Sunny Day Continues

Today we have two assignments 😊

Long but easy Lab 😊 😊 😊

**Assignment A:**

*Use the project which we did yesterday , but make the communication with the open weather API done by Guzzle not Curl*

**The Implementation steps:**

1. Take the Composer.Json file in the resources folder, read it .
2. Run the composer install command so the composer installs what needs to be installed .
3. Remove the existing autoload.php file in the project (our custom autoload file) we will use composer’s autoloader only in the future !!!
4. Remove the require (“autoload.php”) from the index file
5. Require the autoloader of composer in the index file
6. Duplicate the Model/weather.php class , so you have another class with same content with name: Model/GuzzleWeather.php
7. Edit the get\_weather($cityid) function to use Guzzle not Curl
8. In the index file make the $weather object be initialized from the GuzzleWeather not from Weather
9. Run and make sure all is fine
10. Don’t forget composer dump-autoload whenever you change any thing in autoload section in composer.json or whenever a class file not found error and you are sure it’s found

That’s it Good Luck 😊 😊

**Assignment B**

**Creating a RESTFul web service Part 1 ((to be continued))**

**Note :**

**This assignment will be done in Day02 and continued in DAY03**

**Back to the ABD Glasses shop that we used in the 4rth and 5th day of the PHP Course, now the shop is bigger so the owner decide to create a mobile application beside his web application to manage it’s items (show item details, add new item, edit and delete existing items)**

**To do all that we need a RESTFUL web service for the shopwhich will create all CRUD operations for the items**

**The web service will be used by both imaginary front end and mobile team so we need to do our best so the imaginary teams are happy with the web service they will all use 😊 😊 😊**

**Let see how the web service will be used**

**Each item will have a URL with his ID like the following**

Localhost/GlassShopAPI.php/items/100

* If the ID is correct and the verb is GET then the result should be all fields of that item like the following :

{

"id": "100",

"product\_code": "NWTCFV-100",

"product\_name": "new\_glass very new1 ",

"photo": "09.png",

"list\_price": "14.00",

"reorder\_level": "10",

"units\_in\_stock": "4",

"category": "sunglasses",

"country": "USA",

"rating": "4.60",

"discontinued": "1",

"date": "2018-08-28 22:52:14"

}

**If the ID is not correct for any reason the API should produces the following response :**

{

"error": "Resource dosn't exist"

}

With Result code: 404

The same response should be retuned if the resource requested is any thing other than items .

* Handle your webservice to allow only 4 HTTPverbs GET, POST, PUT and DELETE . if any HTTP verb is used it should give access code 405 with the following results :

{

"error": "method not allowed!"

}

If the database connection failed for any reason, the response should be with status 500 internal server error and the following custom error .

{

"error": "internal server error!"

}

**That’s all for today , other features will be included in Day03 so keep this example alive it will be continued next lab**

**Implementation notes :**

1. You need the MySQLHandler class in the Resources directory not to loose time working with SQL queries, it has methods for select, update, delete and update so use them , and feel free to change any thing you like .
2. To do this service in an OOP style you need :
   1. Composer for autoloading
   2. A class for handling the requests call it RequestHandler
   3. A class for handling the responses call it ResponseHandler
   4. You will find templates for those two classes with only declarations, feel free to change the design if you want .

**Bonus :**

**Add a logger tool such as monolog to the project via composer and log the request method, resource, resource id, and parameters for each request to your webservice .**