LAB 2: Database and OOP

This Lab assignment consists of three tasks:

```
A) installation of MySQL server and access tools, setting up sample data;

B) combination of user input handling and database access;

C) writing a simple class for auditing user requests.
```

--> Collect the source code and data files from GitHub:
https://github.com/rauhvargers/TTII_2023_labs/tree/Lab2

--> Please read all comments given with PHP codes, look for "TODO" tasks

Task A – The database

Subtask 1:

Installing the database server

You will need a working MySQL server for this task.

For Wamp.Net users: go to "Status" page, click "Install Packages", choose "MySQL 10.0.30" from the list.

For XAMPP users: the MySQL service is bundled with XAMPP, just click "Start" with the default settings.

As the result, you should have MySQL running on your computer, listening to port 3306.

Subtask 2

You will also need a tool for accessing MySQL. Pick one of the following:

- **MySQL Workbench** (https://dev.mysql.com/downloads/workbench/) is a great tool for local use.
- **PhpMyAdmin** is a standard solution for remote management of MySQL, when you can only access the server over HTTP.
    - For Wamp.Net users: create an empty site; assign it a domain name (e.g. myadmin.com). Then, from the status page install PhpMyAdmin, assign it the empty site you just created.
    - For XAMPP users: it is bundled with XAMPP, just visit http://localhost/phpmyadmin
- **Adminer** is similar to PhpMyAdmin, but works with other DB management systems, too.

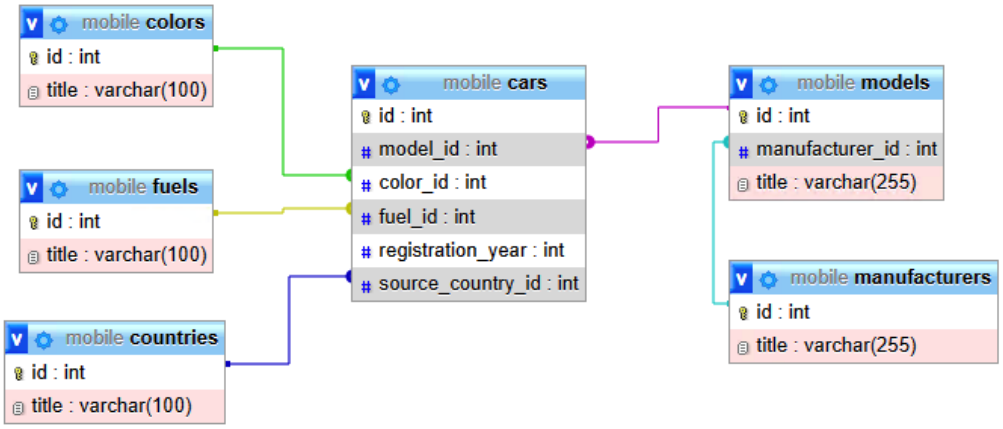
Subtask 3

Connect to your local MySQL instance and create a new database. Set database name to "**mobile**".

Execute the database setup scripts found in /database/ folder of GitHub repo.

a) If using PhpMyAdmin, you can try importing the 000_database_and_data.sql.zip (you may need to increase the upload file limits, follow this guide then: https://www.keycdn.com/support/413-request-entity-too-large#nginx)

b) If the .zip file option fails, you can execute scripts 001-007 manually.
As the result, you should have a database with tables like this, filled with some data:



Task B – Working with user inputs and database.

Subtask 1: Create a website for Lab 2.
Wamp.Net users: Create a new site for Lab2 with the following properties:

a Domain name: lab2.dev
b Domain aliases: www

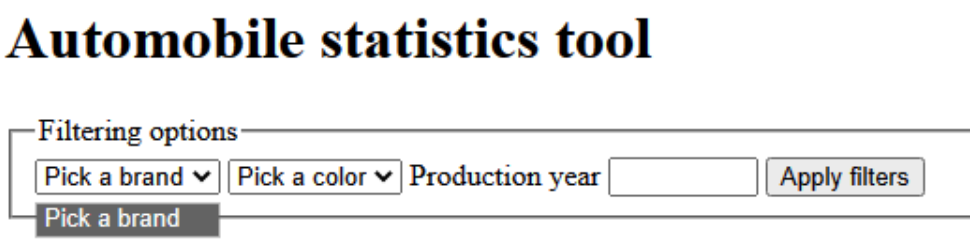
This will be your "web root" for the assignment. It will be accessible on the web as https://lab2.dev/

