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

COS 216 Practical Assignment 3

- Date Issued: **27 March 2023**
- Date Due: **17 April 2023** before **08:00**
- Submission Procedure: **Upload to the web server (`wheatley`) + clickUP**
- This assignment consists of **4 tasks** for a total of **95 marks**.

1 Introduction

During this practical you will be creating a site to view and compare different cars and brands. The idea is to give users of the site the ability to look at different cars and see the specs helping them make the right choice of which car to buy. Users of the site can choose to view car models, view car brands, compare cars and even use the find me a car feature.

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:

- Using a MySQL DB with PHP
- Create PHP API
- User registration with an 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.(**This includes ChatGPT!!**)
4. Your assignment will be viewed using Brave Web Browser (`https://brave.com/`) 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 utilise any text editor or IDE, upon an OS of your choice, again, as long as you do not make use of any tools to generate your assignment. (**This includes ChatGPT!!**)
6. All written code should contain comments including your name, surname and student number at the top of each file.
7. Your assignment must work on the `wheatley` web server, as you will be marked off there. However you are free and encouraged to **Develop** using a local tool like XAMPP and later move over to wheatley however it must work off wheatley during marking.
8. **You may use JavaScript and/or JQuery(You may not use Jquery for AJAX) for this however no other libraries are allowed (unless specified in a previous practical). You must use the PHP cURL library for the API you are developing.**

9. **Server-side scripting should be done using an Object-Oriented approach.**
10. You must **NOT** use any features from PHP Version 8 or higher. Wheatley is on PHP version 7.3 therefore while version 8 features may work locally they will not work on Wheatley.

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 clickUP in a 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 your practical assignment works on the web server before the deadline. No late submissions will be accepted, so make sure you upload in good time. The server will not be accepting any uploads and updates to files from the stipulated deadline time until the end of the marking week (Thursday at 3pm).

The deadline is on Sunday but we will allow you to upload until Monday 8am. After this NO more submissions will be accepted.

Note, `wheatley` is currently available from anywhere. But do not rely that outside access from the UP network will always work as intended. You must therefore make sure that you `ftp` your assignment to the web server. Also make sure that you do this in good time. A snapshot of the web server will be taken just after the submission was due and only files in the snapshot will be marked.

Practicals are marked by demonstrating your practical to a tutor during the allotted marking weeks. **If you do not demonstrate your practical you will receive 0 for the practical**. You will only be marked in the practical session that you have booked on the cs portal, if you miss it, you will receive 0 (Unless special permissions have been granted by the lecturer. i.e. You were sick and able to provide a sick note).

NB: You must also submit a ReadMe.txt file.
It should detail the following:

- how to use your website
- default login details (username and password) for a user you have on your API
- any functionality not implemented
- explanations for the password requirements, choice of hashing algorithm and generation of API keys

4 Online resources

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

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`

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

5 Rubric for marking

Setup	
ReadME	5
Include	3
Header	2
Footer	1
DB	4
User Registration	
SQL	10
Validation	5
Security	5
Cars PHP API & SQL	
API Setup	4
Headers	1
Each row includes API & SQL & Error logic	
Type	3
APIkey	3
Return	10
Fuzzy	5
Limit	4
Sort	8
Order	2
Search	15
Image	5
Upload/Deductions	
Does not work on `wheatley`	**-20**
Not uploaded to clickUP	**-95**
Not demoed	**-95**
SQL Injection Vulnerable	**-5**
Bonus	5
Total	**95**

6 Assignment Instructions

Task 1: Basic setup and page construction .... (15 marks)

You will need to create the following pages as a skeleton for your project:

- config.php, header.php, footer.php, api.php
- login.php, validate-login.php, logout.php – used in Practical 4
- signup.php, validate-signup.php – see Task 2

Note: These are the minimum required files and you are welcome to add anything extra you think is necessary.

The objective of this task is firstly to make 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 using the include function).
- Your footer.php file should contain the footer of your website.

This means that every 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 as well. This config file serves as your database connection as well as global variables and other configurations you might need for your website.

This also means that all pages (HTML files) from the previous practical (With the exception of the launch page with links to all practicals) should be converted to PHP files.

The navigation for this entire project should be stored in a single file (header.php) and included in every page. Your navbar should also include links to login and register. If the user is already logged in, there should be no login and register links. Instead, the name of the user should be displayed on the website with a 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. You can access it at `https://wheatley.cs.up.ac.za/phpmyadmin/`. The username is your student number and the password if found in the *db_password* file found in your Wheatley root directory.

The last objective of this task is to import a given MySQL DB Dump. Along with the specification a Database dump will be found on clickup. When imported correctly it should result in a populated cars table. This is the table you will use in a later task to pull data for your API.

For this practical you will need at least 1 table in your database apart from the cars table. 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 .... (20 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 on your car website. 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. Of course, you will need a way for the user to easily register/login to the site.

