DB Practical Work 1:
The User model

November 23, 2017

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_student/user.php`

Contents

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 **followings**.

2 Work to do

You have to fill out the functions defined in the file `model_student/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 Creation, getting, password management and authentication

The following functions have to be coded before running the unit tests. They are in charge of handling both the creation and the fetching of the user objects.

3.2.1 `hash_password($password)`

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

3.2.2 `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.

3.2.3 `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"
);
```

3.2.4 `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.

3.2.5 `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*

3.2.6 `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($id)`) if everything went fine. Returns `null` else. *This function **doesn't** need to hash the password*, because `$password` is already given in its hashed form.

3.3 User modification and deletion

After the user objects being created, its attributes can be modified. The following functions are in charge of this and should be coded before running the unit tests.

3.3.1 `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.

3.3.2 `change_password($uid, $new_password)`

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

3.3.3 `change_avatar($uid, $avatar_path)`

This function changes the avatar of the user.

3.3.4 `destroy($id)`

This function deletes a user entry.

Important!
The deletion behaviour must be allowing deleting user which is following a user or followed by a user. **To see and modify the deletion policy in phpMyAdmin, you must go in the table view → Structure → Relation view.**

3.4 Searching and listing

The user objects can be searched for or listed. The two following functions can be coded separately.

3.4.1 `search($string)`

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

3.4.2 `list_all()`

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

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

Every functions relative to following must all be coded before running the unit tests.

4.2 `follow($id, $id_to_follow)`

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

4.3 `unfollow($id, $id_to_unfollow)`

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

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.