COSC260 Assignment 4: Animal Shelter Server-Side Registration

Your task is to develop a basic server-side user registration service in PHP. This service will operate in a very similar manner to the one your client-side application interacted with in Assignment 3.

Upon receiving a HTTP POST request, with the fields `username`, `fullName`, `dateOfBirth`, and `email`, your service should return a randomly generated temporary password to the client. All requests must be checked for invalid input, and if found, an error message and appropriate HTTP status code must be returned. All successful requests should be written to a file, in JSON format.

There are 5 components to the assignment:

A) **Error response function**
B) **Request validation**
C) **Data validation**
D) **Temporary password**
E) **Pet information database**

Project Structure

Please use the following structure:

```
assignment4
|       a4.html
+----css
|           your_custom.css (as many files as needed)
+----js
|           submit.js
+----php
|           AnimalShelter.php
|           register.php
```

Starter Code

You are provided with starter code, both for your server-side application, and for a client-side testing application. Please use the PHP starter code provided, as you will be required to use some of the pre-existing variables present. The client-side application is used to test your server-side code, as it allows you to make a POST request using a form, with output displayed. You should modify the `a4.html` file as necessary. **Your submission for Assignment 4 MUST contain your server-side code AND your a4.html file**.

Deploying Your Code

To test your PHP code, run your own web server using XAMPP. **Note: the marker will be using XAMPP AND Firefox so ensure that you test your solution using both.**

To get started, follow the installation and usage instructions here: https://www.simplilearn.com/tutorials/php-tutorial/php-using-xampp.

If your code does not run in Firefox, open the developer console by going to *Menu>More tools>Web Developer Tools*. If you see CORs error, follow the steps below to rectify it:

1. Go to Settings -> Privacy & Security
2. Scroll down to Certificates and click 'View Certificates'
3. A popup will open
4. Select Servers tab
5. Click 'Add Exception'
6. Enter the localhost URL to your API server and click 'Get Certificate'
7. Make sure you have checked 'Permanently store this exception'
8. Then 'Confirm Security Exception'.

If this doesn't work, you may choose to install the following add-on: https://addons.mozilla.org/en-US/firefox/addon/cors-unblock/, however, be aware that it poses a security risk, so ensure that you toggle off when running it isn't necessary.

Testing Your Service

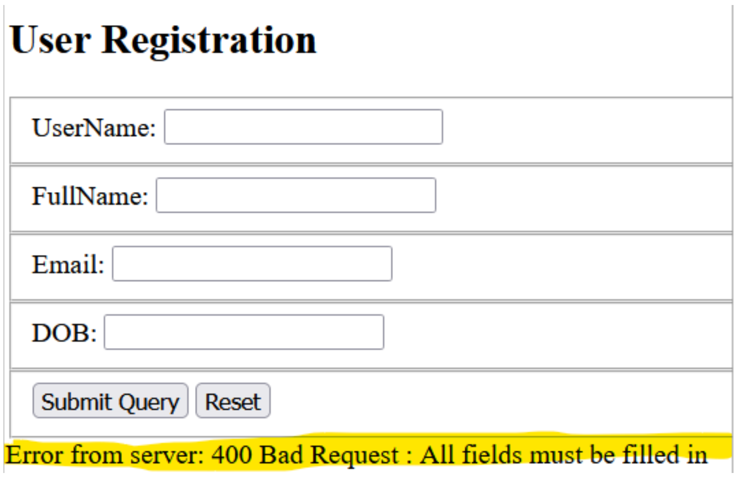
You are provided with a test client (HTML, CSS, and JS with jQuery) which allows you to submit data to your web server, using a web form: you can use this to test your server-side application. The client will display responses from the server both on the page and in the JS console. To use this testing client, you will need to change the `url` variable at the start of the `submit.js` file to the address of your web server.

If you wish to use this client for testing, please familiarize yourself with the code, and feel free to edit it if needed.

Make sure to also perform your own testing and edit the provided client as necessary. You should also use the 'network' and 'console' tabs of your browser developer tools to inspect requests and responses. It is highly recommended that you use a browser addon, such as Advanced Rest Client or RESTED, or use cURL to script your own tests. You could also write your own client-side tester using HTML, CSS, and JS!

A) Error Response

Implement a PHP function that returns a HTTP status code along with a custom message in the header and/or body to the client. Here is an example:



User Registration

UserName:	
FullName:	
Email:	
DOB:	

Error from server: 400 Bad Request : All fields must be filled in

This PHP function will be needed for all subsequent sections and should be used when an error occurs (invalid input, wrong request type etc.).

A good function definition would look like: `send_error($code, $message)` The function should take 2 input parameters, as follows:

`$code`	(Integer) will be the HTTP status code to return.
`$message`	(String) will be a custom message text.

