

# UNIVERSITY OF THE WEST INDIES, CAVE HILL CAMPUS

Department of Computer Science, Mathematics & Physics

COMP3415 Database Management Systems 2

Assignment Due Date: **April 14th 2024 Midnight**

## NoSQL Final Project

For this assignment we are developing a POS web application. The frontend is HTML and JavaScript, the backend will be PHP connected to a MongoDB database. You will have to import the **products.json** and the **sales.json** files from the zip file, these json files contain the documents for the **products** collection and the **sales** collection respectively; both collections are to be placed in the **POS** database. You will be provided with the following files in a zip file



- **index.html** – this file does nothing really, it just redirects to the product.html page.
- **product.html** – allow for CRUD of products
- **pos.html** – allow for sales transactions
- **product\_by\_vendor.html** – display products sold by a particular vendor
- **product\_price\_range.html** – display products by price range
- **product\_sold\_on\_date.html** – display products sold on a particular date
- **other\_products\_sold.html** - display the other products sold with a specified product
- **sales\_by\_product.html** – display sales for a particular product
- **total\_sales\_by\_date.html** – display total sales by date
- **best\_selling\_products.html** – display products sold, order in descending order
- **css\mystyle.css** – provides the global style of the web pages
- **js\myScript.js** – provides the global functionality of the web pages
- **php\addSale.php** – this insert sales documents into the Sales collection
- **php\first\_last.php** - this is used to get and store the first and last objectID of the documents in the products collection

For this assignment you will not generally have to write any HTML, CSS or JavaScript code, that is mostly all done, you will be the backend developer on this project which is writing the PHP code to connect to the MongoDB server. For each of the html files above they will be listed below precisely what PHP files are needed.

Considering that the JavaScript code has already been completed it is imperative that you name the files exactly as specified and you name the variables that will be received and returned from the PHP exactly as specified. Failing to use the specified variable names will break the JavaScript code.

The index.html file will simply display the products.html page of the application. No PHP code is required for this screen. The main menu will be visible to the left of every page in this app, an arrow shape will be pointed to the currently visible page.

## product.html

 Product Screen 

Product Code:

Product Name:

Product Cost:

Product Price:

Product OnHand:

Product Vendors: (Comma Separated List)

Save

Cancel

Delete

First

Next

Previous

Last

This is the **Product Screen** page, this page is responsible for all the CRUD of the products in the product collection. There will be a number of PHP files associated with this page.

Specifications of the PHP files for this page:

**getFirst.php** – this file will not receive any values and it will simply return the first document in the product collection. Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
$myObj=new stdClass();
$myObj->_id = (string)$result['_id'];
$myObj->Code = $result['code'];
$myObj->Name = $result['name'];
$myObj->Cost = $result['cost'];
$myObj->Price = $result['price'];
$myObj->Onhand = $result['onhand'];
$myObj->vendors = $result['vendors'];
echo json_encode($myObj);
```

[2.5 Marks]

**getNext.php** – this file receives a variable named **\_id** it uses the value of this variable to query the **Product** collection for the first document that is greater than the said **\_id**. Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
$myObj=new stdClass();
$myObj->_id = (string)$result['_id'];
$myObj->Code = $result['code'];
$myObj->Name = $result['name'];
$myObj->Cost = $result['cost'];
$myObj->Price = $result['price'];
```

```
$myObj->Onhand = $result['onhand'];
$myObj->vendors = $result['vendors'];
echo json_encode($myObj);
```

[2.5 Marks]

**getPrev.php** – this file receives a variable named **\_id** it uses the value of this variable to query the **Product** collection for the first document that is smaller than the said **\_id**. Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
$myObj=new stdClass();
$myObj->_id = (string)$result['_id'];
$myObj->Code = $result['code'];
$myObj->Name = $result['name'];
$myObj->Cost = $result['cost'];
$myObj->Price = $result['price'];
$myObj->Onhand = $result['onhand'];
$myObj->vendors = $result['vendors'];
echo json_encode($myObj);
```

[2.5 Marks]

**getLast.php** – this file will not receive any values and it will simply return the last document in the **Product** collection. Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
$myObj=new stdClass();
$myObj->_id = (string)$result['_id'];
$myObj->Code = $result['code'];
$myObj->Name = $result['name'];
$myObj->Cost = $result['cost'];
$myObj->Price = $result['price'];
$myObj->Onhand = $result['onhand'];
$myObj->vendors = $result['vendors'];
echo json_encode($myObj);
```

