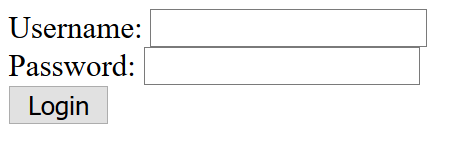
# PHP in Web Environment - Exercises

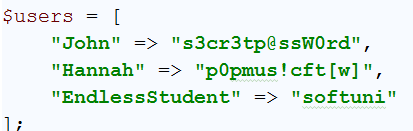
PHP’s http context is the fundamental part of writing web applications with PHP. A web application at its core is nothing more than sending data between requests and persisting it for further uses. That’s why it’s so important to understand concepts like **POSTi**ng, **GET**ting data and saving it in **SESSION**s and **COOKIE**s.

## Login Page

Define an HTML form for user login. It should consist of input fields for username, password and a submit button. Preferably called “login\_frontend.php” and posting to “login.php”.



Let’s create in the backend (login.php) an in-memory database of username/password pairs with predefined credentials. Associative array will fit the needs.

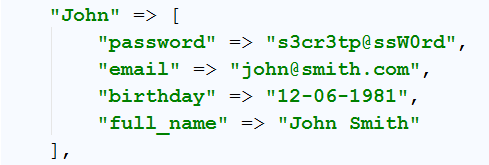


Now we need to check whether the sent username and password match one of the credential pairs. If true, then save the current username in the session and permit them to the next pages e.g. to profile.php.

Keep in mind that any unauthorized user can try to access profile.php. If the user is not authenticated, then you need to redirect them back to login.php.

## Profile Page – Greeting by Name

Once the user is logged in, it’s redirected to the profile page. Let’s extend our mini database in login.php. It needs to have some more information about each user. For instance, a user usually has “email”, “birthday” and “full name”. We can use nested associative array in this case. Each username (key) will contain another array with key=>value pairs for email, birthday, and so forth



So we want to salute the user with “Hello {full name}” once they are navigated to the profile page. If we have persisted the username (the top-level key of the database array) in the session, we can extract it back and lookup the array to find their full name.

But to lookup the array in the “profile.php” page we need either to copy the whole array or to include back the “login.php” page. Both decisions are not the best choice one can take. The first one makes a huge code duplication and the second one will make the login form appear in the profile page.

The decision here is to extract the common component into another file. Let’s call it “database.php” and move the whole array there. This gives us the agility to include the database component wherever we want it. We do want it in the login page, but as well in the profile page.

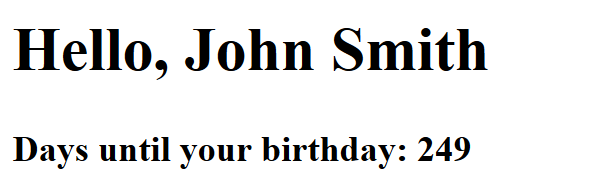
So just **include** **“**database.php**”** in both files.

The only thing needed here is to print the full name of the currently logged user:



## Profile Page – Days until Birthday

After the greeting we need to display the user how many days are left until they will celebrate a birthday.

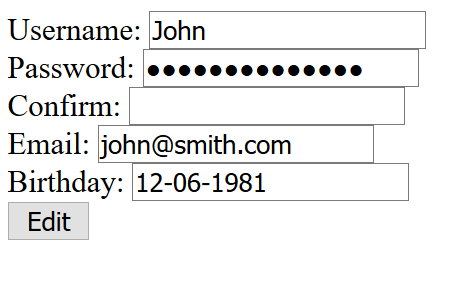


You may want to check how to work with dates - <http://php.net/manual/en/class.datetime.php>

## Profile Page – Edit Profile

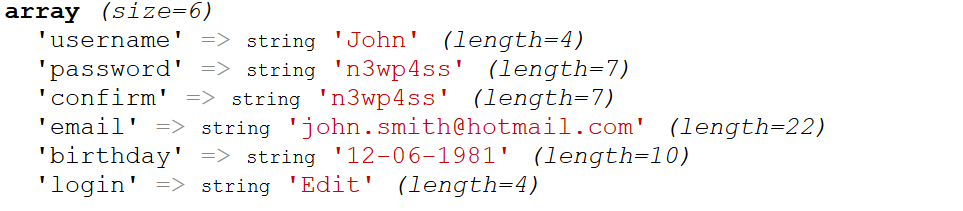
Let’s render a link in the profile page which will navigate the user to profile edit page (e.g. profile\_edit.php). Use the separation of concerns there too.

