**PHP Assignment**

**General**

* Root url - investing/public/api/..
* In 'API methods' in this article I'm just declaring all the methods names, their requests type, the url for excuting the http request and sometimes remark or small explanation (There are methods which are excuted by same url but their http request is different). Documentiaon to each function exist in the code, here I have detailed only the assential information needed for checking the functionalities.
* All methods are available in the controller folder (investing/app/Http/Controller), the calls to the methods are available in investing/routes/web.php and the model classes are in investing/app.
* There are 3 tables in DB + a table for the bonus (shoppingcarts).

Table 1 – products; Table 2 – catalogs; Table 3 – productscatalogsrelations.

Each catalog contains some products and each product can exist in zero or more catalogs. The connection between the tables are defined with the third table, each record in the third table contains a catalog id and a product id (both PK).

* The queries to create the tables and examples records to mysql are available in other file – 'investing.sql'.
* Every post/put method include validation for the input data. The validations match the DB fields properties.

**Products API**

**Table fields - products:**

* ProductID - PK, FK (to productscatalogsrelations.ProductID)
* ProductName - unique, not null
* ProductDescription - unique, not null
* ProductPrice - not null
* Stock - not null

**API methods:**

**Get methods -**

* getAllProduct - ../products
* getProduct - ../products/{id of product}
* getValue - ../products/{field}/{id of product }

{field} can be - 'name'/'description'/'price'/'stock'.

* getCatalogs - ../products/catalogs/list/{id of product }

All the catalogs which linked to {id of product}

**Post methods -**

* newProduct - ../products?ProductID=?&ProductName=?&ProductDescription=?& ProductPrice=?&Stock=?

Json format -

{

"ProductID": ,

"ProductName": ,

"ProductDescription": ,

"ProductPrice": ,

"Stock":

}

Once we add a new product, obviously it's not linked to any catalog. In order to connect a product to a catalog we will need to do it manually or to use an API relevant method of the catalogs table.

**Put methods -**

* updateProduct –

../products/{id product}?ProductName=' '&ProductDescription=' '&ProductPrice=' '&Stock=' '

Json format -

{

"ProductID": ,

"ProductName": ,

"ProductDescription": ,

"ProductPrice": ,

"Stock":

}

"ProductID" can be or not to be in the json.

* updateField - ../products/{field}/{id of product}/{new value}

{field} can be - 'name'/'description'/'price'/'stock'.

In order to prevent changes in the DB value when using spaces or special characters, use the 'updateProduct' method instead.

**Delete methods -**

* deleteProduct - ../products/{id of product}

**Catalogs API**

**Table fields - catalogs:**

* CatalogID - PK, FK (to productscatalogsrelations.CatalogID)
* CatalogName - unique, not null
* CatalogDescription - unique, not null

**API methods:**

**Get methods -**

* getAllCatalogs - ../catalogs
* getCatalog - ../catalogs/{id of catalog}
* getValue - ../catalogs/{field}/{id of catalog}

{field} can be - 'name'/'description'.

* getProducts - ../catalogs/products/list/{id of catalog}

All the products which linked to {id of catalog}

**Post methods -**

* newCatalog –

../catalogs?CatalogID =' '&CatalogName=' '&CatalogDescription=' '

Json format -

{

"CatalogID": ,

"CatalogName": ,

"CatalogDescription":

}

**Put methods -**

* updateCatalog –

../catalogs/{id of catalog}?CatalogName=' '&CatalogDescription=' '

Json format -

{

"CatalogID ": ,

"CatalogName ": ,

"CatalogDescription ":

}

"CatalogID " can be or not to be in the json.

* updateField - ../catalogs/{field}/{id of catalog}/{new value}

{field} can be - 'name'/'description'.

In order to prevent changes in the DB value when using spaces or special characters, use the 'updateProduct' method instead.

* updateProductsList - ../catalogs/{id of catalog}/{id of product}

Linking products to the relevant catalog.

**Delete methods -**

* deleteCatalog - ../catalogs /{id of catalog}

**Shopping Cart API - Temporary**

**General:**

For saving the temporary shopping cart data I choose to use sessions. The properties of the class are array which contains all the items of the cart and a variable which store the number of different items in the cart. The class properties and the sessions are enough to all the required functionalities.

All the API are used with http post request.

First, I wanted that the methods on the cart (add/update/delete/remove) will modify the amounts in the DB products table (I leaved this coding with uncomment under uncomment 'Updating the amount in DB'). The problem was when the browser is closed or the sessions are deleted, therefore the updating in the DB will be only while the payment will be.

**API methods:**

* get - ../shoppingcart
* delete - ../shoppingcart/delete
* add - ../shoppingcart/newItem?name=x&qty=y

The name is the product name as it is in the DB.

If the user adds amount to an existing item the quantity will be added to the current quantity.

* update - ../shoppingcart/updateItem?name=x&qty=y

The name is the product name as it is in the DB.

It's possible to update quantity only to an item which existing in the cart. The new quantity will replace the current quantity (and will not be added).

* remove - ../shoppingcart/removeItem?name=x

The name is the product name as it is in the DB.

The url contains only the product name (all the quantity will be removed).

* getTotalPrice - ../shoppingcart/totalPrice/{currency}

{currency} can be'USD'/'EUR' and the default prices in 'USD' (also in the DB).

* sortBy - ../shoppingcart/sortBy/{type}

{type} can be 'name'/'price'/'quantity'.

**Shopping Cart API - DB (Bonus)**

**Table fields - shoppingcarts:**

* CartID – PK
* ProductID - PK, FK (to products.ProductID)
* Quantity - not null

**API methods:**

**Get methods -**

* get - ../shoppingcart/{cartId}
* getTotalPrice - ../shoppingcart/{cartId}/totalPrice/{currency}

{currency} can be'USD'/'EUR' and the default prices in 'USD' (also in the DB).

* sortBy - ../shoppingcart/{cartId}/sortBy/{type}

{type} can be 'name'/'price'/'quantity'.

**Post method -**

* add - ../shoppingcart/{cartId}?name=' '&qty=' '

**Put method -**

* update - ../shoppingcart/{cartId}?name=' '&qty=' '

**Delete methods -**

* delete - ../shoppingcart/{cartId}
* remove - ../shoppingcart/{cartId}/removeItem/{productId}