XAMPP users:

Make a subfolder in c:\xampp\htdocs\lab2.
It will be accessible on the web at http://localhost/lab2/

Move the files found in Lab assignment's GitHub repo folder "web" into your web root
Try accessing the address of task 1: (Wamp.Net: https://lab2.dev/task1/; XAMPP: http://localhost/lab2/task1/)

In folder "task1", you will find files "index.php" and "view.php". The index file should contain all the business logic, while the view.php should only display information prepared by the index.php.
A page like this should open:



Subtask2: Fill the classifier boxes (brand/manufacturer and color).
Find a "TODO" task that asks you to set up a Mysql connection.
Paste the following code there:

```
$server = "127.0.0.1:3306";
$database = "mobile";
$user = "root";
$password = "password";
$mysqli = new mysqli($server, $user, $password, $database);
```

The variable $mysqli will now hold a reference to MySQL connection. Use this variable throughout the script.

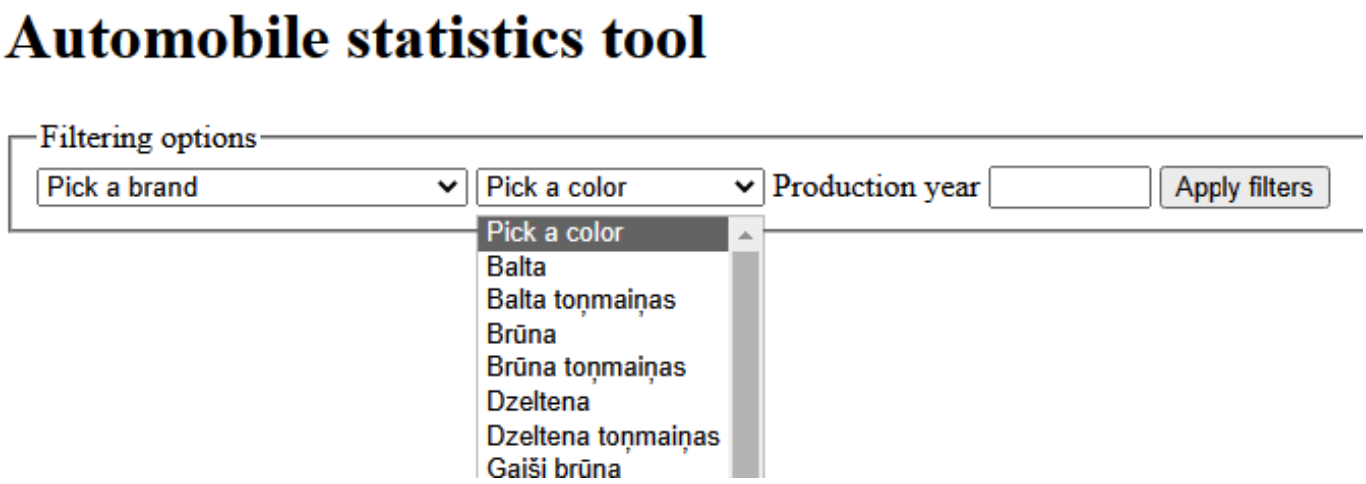
Find a "TODO" task which asks you to fill the manufacturer IDs. Paste the following code there (it reads table contents from MySQL table and then row by row appends to an array of key=>value entities):

```
$manufacturers = array();
$manufacturer_handle = $mysqli->query("select id, title from manufacturers
order by title");

while ($row = $manufacturer_handle->fetch_assoc()) {
    $manufacturers [$row["id"]] = $row["title"];
}
```

Repeat similar steps to fill the array $colors with data from database table `colors`.