In the profile edit page, we need to display a form containing all the relevant information about the user already prefilled and give them the way to edit it



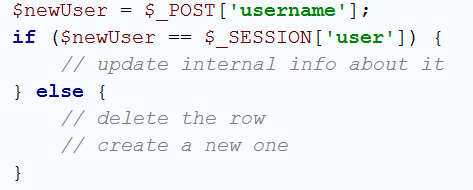
So, the tricky part here is how to edit the array persistently. Pretty much it’s impossible until we make some “hacks”. Here we need a real database, but we haven’t studied about it yet. If you are a curious reader, you can take a look at [MySQL](http://www.mysql.com/) and using it via [PDO in PHP](http://php.net/manual/en/book.pdo.php). But, in this exercise we will make it the unconventional way ☺

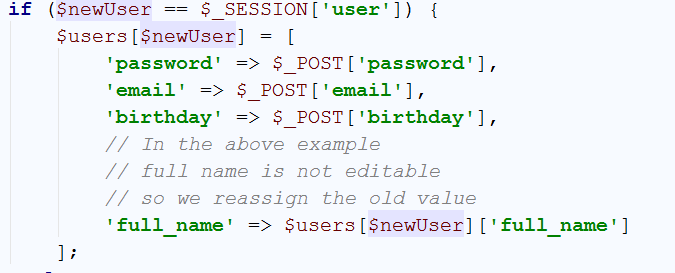
PHP can read and write to files and so is database.php – a file. Once the form is submitted and the password is validated with confirm password we have the necessary information to edit the array.



If the username was changed, we need to change the key in the $users array. This means we need to delete the the whole key=>value association and create a new one.

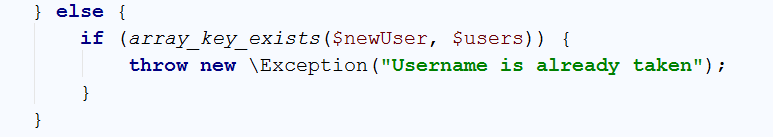
If the username is the same, we need just to change the internal keys regarding this user.



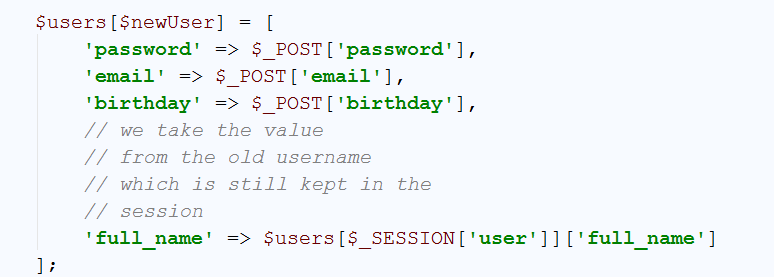


All good here. Now we need to take the other scenario. If the username was changed.

Well, there’s a tricky part there. What if “John” tries to change the username to “Hannah”? Would he edit her information? If we don’t stop it – probably yes. So we need to check whether the new username is already in our mini-database:



Then we can proceed with the update



Now we need to delete the old association

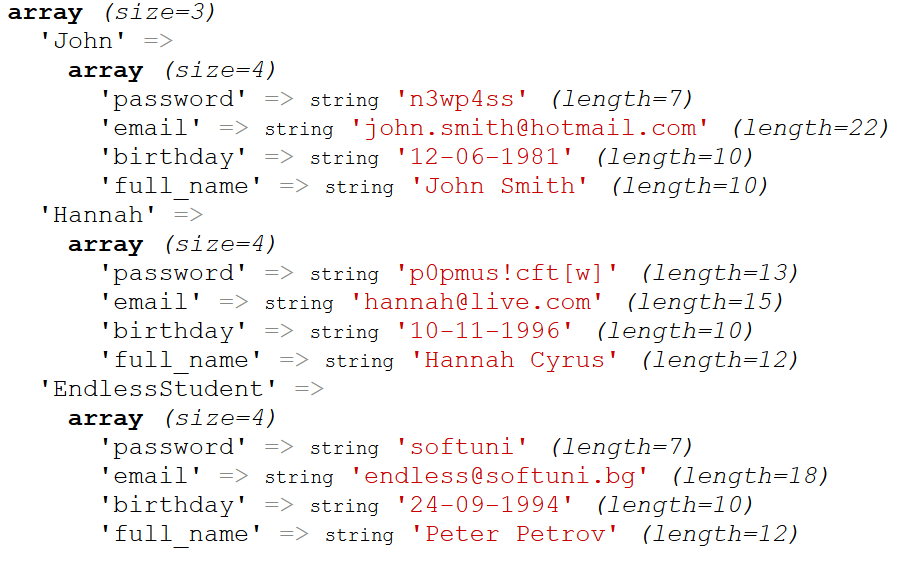


And change the value in the session to the new one so we can lookup the new values in the array (otherwise we will still try to find what is the birthday of “John”, but he is now e.g. “Johnathan”)

