

UNIVERSITY OF THE WEST INDIES, CAVE HILL CAMPUS

Department of Computer Science, Mathematics & Physics

COMP3415 Database Management Systems 2

Assignment Due Date: April 9th 2021 Midnight

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 are required to watch the video posted on eLearning so you can see how the web application looks and functions when finished. You will have to import the **products.json** and the **sales.json** files from the zip file on eLearning these json files contain the documents for the **Product** 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** - main menu of the website
- **product.html** – allow for CRUD of products
- **POS.html** – allow for sales transactions
- **product_by_vendor_try.html** – display products sold by a particular vendor
- **product_price_range_try.html** – display products by price range
- **sales_by_product_try.html** – display sales for a particular product
- **total_sales_by_date_try.html** – display total sales by date
- **best_sellers.html** – display products sold, order in descending order
- **mystyle.css** – provides the global style of the web pages
- **myScript.js** – provides the global functionality of the web pages
- **addSale.php** – this insert sales documents into the Sales collection

For this assignment you will not have to write any HTML, CSS or JavaScript code, that is 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.

index.html

Chris Smith
20002996
Database Management Systems 2

Product Data Entry Screen

Point of Sales Screen

Product by Vendor Search Screen

Product Price Range Search Screen

Sales of a specific product

Total sales by date

Best Sellers


University of the West Indies
NoSQL Database Project
Built by Chris Smith

The index.html file will simply display the menu to the various pages of the application. No PHP code is required for this screen.

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.html** 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);
```

[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);

```

[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);

```

[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);

```

[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 **m_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);
```

[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 **m_code** and it searches the product collection for all documents that their **name** field starts with the value stored in **m_code**, 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);  
    return;  
}  
else  
{  
    echo "X";  
    return;  
}
```

[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.

[5 Marks]

addProduct.php – this file receives a JSON object named **product**, this object is to be stored in a PHP variable and that variable should be converted to a PHP associative array using the following commands:

```
$product = $_POST['product'];  
$decodedProduct = json_decode($product, true);
```

Next the **cost** and **price** of the array should be converted to **floats** and the **onhand** of the array should be converted to **int**, then insert this array into the **Product** collection.

[5 Marks]

updateProduct.php - this file receives a JSON object named **product**, this object is to be stored in a PHP variable and that variable should be converted to a PHP associative array using the following commands:

```
$product = $_POST['product'];  
$decodedProduct = json_decode($product, true);
```

Next 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 \$decodedProduct array base on the following filter:

```
[ '_id' => new MongoDB\BSON\ObjectId($decodedProduct['_id']) ]
```

 [10 Marks]

POS.html

<< POS Screen

Product Code:

UP/DOWN arrow to go to desired product, LEFT/RIGHT arrow to decrease or increase quantity

Code	Product Name	Unit	Qty	Total
524	PN Jerry Curl Wvg 8" - 1	36.95	1	36.95
625	Soft & Dri Power Stripe Passion Flower 2oz	7.99	1	7.99
8645	Cadbury Brunch Bar	2.45	1	2.45
958	Ever Bright Hip Hop Curl	21.49	1	21.49
635	P. LaTouche Body & Hand Lotion 2oz	3.49	1	3.49