[2.5 Marks]

**getProduct.php** – when the user types a product code into the Product Code textbox and leaves the textbox this file is called, it receives a variable named **code** it uses the value of this variable to query the **Product** collection for the product that has a code equals to this value. Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
$myObj=new stdClass();
$myObj->_id = (string)$result['_id'];
$myObj->Code = $result['code'];
$myObj->Name = $result['name'];
$myObj->Cost = $result['cost'];
$myObj->Price = $result['price'];
$myObj->Onhand = $result['onhand'];
$myObj->vendors = $result['vendors'];
echo json_encode($myObj);
```

If there is no matching document simply return the following:

```
$myObj=new stdClass();  
$myObj->Name = '';  
echo json_encode($myObj);
```

 [2.5 Marks]

**getProdName.php** – when the button next to the Product Code textbox is clicked a screen pops up that allows the user to search for a product by name, this is the php file that helps to facilitate that functionality. This file receives a variable named **searchItem** and it searches the product collection for all documents that they **name** field *starts with* the value stored in **searchItem**, the number of returned documents should be limited to 8 and only three fields, **name**, **price** and **code** should be returned. Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return values from the mongo command.

```
foreach($result as $row) {  
    $arr_out[] = $row;  
}  
if (isset($arr_out)){  
    echo json_encode($arr_out);  
}  
else  
{  
    echo json_encode([]);  
}
```

 [2.5 Marks]

**deleteProduct.php** - this file receives a variable named **\_id** it uses the value of this variable to delete the document in the **Product** collection that has a **ObjectID** with the same value. [2.5 Marks]

**addProduct.php** – this file uses the **\$\_REQUEST** associative array sent to it from the product.html frontend, this array is to be inserted into the **Product** collection similar to the code found in the **addClient.php** of Lab6.

However before it is inserted the **cost** and **price** of the array should be converted to **floats**, the **onhand** of the array should be converted to **int**, and the **\$\_REQUEST['vendors']** is to be json decoded before the document is inserted in the Product collection using the following line for code :

```
$_REQUEST['vendors'] = json_decode($_REQUEST['vendors']);
```

 [5 Marks]



**updateProduct.php** – this file uses the **\$\_REQUEST** associative array sent to it from the product.html frontend, this array is to be used to update the **Product** collection similar to the code in the **addProduct.php** from above.


However before the collection is updated with this document the **cost** and **price** of the array should be converted to **floats** and the **onhand** of the array should be converted to **int**, then the product should be updated with the values in the **\$\_RESULT** array base on the following filter:

```
['_id' => new MongoDB\BSON\ObjectID($_RESULT['_id'])]
```

 [5 Marks]

## pos.html

 POS Screen 

Product Code:  

UP/DOWN arrow to go to desired product, LEFT/RIGHT arrow to decrease or increase quantity. DELETE key to remove a product.

Code	Product Name	Unit Price	Qty	Total
524	PN Jerry Curl Wvg 8" – 1	36.95	2	73.90
625	Soft & Dri Power Stripe Passion Flower 2oz	7.99	1	7.99
8645	Cadbury Brunch Bar	2.45	3	7.35
958	Ever Bright Hip Hop Curl	21.49	1	21.49

**\$114.22**

Save Transaction

Clear Transaction

This is the **POS Screen** page, this is used to make the sales of the products. The product sales are inserted into a **Sales** collection of the **POS** database. There are three php files associated with this page.

Specifications of the PHP files for this page:

**getitem.php** – this file is called when the user type a code into the Product Code textbox and click on the enter key, it receives a variable named **code** it uses the value of this variable to query the **Product** collection for the product that has a code equals to this value. Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
if ($result)
{
    $myObj=new stdClass();
    $myObj->Name = $result['name'];
    $myObj->Unit = $result['price'];
    echo json_encode($myObj);
}
else
{
    $myObj=new stdClass();
    $myObj->Name = '';
    echo json_encode($myObj);
}
```