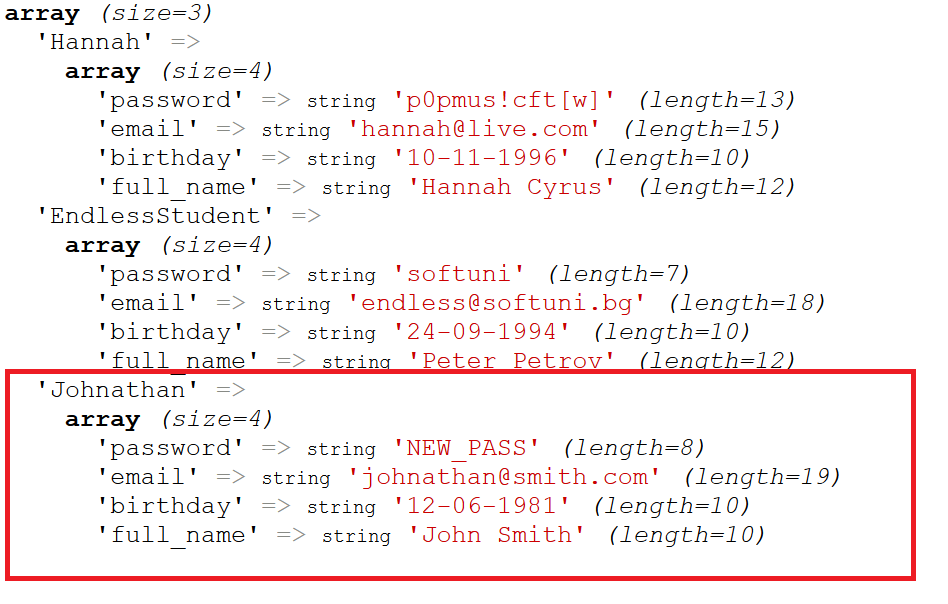


All cool, but we have not persisted the $users array. It will be gone once you refresh the page and load it again from database.php. Yet, it looks fine before the refresh:

Scenario 1 (Username not changed only email and password):



Scenario 2 (The username is changed along with the password and email):



Okay! But how to persist it. Well, one of the natural ways is to convert the array code (the variable declaration and its value) to text which we need to save in database.php as text again and override the old text.

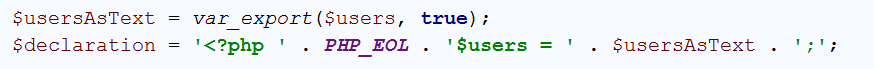
One function that could help us is var\_export(). It will convert the declaration into a text. It uses the old declaration of arrays with the array() directive instead of **[]**. You know, you can declare a variable as array both with $var = array(); and$var = [];?



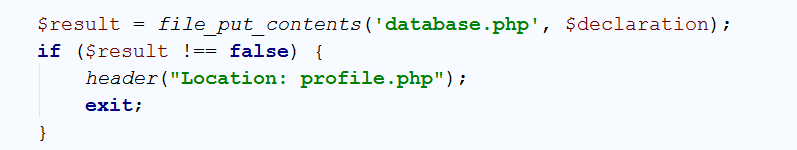
Which produces:



Now we can just concatenate this string to a string representing variable declaration and the whole text save it to a file



There are [many ways to manipulate files](http://php.net/manual/en/function.fputs.php). We will stick to [file\_put\_contents()](http://php.net/manual/en/function.file-put-contents.php).



As you can see you are now greeted with the new username and the database.php file is a little bit changed:



## Login Page – Security

Think about the security. Once the user is available to change things that are later displayed on the screen, one may set unwanted input. Escape the input properly.

## Registration

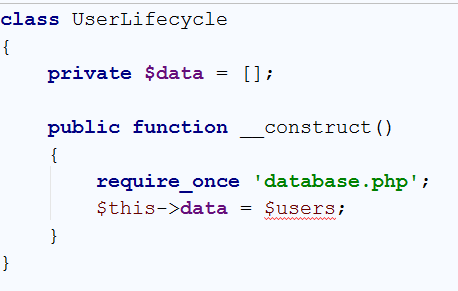
Create a registration form that fills new user entries to the $users array and after successful registration redirects to login page.

## \* Data Retrieval Class

We have a raw array used to retrieve user information. This is good until we change the strategy to real database. Then all of our pages needs to change. One core feature of the Object Oriented Programming is provoking reusability and smooths the change via abstraction.

