UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS 216 Practical Assignment 3

- Date Issued: **16 March 2020**
- Date Due: **5 April 2020** before **10:00**
- Submission Procedure: **Upload to the web server (`wheatley`)** and CS Web
- This assignment consists of **4 tasks** for a total of **95 marks**.

1 Introduction

During this practical assignment you will be designing and developing a web site that will showcase a Music Database similar to what can be seen from ALLMUSIC (`https://www.allmusic.com`). Each assignment will build of the other in attempt to have a fully functional Music database listing website at the end of all the practicals. **NB: It is important that you do not miss any practicals or you will fall behind.**

After successful completion of this assignment you should be able to create web pages in PHP which complies to the HTML5 and JavaScript Standards. The specific web page for this assignment will showcase the following functionality:

- Register functionality
- Create PHP API
- API key generation and authorization

2 Constraints

1. You must complete this assignment individually.
2. You may ask the Teaching Assistants for help but they will not be able to give you the solutions.
3. You must produce all of the source files yourself; you may not use any tool to generate source files or fragments thereof automatically.
4. Your assignment will be viewed using Firefox (the version in the labs or newer) so be sure to test your assignment in this browser. Nevertheless, you should take care to follow published standards and make sure that your assignment works in as many browsers as possible.
5. **You may use JavaScript or/and JQuery for this however no other libraries are allowed. You must use the PHP cURL library for the API you developing.**
6. **Server-side scripting should be done using an Object Oriented approach.**

3 Submission Instructions

You are required to upload all your source files (e.g. HTML5 documents, any images, etc.) to the web server (`wheatley`) and the CS Web in an compressed (zip) archive. Make sure that you test your submission to the web server thoroughly. All the menu items, links, buttons, *etc.* must work and all your images must load. Make sure that practical assignment works on the web server before the deadline. No late submissions will be accepted, so make sure you upload in good time. From the stipulated deadline time to after the last demonstration of the assignment on the Friday evening, the web server will not be accepting updates to the files.

4 Online resources

Databases - `http://www.smartwebby.com/PHP/database_table_create.asp`

PHP Sessions - `http://www.w3schools.com/php/php_sessions.asp`

PHPMyAdmin - `http://www.phpmyadmin.net/home_page/index.php`

Timestamps - `https://en.wikipedia.org/wiki/Unix_time`

Cookie - `https://www.w3schools.com/js/js_cookies.asp`

PHP Headers - `http://php.net/manual/en/function.header.php`

PHP cURL - `http://php.net/manual/en/book.curl.php`

PHP GET - `http://php.net/manual/en/reserved.variables.get.php`

PHP POST - `http://php.net/manual/en/reserved.variables.post.php`

5 Rubric for marking

Setup	
Include	3
Header	2
Footer	1
DB	4
User Registration	
SQL	10
Validation	15
Security	5
Music PHP API	
Methods	4
Type	20
Key	13
Return	10
External API's	2
Headers	1
Upload	
Does not work on `wheatley`	**-10**
Not uploaded to CS web	**-40**
Bonus	5
Total	**90**

6 Assignment Instructions

Task 1: Basic setup and page construction ....(10 marks)

You will need to create the following pages as a skeleton for your project. Note that these are the minimum required files and you are welcome to add anything extra you think is necessary.

- config.php, header.php, footer.php, api.php
- (login.php, validate-login.php, logout.php) – used in Practical 4
- signup.php, validate-signup.php

The objective of this task is firstly to be able to build a login system for your site that you have been developing by making use of the "include" function to stitch pages together. Your header.php file should contain the navbar you made from your previous assignments (such that it can be repeated on each php page). The footer.php should contain the footer of your page. This means that the index.php page should get the header and footer information from the header.php page and footer.php respectively. It is highly recommended that you include config.php in the header.php page also. This config file serves as your database connection as well as global variables and other configurations you might need for your website.

The navigation for this entire project should be stored in a single file (header.php) and included in every page. Your navbar should be in the header.php as it is needed on every page and should include links to login and register. If the user is already logged in, the name of the user should instead be displayed on the website with the logout button (the functionality of logging in and out will be implemented in Practical 4).

The second objective of this task is to setup a database on Wheatley. It is recommended that you do this through the use of phpMyAdmin. **Hint:** download phpMyAdmin and extract the contents of the zip. Then use an FTP program (such as FileZilla) to copy phpMyAdmin over to your Wheatley folder (wheatley.cs.up.ac.za/uXXXXXXXX/phpMyAdmin). You can now create a database and edit tables by visiting this url. Alternatives to phpMyAdmin are: mysql command line client (`https://dev.mysql.com/doc/refman/8.0/en/mysql.html`), MYSQL workbench (`https://www.mysql.com/products/workbench/`), etc.

For this practical you will need at least 1 table in your database. This table will contain all your user information so make sure you include the following fields: "id", "name", "surname", "email", "password", "API key".

Task 2: User Registration ....(30 marks)

This task focuses on the signup page and signup-validation function. The goal is for the user to be able to enter in various details on a form on the signup page and register an account. When the form is submitted, it must be received by the signup-validation function which checks (using JavaScript and PHP)[i.e. Both client and server-side validation] whether the information is correct or not. If it is valid, the user is added to the relevant table in your database.