You should complete the following functionality for this task:

- Create a signup form on the signup page (signup.php) 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, contain upper and lower case letters, at least one digit and one symbol (**NB: You need to explain why you think this is necessary in your ReadMe.txt**

file). 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. Any other Regex needs to be done by yourself.
Note: Using the 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. The user should also be notified client side if there is an issue such as a duplicate email.
- 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. (**NB: You will be evaluated on your choice so make sure to include this in your ReadMe.txt file**).
- You will need to **add salt** to your passwords to make them more secure. They should also be above 10 characters, as shorter salts are more susceptible to brute force attacks. 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.
- 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. (**NB: You will be assessed on how you generated this key so make sure to include this in your ReadMe.txt file**). 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 .... (60 marks)

This task requires you to **create a PHP API** for your car 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 root folder.
NB: Your API should only produce/consume structured JSON data.
Also note that you do not yet need to exclusively use this API in your website yet. You simply need to make sure the API itself is functional.

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. For this practical you may simply hard code it on the client and server side.

You will be recreating a modified version the "Get All Cars" section of the API used for Practical 2 `https://wheatley.cs.up.ac.za/api/doc.html`. You will be modifying it slightly so that the car image is included in the request. Included with the practical is a MySQL DB dump that you should have imported from task 1.

Based on on the post parameters of the request you should build query and query the database and call the Wheatley image endpoint and return that data to the user in the correct format. You should use SQL Queries to extract data from the database dynamically. i.e you cannot return the entire database with an SQL query and do all data processing from PHP side as that wastes resources. However you can clean up the data a little from your query such as removing some rows/columns, marks will be awarded for how efficient you solution is.

From the user side it may seem that your API did all the work but its common for APIs to call other APIs and parse that data before it is returned. 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. 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. You will be using PHP Curl to get the images for the all the cars. Here are some additional resources:

- `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`

You may use the example requests and responses 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)

JSON POST body

Parameter	**Required**	**Description**
apikey	Required	the user's API key . In this Practical you can simply hard code a valid API key on the client side. 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: • ***GetAllCars*** - Returns information about cars • More parameters will be introduced in Practical 4
limit	Optional	A number between 1 and 500 indicating how many results should be returned
sort	Optional	If sort is used. You should sort on the listed field. Sort can be any of **but not limited to** the following: ['id_trim', 'make', 'model', 'year_from', 'year_to', 'max_speed_km_per_h']
order	Optional	If sort is used. Order can be "ASC" or "DESC" for ascending or descending respectively
fuzzy	Optional	Indicates if fuzzy search should be used, default value is true
search	Optional	A JSON object where the keys are columns of the data and the values are the search terms. Columns can be any of the following. *['make', 'model', 'body_type', 'engine_type', 'transmission'].*
...	...	you may add more parameters as you require ...
return	Required	specifies the fields to be returned by the API. In order to return all fields the wildcard '*' should be used. The fields you can return are *['id_trim', 'make', 'model', 'generation', 'year_from', 'year_to', 'series', 'trim', 'body_type', 'number_of_seats','length_mm', 'width_mm', 'height_mm', 'number_of_cylinders', 'engine_type', 'drive_wheels', 'transmission', 'max_speed_km_per_h', 'image']*

Request Example Here is an example of the post parameters you would use.

Example Post body

```
        {
                "type":"GetAllCars",
                "apikey":"a9198b68355f78830054c31a39916b7f",
                "return":["id_trim", "make", "model", "max_speed_km_per_h","image"],
                "search":{
                        "make":"audi",
                        "model":"q3"
                },
                "fuzzy":true

        }
```

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

```
        {
                "status": "success",
                "timestamp":"1679507636541"
                "data": [
                {
                        "id_trim": 3567,
                        "make": "Audi",
                        "model": "RS Q3",
                        "max_speed_km_per_h": 250,
                        "image":"https://wheatley.cs.up.ac.za/api/images/models/audi_rs
                        ↪  q3.jpg"
                },
                {
                        "id_trim": 3389,
                        "make": "Audi",
                        "model": "Q3",
                        "max_speed_km_per_h": 233,
                        "image":"https://wheatley.cs.up.ac.za/api/images/models/audi_q3.⌋
                        ↪  jpg"
                }
                ]
        }
```

ALL responses should have these three fields
status - defines whether the request was successful (`success`) or unsuccessful if an error occurred or an external API is not reachable (`error`).
timestamp - the 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.

Error Handling Your API should should be able to cater for invalid input by returning an error back that will be handled client side. Things like misspelling/missing/malformed keys should be catered for. You will be marked on how many cases your code caters for. You must follow the Error Response object structure as we will use a script to mark this section.

Error Response Object

If you post without any post data your API should return a structured JSON Error Object as a response. It should look something like this: Reminder that your timestamp and/or data should change based on the nature of the error.

```
        {
                "status": "error",
                "timestamp": 1679391940921,
                "data": "Error. Post parameters are missing"
        }
```

Security As we are dealing with SQL it is **CRUCIAL** your system is safe from SQL injection. (-5 if SQL injection is possible)

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!