The `code`variable contains a value which is also a key in the `$responses` array provided. This array maps integer codes with their appropriate reason text. **Make sure to use this array!** You can get the server protocol from the `$_SERVER` superglobal.

B) Request Validation

Requests must be validated to ensure they are correct for the service. This means the request must be of the right type and must contain valid data.

All incoming requests must meet the following criteria:

1) The request **must** be POST: No other request types are accepted.
2) The request **must** contain POST data: An empty body is invalid.
3) All required fields **must** be present in the POST data: `username`, `fullName`, `dateOfBirth`, and `email`.

If **any** of the criteria above is not met, an appropriate HTTP status code must be returned to the client. The body of the response must also contain the HTTP status code and reason, along with a custom error message. You must use your error response function from **Part A**.

C) Data Validation

All user data must be validated on the server-side, to ensure it is safe, complete, and correct. Even though client-side validation *may* have been conducted, there is no guarantee that client-side code has not been modified, or even executed at all.

All incoming POST data for the service must be validated to meet the conditions below.

POST Field	Description	Validation Requirements
`username`	Unique username	• MUST be between 8 and 20 characters long • MUST only contain at least ONE upper case letter. MUST contain at least one number and one special character (~!@#$%^&*)
`fullName`	Name of the user	• MUST contain at least ONE white space, separating first name from surname. • MUST only contain characters a-z (upper and lower case), - (hyphen), or ' (apostrophe)
`dateOfBirth`	User's birthdate, e.g., 02/03/1989	• MUST contain the date in dd:mm:yyyy format.
`email`	Email address	• MUST be validated using either: o The provided Regular Expression (See last page) o An appropriate regex found online (must be credited with the URL at which you found it)

If any of these validation requirements are not met, an appropriate error message should be returned, using your error response function from **Part A**. You may return the first error encountered, all present errors, or some other variation.

D) Temporary password

If all validation is successful, your service should return success response, and a randomly generated temporary password. The response should contain a JSON object with the format:

```
{
    "password": "some password here"
}
```

The password must commence with a randomly selected pet name from the following list:

```
pet_names = [“snowball”, “disco”, “jester”, “sandwiches”, “dasher”, “toffee”,
“tigerlilly”, “mushu”, “solemnbum”, “spats”, “bluey”, “wonda”]
```

Note: The pet names must be stored and processed on the server.

The password must contain all the characters in the username, arranged randomly **on the server**. It must end with the user’s age.

For example, if the username is Walter$hepman5 and the user’s date of birth is 07/03/1956, then the password may be mushuW$npahan5elmr66.

E) Pet Information Database

In a typical server-side application, user data would then be written to a database once it is validated. We will simulate a database by writing each successful request to a text file. For each request that passes validation, all received data should be written to file, in JSON format.

You **must** implement your "database" using **PHP classes**. An empty class file `AnimalShelter.php` has been provided in the `class/`directory.

You **must** have a class property for each of the received fields: `username`, `fullname`, `dateOfBirth`, `email`. Your class **must** implement the `JsonSerializable` interface and use this to write to the database file. This does mean that during the execution of your application, you will convert data from JSON (received from the client) to a `AnimalShelter` object, then back to JSON for writing.

Note that there is **no requirement** to be able to programmatically read data back from your file, you only need to append the new data to the end.

F) Resources

Regular Expression Hints

`^`	:	Matches the start of a string
`$`	:	Matches the end of a string
`[]`	:	Matches all characters and character ranges contained within
`\w`	:	Matches a "word" character, same as `[A-Za-z0-9_]`
`*`	:	Matches the previous selector 0 or more times
`+`	:	Matches the previous selector 1 or more times
`?`	:	Matches the previous selector 0 or 1 times ("optionally matches")

Simple Email Validation Regular Expression /`^[a-zA-Z-]([\w-.]+)?@([\w-]+\.)+[\w]+$`/

RESTED Browser Addon
https://addons.mozilla.org/en-US/firefox/addon/rested
https://chrome.google.com/webstore/detail/rested/eelcnbccaccipfolokglfhhmapdchbfg

HTTP Status Codes: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Links - PHP Documentation

Arrays: https://secure.php.net/manual/en/book.array.php
Functions: https://secure.php.net/manual/en/language.functions.php
Classes & Objects: https://secure.php.net/manual/en/language.oop5.php
Superglobals: https://secure.php.net/manual/en/language.variables.superglobals.php
$_SERVER: https://secure.php.net/manual/en/reserved.variables.server.php
isset(): https://secure.php.net/manual/en/function.isset.php
array_push(): https://secure.php.net/manual/en/function.array-push.php
json_encode(): https://secure.php.net/manual/en/function.json-encode.php
header(): https://secure.php.net/manual/en/function.header.php
file_put_contents(): https://secure.php.net/manual/en/function.file-put-contents.php
JsonSerializable: https://secure.php.net/manual/en/jsonserializable.jsonserialize.php