- Create a signup form on the signup page with the following fields: "name", "surname", "email", "password".
- Before allowing the user to submit the form, client-side JavaScript must be used to check that all the fields are filled out correctly. This means that the email address should have an '@' symbol and the password should be longer than 8 characters, containing upper and lower case letters, at least one digit and one symbol (You need to explain why you think this is necessary in the demo). You must make use of JS Regex for this, no plugins or libraries are allowed. You may only copy a Regex string for the email from the web, but ensure you have the best one, other Regex needs to be done by yourself. Using HTML required attribute is not enough since one can easily change the type of input box.
- Make sure that the form submits the information via post to signup-validation.
- Signup-validation must validate the user (check if the user exists as well as validate the input fields both on client and server side). You may only use built-in PHP functionality for this, no libraries or frameworks are allowed.
- Signup-validation must then perform the necessary MySQL query to insert the user into the database table.

- It should go without saying that passwords should not be stored in plain text. You will need to choose a Hashing algorithm (you may **not** use Blowfish), you will be evaluated on your choice.
- You will need to add salt to your passwords to make them more secure, your salt should be dynamically created and not be a fixed string.
- If the email address is already in the table, the query should be ignored, and an error message displayed. Hint: make use of unique keys.
- You must be able to demonstrate to the marker that the user was not in the table before you added them through the signup page.
- an API key needs to be generated once all the validation has been successful. The key should be an Alpha-numeric string consisting of at least 10 characters. You will be assessed according to how you generated this key. This API key should be shown to the user once it is created. Task 3 provides more detail as to what this key is used for.

Task 3: Create a PHP API .... (50 marks)

This task requires you to create PHP API for your Music listing website in an Object Oriented manner. Hence you need to make use of PHP classes. **Hint:** take a look at how singletons work. You should save the API in a file called "api.php" and it should reside in your home folder. Your API should only produce/consume structured JSON data.

Your API must make use of an API key for each request in order to prevent unauthorized access or security attacks. The API key is simply a randomly generated key consisting of alphanumeric characters for an authorized party.

Your API will also need to source the data from 2 or more external APIs (such as LastFM, Spotify, Deezer, SoundCloud etc.). Creating APIs are necessary to have a single consistent backend interface for data exchange, that is sourcing information from other external sources to make it more accurate and robust. This is all done server side and the end-user is not aware of what is going on in the background as opposed to using JQuery/JS to perform the same purpose. APIs are standalone and can be used through any interface that supports it (REST/SOAP). Make sure that your API works as you will need it for the remainder of the practicals (if you cannot get the API to work, as a last resort you may use mock data, however, that won't earn you many marks). In order to make server side external requests in PHP you will need to use the PHP cURL library (`http://php.net/manual/en/curl.examples.php`, `https://stackoverflow.com/questions/3062324/what-is-curl-in-php`, `https://www.startutorial.com/articles/view/php-curl`). The cURL library is used to access/send data to/from web pages (web resources). It supports many internet protocols for connecting to the resource required.

IMPORTANT: In order for Wheatley to fulfil external requests the following optional parameter must be used when making a cURL request:

```
curl_setopt($ch, CURLOPT_PROXY, "phugeet.cs.up.ac.za:3128");
```

Your API must provide at least the following fields as POST parameters (title, artwork, year, genre, album, duration, Billboard ranking, user rating). Since you are only using one route for the main API functionality you need to have your own structured request and should at least contain (title, ranking, return) POST parameters. You may use the example request and response given below **as a basis**, you **should** extend on it and **add more parameters** as you require:

Request
url: wheatley.cs.up.ac.za/uXXXXXXXX/api.php – URL to your PHP API
method: POST – HTTP method

Query String

Parameter	**Required**	**Description**
key	Optional only for LOGIN	the user's API key in this Practical you can simply hard code a valid API key. In Practical 4 you will use the login method to retrieve and store the API key.
type	Required	method to identify what information is needed, consists of the following values: ***info*** - simple information about the requested music, ***update*** - to update user preferences and settings, such as filters, theme, etc. (introduced in Practical 4), ***login*** - to authenticate a registered user (using username and password) and retrieve API key for a user (introduced in Practical 4), ***rate*** - registered user has the ability to rate a music (introduced in Practical 4), ***track*** - tracks audio/clip progress (seen in Home Work Assignment).
title	Optional	specifies the title of the music being requested, to retrieve all titles the wild card '*' should be used. (Since there are thousands of music you can limit it to show a minimum of 20 songs)
ranking	Optional	specifies the Billboard rank for a music
...	...	you may add more parameters as you require ...
return	Required	specifies the fields to be returned by the API, to return all fields the wildcard '*' can be used.

Request Example Here is what an example request using the GET method, note you should use POST method, this only serves as an example. The example url would be:

```
http://wheatley.cs.up.ac.za/uXXXXXXXX/api.php?key=7asda7865dv71sda&type=info&title=
  ↪ Saturday+Night&return[]=title&return[]=rank&return[]=release
```

Response Object
Your API should return structured a JSON Object as a response. Here is what the response to the request example give above should be:

```
{
        "status": "success",
        "timestamp": 1549286137,
        "data": [{
                        "title": "Saturday Night",
                        "billRank": "400",
                        "release": "8 February 2019"
        }]
}
```

status - defines whether the request was successful (success) or unsuccessful if an error occurred or a external API is not reachable (error).
timestamp - current timestamp (`https://en.wikipedia.org/wiki/Unix_time`) that will be used for the HomeWork Assignment.
data - the fields to be returned from the requested "return" parameter.

Task 4: Bonus .... (5 marks)

As a bonus question, you can add an extra functionality for the API by adding more security features and using the timestamp feature to do something cool like refreshing the data once a specific time has elapsed or caching data to make the website load faster. Be creative!