

# DB Practical Work 1:

## The User model

December 13, 2016

### Abstract

The following subject aims at implementing the data handling for users in a twitter-like web-application. Implementations are to be done in the file `model/user.php`

## Contents

<b>1</b>	<b>Requirement</b>	<b>3</b>
<b>2</b>	<b>Work to do</b>	<b>3</b>
<b>3</b>	<b>The User entity</b>	<b>3</b>
3.1	Presentation . . . . .	3
3.2	hash_password(\$password) . . . . .	4
3.3	create(\$username, \$name, \$password, \$email, \$avatar_path)	4
3.4	get(\$id) . . . . .	4
3.5	modify(\$uid, \$username, \$name, \$email) . . . . .	4
3.6	change_password(\$uid, \$new_password) . . . . .	4
3.7	change_avatar(\$uid, \$avatar_path) . . . . .	5
3.8	destroy(\$id) . . . . .	5
3.9	search(\$string) . . . . .	5
3.10	list_all() . . . . .	5
3.11	get_by_username(\$username) . . . . .	5
3.12	check_auth(\$username, \$password) . . . . .	5
3.13	check_auth_id(\$id, \$password) . . . . .	5
<b>4</b>	<b>Following users</b>	<b>5</b>
4.1	Presentation . . . . .	5
4.2	follow(\$id, \$id_to_follow) . . . . .	6
4.3	unfollow(\$id, \$id_to_unfollow) . . . . .	6
4.4	get_followers(\$uid) . . . . .	6

4.5	<code>get_followings(\$uid)</code>	6
-----	------------------------------------	---

# 1 Requirement

To fulfill this work, you will need the following elements:

- A working environment with db connection to both app and test databases (see 0setup.pdf).
- On the two databases, at least the tables modeling **user** and **user** followings.

# 2 Work to do

You have to fill out the functions defined in the file `model/user.php`

These functions are used in the application to get access to the database. Therefore, these functions must observe some rules about both input data (the formats of the parameters of the functions) and output data (the returned values).

In the functions, you can access to the PDO object by using the following instruction:

---

```
$db = \Db::dbc();
```

---

Then, you can perform queries using `$db` like a PDO object:

---

```
$db = \Db::dbc();  
$result = $db->query('SELECT * FROM user');
```

---

When you completed all the functions, you can check them by using the unit tests available. In a command line window (at the root of the project), type in the following command:

---

```
vendor\bin\phpunit --bootstrap autoload.php tests\user.php
```

---

# 3 The User entity

## 3.1 Presentation

The User entity represents a user and its properties:

- its login username (used in URLs and during identification)
- its displayed name (which is a name to be displayed in the application)
- its hashed password (for identification)
- its email
- its avatar (or profile picture)

### 3.2 hash\_password(\$password)

This function takes a clear password as a parameter and returns a hashed version of it.

### 3.3 create(\$username, \$name, \$password, \$email, \$avatar\_path)

This function inserts a user in database. *It is to be noted that the password must be hashed (using `hash_password($password)`)*

The function returns either the id of the newly inserted user. If there was a problem during the insertion, The `null` value is returned.

It doesn't check whether the username is already taken or not.

### 3.4 get(\$id)

This function gets a post with a given id (the one given in parameter). Returns `null` if no user with the given id were found.

The application asks for a particular output: `get($id)` must return a `stdClass` PHP object. Such an object can be declared as follows:

---

```
$o = (object) array(  
    "attribute" => "value"  
);
```

---

In the case of our User entity, an object will be owning the following attributes:

---

```
$o = (object) array(  
    "id" => 1337,  
    "username" => "yrlgtm",  
    "name" => "User 1",  
    "password" => "hashed",  
    "email" => "yrlgtm@gmail.com",  
    "avatar" => "images/sddfvjdfvj.png"  
);
```

---

### 3.5 modify(\$uid, \$username, \$name, \$email)

This function updates a user whose id is `$uid`. It doesn't check whether the new username is already taken or not. It returns a boolean which gives the state of the query.

### 3.6 change\_password(\$uid, \$new\_password)

This function updates only a user's password. This function hashes the new password. It returns a boolean.

### 3.7 `change_avatar($uid, $avatar_path)`

This function changes the avatar of the user. The provided path is a temporary path and therefore the file has to be saved somewhere else. Returns a boolean.

### 3.8 `destroy($id)`

This function deletes a user entry. It returns a boolean.

### 3.9 `search($string)`

This function searches for users by query on both username and displayed name.

### 3.10 `list_all()`

This function returns an array of every users objects (same return format as in `get($id)`).

### 3.11 `get_by_username($username)`

This function returns a user matching the given username (same return format as in `get($id)`). Returns `null` if no user were found.

### 3.12 `check_auth($username, $password)`

Tries to authenticate a username with a given password. Returns the user object (same return format as in `get($id)`) if everything went fine. Returns `null` else.  
*This function **does** need to hash the password*

### 3.13 `check_auth_id($id, $password)`

Tries to authenticate a user id with a given password. Returns the user object (same return format as in `get($id)`) if everything went fine. Returns `null` else.  
*This function **doesn't** need to hash the password*

## 4 Following users

### 4.1 Presentation

Users can follow each others. Following someone enables a user to receive in their timeline the other user's posts.

- A user's followers are the users following him/her
- A user's following are the users he/she follows

**4.2** `follow($id, $id_to_follow)`

This function creates a "follow" association between two users. Returns a boolean.

**4.3** `unfollow($id, $id_to_unfollow)`

This function deletes a "follow" association between two users. Returns a boolean.

**4.4** `get_followers($uid)`

This function returns an array of objects for every users that follow a given user.

**4.5** `get_followings($uid)`

This function returns an array of objects for every users that a given user follows.