[5 Marks]

**addSale.php** – This file adds sales to the **Sales** collection, I have decided to give the entire code for this file, below:

```
1 <?php
2 require_once "../vendor/autoload.php";
3
4 $collection = (new MongoDB\Client)->POS->sales;
5 $unique = $collection->findOneAndUpdate(
6     [ 'field' => 'unique' ],
7     [ '$inc' => [ 'unique_number' => 1 ] ],
8     [
9         'upsert' => true,
10        'returnDocument' => MongoDB\Operation\FindOneAndUpdate::RETURN_DOCUMENT_AFTER,
11    ]
12);
13 $_REQUEST['salesno'] = $unique->unique_number;
14 $dateTime = new DateTime();
15 $utcDateTime = new MongoDB\BSON\UTCDateTime($dateTime);
16 $_REQUEST['salesdate'] = $utcDateTime;
17 $_REQUEST['items'] = json_decode($_REQUEST['items']);
18 $_REQUEST['salestotal'] = (float)$_REQUEST['salestotal'];
19 $result = $collection->insertOne($_REQUEST);
20 ?>
```

Now this file works just as is, however there is one thing that needs to be done. As it currently stands when the products are sold their quantity on hand is not reduced; **you are required to modify this code to make that happen.** [15 Marks]

I will explain what most of the lines above does so that you will have an idea as to how to go about updating the product **onhand** field when a product is inserted into the **Sales** collection (i.e a product was sold).

**Lines 5 – 12** Each sales transaction will have a unique **salesno**, this mechanism is facilitated by having these lines of code, essentially this inserts (if not present) or update a document at the top of the Sales collection, this document has two fields **field** and **unique\_number** every time this file is called the **unique\_number** field is incremented my 1.

**Line 13** assigned the unique sales number that was generated above to the **\$\_REQUEST['salesno']** of the associative array that was sent from the frontend.

**Line 14 – 16** these lines obtain the current datetime converts it to a mongoDB datetime and assign it to the **\$\_REQUEST['salesdate']** of the associative array that was sent from the frontend.



**Line 17** - the **\$\_REQUEST['items']** contains the actually details of the individual items that are being sold and this item is to be json decoded before the document is inserted in the Sales collection

**Line 18** – converts the **salestotal** to a float before the document is inserted into the collection.

**Line 19** – insert the data sent from the frontend into the collection.

**getProdName.php** – the specification for this file has already been discussed in the product html section above due to the fact that both this page and the product page share this functionality.

## product\_by\_vendor.html

 Product by Vendor 

Vendor Name :

☒ Anywhere ☐ First ☐ Second ☐ Third ☐ Fourth ☐ Fifth ☐ Sixth

This **Product by Vendor** page is to display the products sold by a particular vendor. The **vendors** field in the documents of the product collection is an array of vendor names. If a vendor is the primary vendor for a product their name would be in the first position of this array and their name would appear at a lower position in the array if they are not the primary vendor for the particular product.

Therefore if you wanted to see all the products that a particular vendor is the primary agency for you would select the vendor name and click on the **First** radio button and click on the **Search** button.

There will be three (3) PHP files associated with this page.

Specifications of the PHP files for this page:

**getUniqueVendors.php** - this file will not receive any values and it will simply return the distinct vendors in the **vendors** array of the documents in the **Product** collection. Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
echo json_encode($result);
```

[5 Marks]

**product\_by\_vendor.php** – this file is use to query the **Product** collection for the selected vendor in the specified position in the vendors array. It receives a **\$\_REQUEST** associative array, this array has fields, (vendor, position, page and limit), and these array values should be assigned to the appropriate PHP variables as seen below.