\$83.36

Save Transaction Clear Transaction

This is the **POS** 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 **m_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 $collection = (new MongoDB\Client)->POS->Sales;
4 $unique = $collection->findOneAndUpdate(
5     [ 'field' => 'unique' ],
6     [ '$inc' => [ 'unique_number' => 1 ] ],
7     [
8         'upsert' => true,
9         'returnDocument' => MongoDB\Operation\FindOneAndUpdate::RETURN_DOCUMENT_AFTER,
10    ]
11 );
12 //Retrieve the string, which was sent via the POST parameter "user"
13 $sale = $_POST['sale'];
14 //Decode the JSON string and convert it into a PHP associative array.
15 $decodedSale = json_decode($sale, true);
16 $decodedSale['salesno'] = $unique->unique_number;
17 $decodedSale['salesdate'] = new \MongoDB\BSON\UTCDateTime(time()*1000);
18 for ($x = 0; $x < sizeof($decodedSale['items']); $x++) {
19     $decodedSale['items'][$x]['AmtSold'] = (int)$decodedSale['items'][$x]['AmtSold'];
20     $decodedSale['items'][$x]['UnitPrice'] = (float)$decodedSale['items'][$x]['UnitPrice'];
21 }
22 $decodedSale['salestotal'] = (float)$decodedSale['salestotal'];
23 $collection->insertOne($decodedSale);
24 echo $decodedSale['salesno'];
25 ?>

```

Now this file works just as is, however there is one thing that needs to be done. As it currently stands when a product is sold its quantity on hand is not reduced; **you are required to modify this code to make that happen.** [10 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).

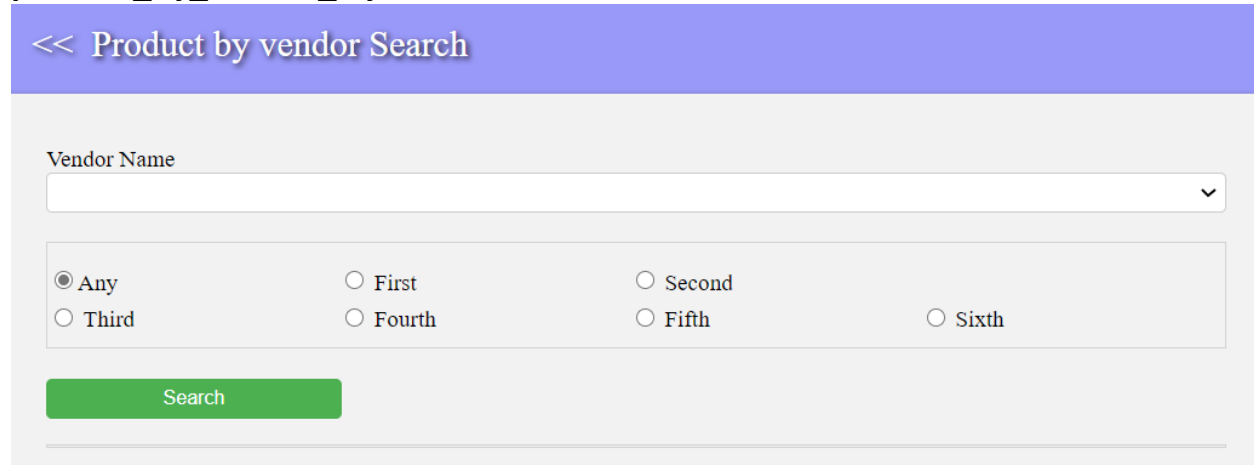
Lines 4 – 11 Each sales transaction will have a unique **salesno**, this mechanism is facilitated by having these lines of code, essentially this inserts (if not present) or update a document at the top of the Sales collection, this document has two fields **field** and **unique_number** every time this file is called the **unique_number** field is incremented my 1.

Line 13 – 15 these lines receive the data sent from the front end that contains the sales data, and convert it to a PHP associative array.

Line 18 – 21 these lines loop through the individual products that are being sold and change the data types to **int** and **float** for the **AmtSold** and **UnitPrice** field respectively.

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_try.html



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:

product_by_vendor_try.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 JSON object that is to be convert to a PHP associative array, the JSON object name is **ProVen** the below code converts that JSON object and places them in appropriate PHP variables.

```
$ProdVen = $_REQUEST['ProdVen'];
$ProdVen = json_decode($ProdVen, true);
//appropriate variables
$vendorname = $ProdVen['vendor'];
$position = $ProdVen['position'];
$page = isset($ProdVen['page']) ? (int)$ProdVen['page'] : 1;
$limit = (int)$ProdVen['limit'];
$skip = ($page - 1) * $limit; //amount of documents to skip
```

Now given 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 be limited to the number specified in the **\$limit** variable and should skip the number of documents specified in the **\$skip** 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 "";
}
```

[5 Marks]

product_by_vendor_try_count.php – this file is used to get the number of documents of the specified criteria above. It receives a JSON object that is to be convert to a PHP associative array, the JSON object name is **ProVen** the below code converts that JSON object and places them in appropriate PHP variables.

```
$ProdVen = $_REQUEST['ProdVen'];
$ProdVen = json_decode($ProdVen, true);
//appropriate variables
$vendorname = $ProdVen['vendor'];
$position = $ProdVen['position'];
```

Now given 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($result);
```

[5 Marks]

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_price_range_try.html

<< Product Price Range Search

Min Price	<input type="text"/>	Max Price	<input type="text"/>
<input type="button" value="Search"/>			

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_try.php - this file is use to query the **Product** collection to retrieve the products that price is between the minimum and maximum that is specified in the front end. It receives a JSON object that is to be convert to a PHP associative array, the JSON object name is **ProdPriceRange**, the below code converts that JSON object and places them in appropriate PHP variables.

