG_FILE to "errors.log".

In case of missing parameters, it displays "Bad params! Usage: php add_user.php login password password_conf role" followed by a new line.

All errors must be displayed on the standard output and in the ERROR_LOG_FILE.

The parameters of connection to the database will be defined as constant DB_HOST DB_USERNAME DB_PASSWORD DB_PORT DB_NAME in the moulinette, don't forget to remove them after your tests.

If the two passwords do not match, your script will display: "Passwords don't match" followed by a new line.

The role will have to be one of the following: ADM – GLOBAL – INVITE.

If the role is incorrect, your script will display "Bad role: ADM|GLOBAL|INVITE" followed by a new line.

The role is case-insensitive, but be careful: it will be stored in uppercase in the database.

If at least one error occurs, your script will stop after displaying all the errors.

If the user is added in the database, your script will display on the standard output "User added to DB" followed by a new line.

Otherwise, your script displays "Error MySQL, user not added, more infos in <path_to_log_file>" followed by a new line. The MySQL error will be stored in the same way as it was in exercise 03.
The password will be stored in sha256.

# Exercise 10

3Pts

Create a script that displays the content of the 'users' table.
**File to hand in:** pool_php_d08/ex_10/dump_users.php

**Execution:** php dump_users.php filter value extact

In case the connection is successful, your script will display on the standard output "Connection to DB successful" followed by a new line.

In case of failure, it will display "Error connection to DB" followed by a new line, then the script should stop.

The connection parameters to the database will be defined as constant DB_HOST DB_USERNAME DB_PASSWORD DB_PORT DB_NAME.

Parameters:

- filter: name of the column to filter (all except password)
- value: the value to search
- exact: exact value or not (true or false). If false, LIKE will be %value%

If no argument is passed as parameter, the script should displays the whole table on the standard output.

If at least one argument is passed and the total number of arguments is incorrect, the script will display "Bad params! Usage: php dump_users.php [filter value exact]" followed by a new line, then the script should stop.

If the filter is password, the script displays on the error output: "Don't try to filter the password, it's not possible" followed by a new line.

If no element matches the search, the script should display on the standard output: "No results matching your search" followed by a new line.

Display:

" - [<id>] <login> <role> (<active|inactive>)" followed by a new line (for each user who match with the filters)

In case of error, your script will display "Error MySQL, more infos in <path_to_log_file>" followed by a new line. The MySQL errors will be stored in the same manner as in the third exercise.

# Exercise bonus

5Pts

**File to hand in:** pool_php_d08/ex_11/user_manager_class.php
**Restriction:** You cannot use: exit, die;

You will be creating a class UsersManager in which you will be able to manipulate the ''users'' table.

This class will enable you to connect to the database.

The users will need to get authenticated before being able to do anything. If the connected user has the role ADM, he will be able to add/modify/delete users.

The GLOBAL role will allow a user to list other users and toggle them between active or inactive. The INVITE role will allow a user to see and modify his own information.

If a user can't connect (wrong login or password), the script will display ''Invalid credentials <nb_times_remaining> attemp(s) remaining'' followed by a carriage return.

After 5 failed connection attempts the account corresponding to the submitted login becomes inactive and the following message followed by a carriage return is displayed: ''For security reasons the account has been blocked''.

During a connection attempt, if the account has been blocked (is_active = 0) then the following message followed by a carriage return is displayed: ''For security reasons your account has been blocked, please contact your manager to unblock your account''.

The script will be run as follow:

$manager = new UsersManager(''localhost'', ''dbuser'', ''dbpass'', ''dbport'', ''dbname'');
$manager->connect()->start();
The parameters for the database connection will be given as parameters of the constructor in the following order: ''host'', ''dbuser'', ''dbpass'' ''dbport'', ''dbname''. Your class will need to contain a method which displays the following prompt: ''$<login>@usermanager> ''

If the user is not connected, you will display two prompts: "Login as: " followed by "Password: ".
Once the user is connected, you will display: "Welcome <login> you're now logged as <role>" followed by a carriage return.
Depending on the rights, the users will have access to:

- adduser <login> <password> <password_conf> <role>
  -> This allows you to add a user. The verifications are the same as the one in ex_09

- modifyuser modifyuser <id> login|password|role <newvalue> [<conf_value>]
  -> This allows the modification of the user possessing the id specified. The value of the field <key> will be updated with <newvalue>. If you are modifying the password, the field <conf_value> is required.
  -> If there is an error for the password, you should display the same message as the one for the addition of a user.
  -> If the user doesn't exist, you should display: "\t User #<id> unknown!\n".

- dump [<key> <value> <exact>]
  -> Equivalent to exercise 10.

- deluser <id>
  -> Deletes a user. The following confirmation prompt is displayed: "Are you sure ? [y/N]: ".
  -> If the user responds Y or y the user specified by id is deleted.
  -> If the user doesn't exist, you should display: "\t User #<id> unknown!\n".

- makeactive <id>
  -> Makes the user active.
  -> If the user doesn't exist, display: "\t User #<id> unknown!\n".

- makeinactive <id>
  -> renders the user inactive.
  -> If the user doesn't exist, display: "\t User #<id> unknown!\n".

- help [<command>]
  -> If no arguments are given, display the list of commands as follows: "\t- <command>\n".
  -> If a command is specified, display the command's usage.
  -> If the command specified doesn't correspond to any existing commands, display the same thing as if no arguments were given.

- logout
  -> Disconnects the current user and displays the login prompt.

- quit
  -> Quits the program after having displayed the same prompt as for the suppression.

If a command isn't available for the current user, display: "'<command>': Command unknown" followed by a carriage return.

If the connect method isn't able to connect to the DataBase, the same error message as the one in exercise 03 is displayed and the program is closed.

If the parameters are wrong, you should display: "\t Bad params. <usage>\n"

The PDO errors will be handled the same way they were in exercise 03.