```
$vendorname = $_REQUEST['vendor'];  
$position   = $_REQUEST['position'];  
$page      = (int)$_REQUEST['page'];  
$limit     = (int)$_REQUEST['limit'];  
$skip      = ($page - 1) * $limit; //amount to skip in the database query
```

Now with these variables you are to issue a query on the **Product** collection to return the documents that has the vendor in the specified position of the array, the documents should be sorted by **name**, the documents should skip the number of documents specified in the **\$skip** variable and should be limited to the number specified in the **\$limit** variable.

Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return values from the mongo command.

```
foreach ($result as $entry) {  
    $arr_out[] = $entry;  
}  
if (isset($arr_out)){  
    echo json_encode($arr_out);  
}  
else  
{  
    echo json_encode(array('status'=> false));  
}
```

[5 Marks]

**product\_by\_vendor\_count.php** – this file is used to get the number of documents of the specified criteria above. It receives a `$_REQUEST` associative array, this array has fields, (vendor and position), and these array values should be assigned to the appropriate PHP variables as seen below.

```
$vendorname = $_REQUEST['vendor'];  
$position = $_REQUEST['position'];
```

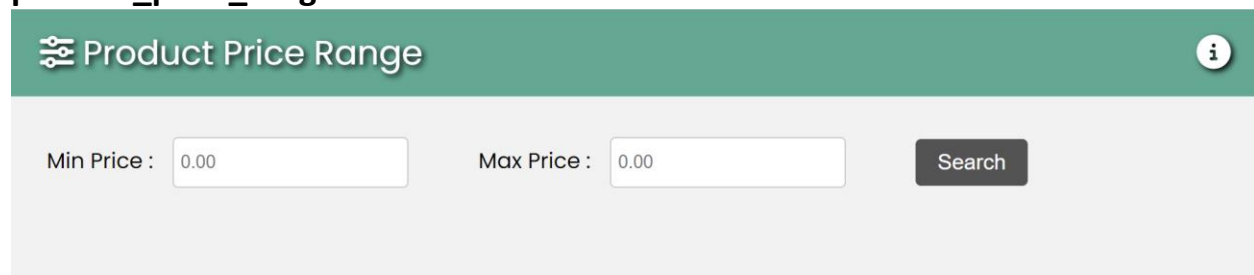
Now with these variables you are to issue a query on the **Product** collection to return the count of the documents that has the vendor in the specified position of the array, the results should be stored in a variable called **\$result**.

Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
echo json_encode(array('count'=> $result));
```

[5 Marks]

## product\_price\_range.html



This **Product Price Range** page is to display all the products that price falls between the **Min Price** and **Max Price**. There will be two (2) PHP files associated with this page.

Specifications of the PHP files for this page:

**product\_price\_range.php** - this file is use to query the **Product** collection to retrieve the products that has a price between the minimum and maximum that is specified in the front end. It receives a



\$\_REQUEST associative array, this array has fields, (min, max, page and limit), and these array values should be assigned to the appropriate PHP variables as seen below.

```
$min = (float)$_REQUEST['min'];
$max = (float)$_REQUEST['max'];
$page = isset($_REQUEST['page']) ? (int)$_REQUEST['page'] : 1;
$limit = (int)$_REQUEST['limit'];
$skip = ($page - 1) * $limit; //amount to skip in the database query
```

Now with these variables you are to issue a query on the **Product** collection to retrieve the documents that **price** falls between the **\$min** and **\$max**, the documents should skip the number of documents specified in the **\$skip** variable and should be limited to the number specified in the **\$limit** variable, also the documents should be sorted by the **price** field.

Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return values from the mongo command.

```
foreach ($result as $entry) {
    $entry['cost'] = number_format($entry['cost'],2);
    $entry['price'] = number_format($entry['price'],2);
    $arr_out[] = $entry;
}
if (isset($arr_out)){
    echo json_encode($arr_out);
}
else
{
    echo json_encode(array('status'=> false));
}
```

[5 Marks]

**product\_price\_range\_count.php** – this file is used to get the number of documents of the specified criteria above. It receives a \$\_REQUEST associative array, this array has fields, (min and max), and these array values should be assigned to the appropriate PHP variables as seen below:

```
$min = (float)$_REQUEST['min'];
$max = (float)$_REQUEST['max'];
```



Now given these variables you are to issue a query on the **Product** collection to return the count of the documents that the field **price** is greater than or equal to **\$min** and less than or equal to **\$max**, the results should be stored in a variable called **\$result**.

Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
echo json_encode(array('count'=> $result));
```

[5 Marks]

## sales\_by\_product.html

 Sales for Product Code 

Product Code

Hair Pins Page 1 of 18

Sales Date	Amount Sold	Sales Total
April 25, 2019	2	2.00
February 18, 2019	2	2.00

The **Sales for Product Code** page is used to display the list of sales date, amount sold and sales total of the product code that is typed into the product code textbox.

There will be two (2) PHP files associated with this page.

Specifications of the PHP files for this page:

**sales\_by\_product.php** - this file is use to query the **Sales** collection for the amount of a particular product sold by dates. It receives a `$_REQUEST` associative array, this array has fields, (pcode, page and limit), and these array values should be assigned to the appropriate PHP variables as seen below.

```
$code = $_REQUEST['pcode'];  
$page = isset($_REQUEST['page']) ? (int)$_REQUEST['page'] : 1;  
$limit = (int)$_REQUEST['limit'];  
$skip = ($page - 1) * $limit;
```

It uses the mongo aggregate function to group the documents by date. In the grouping stage in addition to the having the `_id` field, there should also be a **date**, **subtotal** and a **count** field. The `_id` field here should take a similar form to the `_id` field found at the bottom of page 10 in **Lab3**, this will insure that we are grouping by a specific day of a specific year. Considering that in the **Sales** collection the products sold are in an array called **items** then it would mean that you would have to **\$unwind** the array and then **\$match** for **items.ProdID** equals to **\$code**, before you move on to the **\$group** stage.

The documents should skip the number of documents specified in the **\$skip** variable and should be limited to the number specified in the **\$limit** variable, also the documents should be sorted by the **date** field.

Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return values from the mongo aggregate command.

```
foreach ($result as $entry) {  
    $entry['date'] = getDOB($entry['date']);  
}
```

```

        $entry['subTotal'] = number_format($entry['subTotal'],2);
        $arr_out[] = $entry;
    }
    if (isset($arr_out)){
        echo json_encode($arr_out);
    }
    else{
        echo json_encode(array('status'=> false));
    }
}

```

In the above code you would notice the use of a function called **getDOB()** this function is used simply to change the mongo date to a human readable date. So you would have to have the definition for this function located somewhere in this file before the function is called. The definition for this function can be copied from the **browse.php** file of **Lab6**. [10 Marks]

**sales\_by\_product\_count.php** - this file is used to get the number of documents of the specified criteria above. It receives a `$_REQUEST` associative array, this array has a single field, (pcode), and this array value should be assigned to the appropriate PHP variables as seen below:

```
$pcode = $_REQUEST['pcode'];
```

To get the number of documents, you again have group by date, however before the grouping stage you must **\$unwind** the items array and **\$match** the **items.ProdID** equal to **\$pcode**. Next you must add a **\$count** stage that would be the following code:

```

[
    '$count' => 'total'
]

```

Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```

foreach ($result as $entry) {
    echo json_encode($entry);
}

```



[10 Marks]



## Note on querying a datetime field in MongoDB

When a date is saved in a MongoDB using the `$utcDateTime` as seen in the `addSale.php` file, the `saleDate` field will not only have a date such as 2023-10-23 it will also have a time associated with it, such 2023-10-20 09:45:23. Therefore if we want to query for a particular date we must take into consideration that they will have times ranging from 00:00:00 to 23:59:59 for any particular date.