We can create a class that has common methods for data retrieval and manipulation. Once we change the strategy we will change the behavior of that very class and not its usages. The usages will remain unchanged.

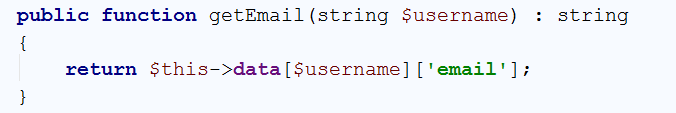
Let’s call the class UserLifecycle in a separate file called the same way. We need the data from database.php so we have many ways to take it from there. The naïve solution is to use it as a global variable which is not recommended and pollute the global scope. The other way is another class to retrieve it and send it to the class’ constructor ([dependency injection](https://en.wikipedia.org/wiki/Dependency_injection)), but it will be verbose for the current exercise. So we will just include the database in the constructor and assign it’s $users value to a local-scoped variable called field.



Now we need to define common methods. Let’s say:

* getEmail(string $username);
* getPassword(string $username);
* getFullName(string $username);
* getBirthday(string $username);
* */\*\*  
   \** ***@param*** *string $username The old username  
   \** ***@param*** *array $data The posted new data  
   \** ***@param*** *array $userInfo Referent information with user info e.g. session  
   \** ***@return*** *bool Whether the saving succeeded  
   \** ***@throws*** *Exception  
   \*/*edit(string $username, **array** $data, **array** &$userInfo) : bool
* register(array $data);

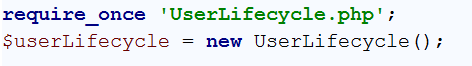
Example of getEmail():

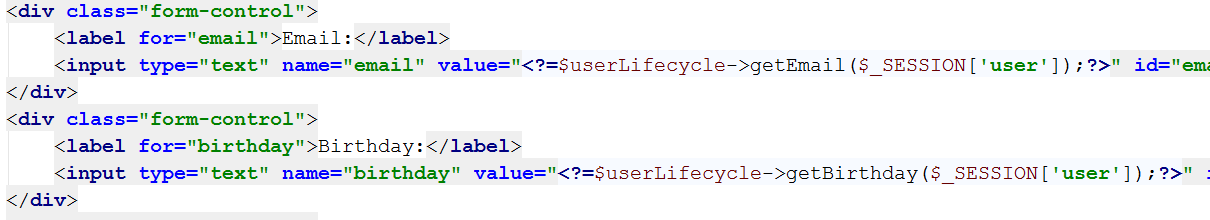


The edit function needs the old username, the new data and the session information in order to assign the new user back. It can access via the superglobals, but it’s a bad practice and provokes coupling e.g. to superglobals, the information may stop coming from there.

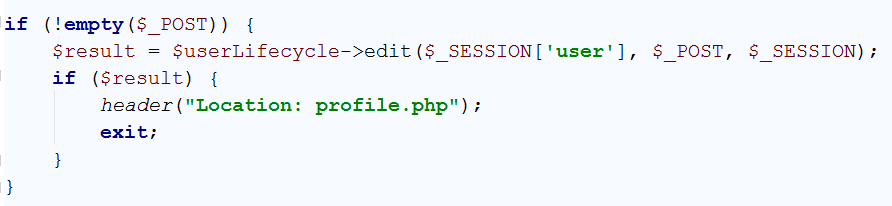


Now we can just use the class in our files instead of the database.php





And use it for modifying the user profile too:



You are left to implement the register() method.

## \* Style the Pages

Style the pages as you want. It’s recommended to use [Bootstrap](http://getbootstrap.com/getting-started/#examples). You may use headers on each page, form controls and so forth.

## \* Admin Role

Define a role in the $users array and don’t forget to update the UserLifecycle class. The value for each user for ROLE can be either USER or ADMIN

## \* Admin Role – List All Users

Create a page accessible only for logged in users with **ROLE => “USER”.** There have to be listed a grid with all users and their emails, fullnames, etc. Use **bootstrap’s grid system**.

## \* Admin Role – Manipulate Other Users

Create pages where users with Admin Role can access and can manipulate other users. E.g. edit or delete them. Use the UserLifecycle class to manage the operations.

You may add “Edit” and “Remove” buttons in the grid from the previous task. The edit button may redirect to profile\_edit.php and the remove button to profile\_remove.php prompting the administrator to confirm the deletion.

## \*\* Dispatch Actions

Instead of redirecting to different pages, create a central page called user\_controller.php**.** It will accept the actions via the form data or the query string and dispatch to the functions of UserLifecycle.

You can read about [Reflection](http://php.net/manual/en/book.reflection.php) and [Dynamic method invocation](http://stackoverflow.com/questions/251485/dynamic-class-method-invocation-in-php).