As the result, you should see the dropdown lists filled with items:



When you click the "Apply filters" button, the data is submitted via GET parameters to "index.php" script, e.g. `http://www.lab2.dev/task1/index.php?manufacturer=548&color=5&year=`
Only the ID values of manufacturer and color are submitted.
The "Production year" is a free form field, any data can potentially be submitted there.

Subtask 3:

Query the database and display the results:

You want to retrieve a report on the number of cars registered in given year, having the given manufacturer and color; broken down by model.

The query might be something similar to this one:

```
select
    manufacturers.title as manufacturer,
    models.title as model,
    count(*) as count
from
    manufacturers
    inner join models on manufacturer_id = manufacturers.id
    inner join cars on cars.model_id = models.id
where
    manufacturer_id = 2713      #volkswagen
    and color_id = 13           #black
    and cars.registration_year = 2019
group by
    manufacturers.title,
    models.title
order by
    count desc
```

Which returns a data set like this

manufacturer	model	count
VW	TIGUAN	96
VW	PASSAT	90
VW	GOLF	36
VW	MULTIVAN	22
VW	T-ROC	18
VW	AMAROK	15
VW	CADDY	10
VW	TOUAREG	10
VW	T-CROSS	10
VW	ARTEON	7
VW	TOURAN	3
VW	SHARAN	2
VW	UP!	2
VW	POLO	1
VW	JETTA	1
VW	CRAFTER	1

In the example above, the value “2173” should be replaced by value of GET parameter ‘manufacturer’, the value “13” – replaced by value of GET parameter “color” and the value of “2019” - replaced by the gET parameter “year”.

Remember to check if given GET parameter is set and if it is of expected type! If any parameter is missing or of wrong type, set an error message in variable $error and do not query the database.

Remember to use mysqli_prepare() and bind_param() function to set the parameter values!

Modify the view.php file to output the results of your query in tabular view.

Task C – Access logger

You want to add some basic usage logging functionality to your application.
When a particular set of data is requested by the client, you want to append the following string to a logfile on the disk:

[90.139.7.252][2023-03-09T23:01:10+00:00][manufacturer=548&color=5&year=][ERROR]
The data can be collected from
IP address: $_SERVER['REMOTE_ADDR']
Formatted UTC time: date('c', time());
Query string parameters: $_SERVER['QUERY_STRING']
The last part informs whether the input was considered valid or not. If the variable $error is set, it should state ERROR, otherwise – OK.

Edit the file "task2/logger.php" to create a code class which does the logging.
Create a class called "Logger". Paste the following code:

```
<?php
class Logger
{

}
```

Inside the class body, add some fields where information will be stored during the life cycle of Logger instance:

```
    #ip and GET parameters cannot change during execution
    private string $ip;
    private string $queryString;
    #this will be passed to constructor
    private string $logPath;
```

Add a constructor which initializes the field values and gets the logfile path from the caller:

```
    public function __construct($logPath){
        $this->ip = $_SERVER["REMOTE_ADDR"];
        $this->queryString= $_SERVER["QUERY_STRING"];
        $this->logPath = $logPath;
    }
```

Add a function that generates a formatted string according to requirements given above:
Note: the function has a modifier 'protected': no callers from outside are allowed to call 'generateLine' directly, however, if some class were to inherit from our Logger, it would be allowed to call the generator method.

```
    protected function generateLine($result)
    {
        $date = date('c', time());
        return sprintf("[%s][%s][%s][%s]\n",
              $this->ip, $date, $this->queryString, $result);
    }
```

Finally, add a public method for actual logging:

```
    public function log($result) {
        file_put_contents($this->logPath,
            $this->generateLine($result), FILE_APPEND);
    }
```

When the class is ready for use, include its definition in task1/index.php. Create an instance of Logger class and call the 'Log' method.

```
require("../task2/logger.php");
$logger = new Logger("./out.txt");
$logger->log("OK");
```

You will need to modify the code above and pass it two values: instead of './out.txt', the location of logfile (e.g. c:\temp\logfile.txt; should be outside of web-accessible folder tree) and the value of result (set it to "OK" if $error is not set, set it to " ERROR" if there is an error).