For this reason you will see in the php files that are doing searches by dates, I have added a day to the ending date of the search and rather than doing a match with a `$eq` for the date, the search is should done with `$gte` of the start date and `$lt` the next day (this will include all possible times for the date).

## total\_sales\_by\_date.html

 Total sales by date 

Start Date:   Finish Date:  

Page 1 of 6

Sales Date	Sales Total	No. of Trans.	Avg per Trans.	View
March 5, 2021	73.94	1	73.94	
March 2, 2021	269.36	4	67.34	

The **Total sales by date** page is used to display the total sales and number of transactions per date, for the specified start and finish date. There will be three (3) PHP files associated with this page.

Specifications of the PHP files for this page:

**total\_sales\_by\_date.php** - this file is use to query the **Sales** collection for the total sales by dates. It receives a `$_REQUEST` associative array, this array has fields, (start, finish, page and limit), and these array values should be assigned to the appropriate PHP variables as seen below.

```
$startD = $_REQUEST['start'];
$finishD = $_REQUEST['finish'];

$date = new DateTime($finishD);
$date->modify('+1 day');
$finishD = $date->format('Y-m-d');

$startD=new MongoDB\BSON\UTCDateTime(new DateTimeImmutable($startD));
$finishD=new MongoDB\BSON\UTCDateTime(new DateTimeImmutable($finishD));

$page = isset($_REQUEST['page']) ? (int)$_REQUEST['page'] : 1;
$limit = (int)$_REQUEST['limit'];
$skip = ($page - 1) * $limit;
```

It uses the mongo aggregate function to **\$match** the documents by date. In the grouping stage in addition to the having the **\_id** field, there should also be a **date**, **amt**, **avgAmt** and **counter** field. The **\_id** field here should take a similar form to the **\_id** field found at the bottom of page 10 in **Lab3**, this will insure that we are grouping by a specific day of a specific year. The **date** field contains the sales date, the **amt** contains the sum of the **\$salestotal** for this grouping, **avgAmt** contains the average of the **\$salestotal** and **counter** contains the amount of documents in this grouping.

The documents should skip the number of documents specified in the **\$skip** variable and should be limited to the number specified in the **\$limit** variable, also the documents should be sorted by the **date** field.

Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return values from the mongo aggregate command.

```

foreach ($result as $entry) {
    $entry['date'] = getDOB($entry['date']);
    $entry['amt'] = number_format($entry['amt'],2);
    $entry['avgAmt'] = number_format($entry['avgAmt'],2);
    $arr_out[] = $entry;
}
if (isset($arr_out)){
    echo json_encode($arr_out);
}
else
{
    echo json_encode(array('status'=> false));
}

```

In the above code you would notice the use of a function called **getDOB()** this function is used simply to change the mongo date to a human readable date. So you would have to have the definition for this function located somewhere in this file before the function is called. The definition for this function can be copied from the **browse.php** file of **Lab6**. [10 Marks]

**total\_sales\_by\_date\_count.php** - this file is used to get the number of documents of the specified criteria above. It receives a `$_REQUEST` associative array, this array has fields, (start and finish), and these array values should be assigned to the appropriate PHP variables as seen below.

```

$startD = $_REQUEST['start'];
$finishD = $_REQUEST['finish'];

$date = new DateTime($finishD);
$date->modify('+1 day');
$finishD = $date->format('Y-m-d');

$startD=new MongoDB\BSON\UTCDateTime(new DateTimeImmutable($startD));
$finishD=new MongoDB\BSON\UTCDateTime(new DateTimeImmutable($finishD));

```

To get the number of documents, you again have to group by date, however before the grouping stage you must **\$match** the sales dates greater than **\$startD** and less than **\$finishD**. Next you must add a **\$count** stage that would be the following code:

```

[
    '$count' => 'total'
]

```

Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```

foreach ($result as $entry) {
    echo json_encode($entry);
}

```

[10 Marks]

**view\_transactions\_details\_by\_date.php** – this file is use to query the **Sales** collection for the items sold by date. It receives a \$\_REQUEST associative array, this array has a single field, (date), and this array values should be assigned to the appropriate PHP variables as seen below:

```
$sdate = $_REQUEST['date'];
$date = new DateTime($sdate);
$date->modify('+1 day');
$fdate = $date->format('Y-m-d');
$sdate=new MongoDB\BSON\UTCDateTime(new DateTimeImmutable($sdate));
$fdate=new MongoDB\BSON\UTCDateTime(new DateTimeImmutable($fdate));
```



It uses the mongo aggregate function where you are to match the **saledate** greater than or equal to \$sdate and less than \$fdate, next you are to \$unwind the \$items finally \$sort by **salesno** by descending order.


Below shows the makeup of the JSON object that is to be returned from this page. Note **\$result** is the variable name that stores the return value from the mongo command.

```
foreach ($result as $entry) {
    $arr_out[] = $entry;
}
if (isset($arr_out)){
    echo json_encode($arr_out);
}
else
{
    echo json_encode(array('status'=> false));
}
```

[10 Marks]

## product\_sold\_on\_date.html


**Product Sold on Date**


Date of Sales:  

Product Code	Product Name	Unit Price	Amt Sold	SubTotal
756	Bantu Relaxer Regular 15oz	11.99	1	11.99
564	Speed Stick Regular 3.25oz	9.89	1	9.89

The **Product Sold by Date** page is used to display the individual products sold on a particular date, for the specified start and finish date. There will be one (1) PHP file associated with this page.

Specifications of the PHP files for this page:

**product\_sold\_on\_date.php** - this file is use to query the **Sales** collection for the items sold on a specific date. It receives a \$\_REQUEST associative array, this array has a single field, (soldDate), and this array value should be assigned to the appropriate PHP variables as seen below:

```
$soldDate = $_REQUEST['soldDate'];
$date = new DateTime($soldDate);
$date->modify('+1 day');
$soldDateNext = $date->format('Y-m-d');
$soldDate=new MongoDB\BSON\UTCDateTime(new DateTimeImmutable($soldDate));
$soldDateNext=new MongoDB\BSON\UTCDateTime(new
DateTimeImmutable($soldDateNext));
```

It uses the mongo aggregate function where you are to match the **saledate** greater than or equal to \$soldDate and less than \$soldDateNext, next you are to \$unwind the \$items, next you are to \$group the documents. In the grouping stage in addition to the having the **\_id** field, there should also be a **ProdID, ProdName, Unit, AmtSold** and **SubTotal** fields.

The **\_id** field for the group should hold the items ProdID, the **ProdID** field should hold the first ProdID of the group, the **ProdName** field should hold the first ProdName of the group, the **Unit** field should hold the first UnitPrice of the group, the **AmtSold** is the sum of the AmtSold in the group, the **SubTotal** field should hold the sum of the AmtSold \* UnitPrice.

Below shows the makeup of the JSON object that is to be returned from this file. Note **\$result** is the variable name that stores the return values from the mongo aggregate command.

```
foreach ($result as $entry) {
    $entry['SubTotal'] = number_format($entry['SubTotal'],2);
    $arr_out[] = $entry;
}
if (isset($arr_out)){
    echo json_encode($arr_out);
}
else{
    echo json_encode(array('status'=> false));
}
```

[10 Marks]

**best\_selling\_products.html**

Product Code	Product Name	Unit Price	Amount Sold
3641	Fashion Earring Bob	1.00	4820
18550	CCC Mints Single	0.25	4165

The **Best sellers** page is used to display the total amount sold of the individual products, sorted by the product that sold the most. There will be two (2) PHP files associated with this page.

Specifications of the PHP files for this page:

**best\_sellers.php** - this file is use to query the **Sales** collection for the total sales by product code. It receives a `$_REQUEST` associative array, this array has a single field, (limit), and this array value should be assigned to the appropriate PHP variable as seen below.