```
$ProdPriceRange = $_REQUEST['ProdPriceRange'];
$ProdPriceRange = json_decode($ProdPriceRange, true);
//appropriate variables
$min = (float)$ProdPriceRange['min'];
$max = (float)$ProdPriceRange['max'];
$page = isset($ProdPriceRange['page']) ? (int)$ProdPriceRange['page'] : 1;
$limit = (int)$ProdPriceRange['limit'];
$skip = ($page - 1) * $limit; //amount of documents to skip
```

Now given 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 be limited to the number specified in the **\$limit** variable and should skip the number of documents specified in the **\$skip** 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 "";
    }

```

[5 Marks]

product_price_range_try_count.php – this file is used to get the number of documents of the specified criteria above. It receives a JSON object that is to be convert to a PHP associative array, the JSON object name is **ProdPriceRange** the below code converts that JSON object and places them in appropriate PHP variables.

```

$ProdPriceRange = $_REQUEST['ProdPriceRange'];
$ProdPriceRange = json_decode($ProdPriceRange, true);
//appropriate variables
$min = (float)$ProdPriceRange['min'];
$max = (float)$ProdPriceRange['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($result);
```

[5 Marks]

sales_by_product_try.html

<<

Sales for Product Code

Product Code

Hair Pins

Sales Date	Amt Sold	Sales Total

The **sales by product** 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_try.php - this file is use to query the **Sales** collection for the amount of a particular product sold by dates. It receives a JSON object that is to be convert to a PHP associative array, the JSON object name is **SaleByProd**, the below code converts that JSON object and places them in appropriate PHP variables.

```
$SaleByProd = $_REQUEST['SaleByProd'];
$SaleByProd = json_decode($SaleByProd, true);
//appropriate variables
$code = $SaleByProd['pcode'];
$page = isset($SaleByProd['page']) ? (int)$SaleByProd['page'] : 1;
$limit = (int)$SaleByProd['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. 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 be limited to the number specified in the **\$limit** variable and should skip the number of documents specified in the **\$skip** 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 "";
}
```

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_try_count.php - this file is used to get the number of documents of the specified criteria above. It receives a JSON object that is to be convert to a PHP associative array, the JSON object

name is **SaleByProd** the below code converts that JSON object and places them in appropriate PHP variables.

```
$SaleByProd = $_REQUEST['SaleByProd'];  
$SaleByProd = json_decode($SaleByProd, true);  
$pcode = $SaleByProd['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]

total_sales_by_date_try.html

<< Total sales by date

Start Date: 01/01/2017

Finish Date: 31/12/2017

Search

Sales Date	Sales Total	No. of Trans.	Avg per Trans.
------------	-------------	---------------	----------------

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 two (2) PHP files associated with this page.

Specifications of the PHP files for this page:

total_sales_by_product_try.php - this file is use to query the **Sales** collection for the total sales by dates. It receives a JSON object that is to be convert to a PHP associative array, the JSON object name is **SaleByDate**, the below code converts this JSON object and places them in appropriate PHP variables.

```
$SaleByDate = $_REQUEST['SaleByDate'];
$SaleByDate = json_decode($SaleByDate, true);
$startD = $SaleByDate['start'];
$finishD = $SaleByDate['finish'];
//appropriate variables
$page = isset($SaleByDate['page']) ? (int)$SaleByDate['page'] : 1;
$limit = (int)$SaleByDate['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**, **amt** and **avgAmt** field. The **date** field contains the sales date, the **amt** contains the sum of the **\$salestotal** for this grouping and **avgAmt** contains the average of the **\$salestotal** for this grouping.

The documents should be limited to the number specified in the **\$limit** variable and should skip the number of documents specified in the **\$skip** variable, also the documents should be sorted by the **_id** 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 "";
}
```

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_try_count.php - this file is used to get the number of documents of the specified criteria above. It receives a JSON object that is to be convert to a PHP associative array, the JSON object name is **SaleByDate** the below code converts that JSON object and places them in appropriate PHP variables.

```

$SaleByDate = $_REQUEST['SaleByDate'];
$SaleByDate = json_decode($SaleByDate, true);
$startD = $SaleByDate['start'];
$finishD = $SaleByDate['finish'];

```

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]

best_sellers.html

<< Best Sellers

Product Code	Product Name	Total Amount Sold
3641	Fashion Earring Bob	4820
18550	CCC Mints Single	4165

The **best sellers** page is used to display the total amount sold of the individual products, sort 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 JSON object that is to be convert to a PHP associative array, the JSON object name is **BestSell**, the below code converts that JSON object and places them in appropriate PHP variables.

```

$BestSell = $_REQUEST['BestSell'];
$BestSell = json_decode($BestSell, true);
//appropriate variables
$page = isset($BestSell['page']) ? (int)$BestSell['page'] : 1;

```

```
$limit = (int)$BestSell['limit'];
$skip  = ($page - 1) * $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** and **totalAmtSold** field. The **pname** field contains the product name, the **totalAmtSold** contains 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 would mean 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 and should skip the number of documents specified in the **\$skip** 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 page. 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 "";
}
```

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]

best_sellers_count.php - this file is used to get the number of documents of the specified criteria above. It does not receives any JSON object. To get the number of documents you are to use the mongo aggregate function, you have group by **\$items.ProdID**, however before the grouping stage you must **\$unwind** the **items** array. 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]