```
$limit = (int)$_REQUEST['limit'];
```

It uses the mongo aggregate function to group the documents by product code. In the grouping stage in addition to the having the **\_id** field, there should also be a **pname**, **pcode**, **unit** and **totalAmtSold** field. The **pname** field contains the product name, the **pcode** field holds the product ID, the **unit** field holds the product unit price, the **totalAmtSold** holds the sum of the **\$items.AmtSold** for this grouping.

Considering that in the **Sales** collection the products sold are in an array called **items** then it means that you would have to **\$unwind** the array before you move on to the **\$group** stage.

The documents should be limited to the number specified in the **\$limit** variable, also the documents should be sorted by the **totalAmtSold** field.

Below shows the makeup of the JSON object that is to be returned from this file. Note **\$result** is the variable name that stores the return values from the mongo aggregate command.

```
foreach ($result as $entry) {
    $entry['unit'] = number_format($entry['unit'],2);
    $arr_out[] = $entry;
}
if (isset($arr_out)){
    echo json_encode($arr_out);
}
else{
    echo json_encode(array('status'=> false));
}
```

[10 Marks]

**other\_products\_sold.html**



## ➡ Other Products Sold With...



Product Code

Search

Speed Stick Regular 3.25oz (1)

Product Code	Product Name	Amount Sale
897	Bio Claire Serum 60ml	1
898	Looks Metal Cushion Wig Brush Large	1

The **Other Products Sold With** page is used to display the products that were sold with the product code that is specified in the textbox. There will be one PHP file associated with this page.

Specifications of the PHP file for this page:

**other\_products\_sold.php** – this file receives a `$_REQUEST` associative array, this array has a single field, (pcode), and this array value should be assigned to the appropriate PHP variable as seen below.

```
$pcode = $_REQUEST['pcode'];
```

It uses the mongo aggregate function where you are to `$match` for documents in the **Sales** collection that has an **item** array with `ProdID` equals to the `$pcode`, next you are to `$unwind` the items arrays in these documents, next you are to `$group` the documents. In the grouping stage in addition to the having the `_id` field, there should also be a **ProdName**, **amtSold** fields.

The `_id` field for the group should hold the items `ProdID`, the **ProdName** field should hold the first `ProdName` of the group, the **amtSold** is the sum of the `AmtSold` in the group.

The documents should be limited to 20, also the documents should be sorted by the **amtSold** field in descending order.

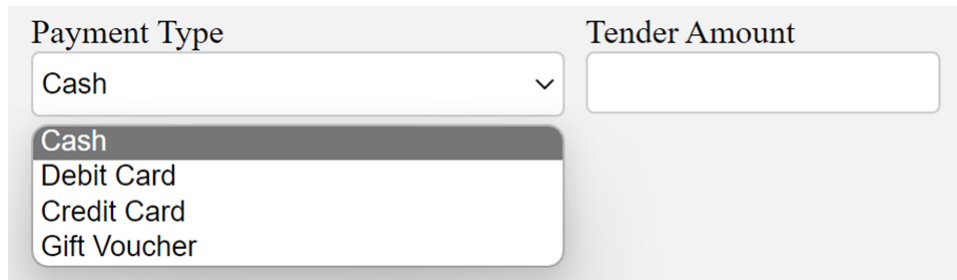
Below shows the makeup of the JSON object that is to be returned from this file. Note `$result` is the variable name that stores the return values from the mongo aggregate command.

```
foreach ($result as $entry) {  
    $arr_out[] = $entry;  
}  
if (isset($arr_out)){  
    echo json_encode($arr_out);  
}  
else{  
    echo json_encode(array('status'=> false));  
}
```

[10 Marks]

## The Final Task

You are to modify the **pos.html** and the **addSale.php** file so as to add a payment facility to the program, this facility should look like the image below, where there is a select box for the payment type and an input box for the amount of money tendered. When the user enters this data it should be placed in the current transaction that is being saved.



Payment Type	Tender Amount
<div>Cash</div> <div>Cash</div> <div>Debit Card</div> <div>Credit Card</div> <div>Gift Voucher</div>	<input type="text"/>

[20 Marks]

**The End**