**Lab # 18 – Supplemental Information**

**Understanding the Requirements**

The application here is a web-based shopping cart that uses PHP’s built-in session management support to track the items selected for purchase by a user. Items are listed in a product catalog, and the user has the ability to select custom quantities of each item using an HTML form. The selected items then appear in the user’s “cart,” with item subtotals automatically calculated from the quantity and unit price. Users can clear their carts of all selected items, or selectively update the quantities to be purchased of each item; the totals are recalculated automatically. The catalog itself is read from a text file; this file contains a list of product IDs, descriptions, and unit prices.

If all this seems somewhat daunting, fear not — it’s pretty simple, once you break it down.

**Retrieving Catalog Data**

***Let’s begin with the catalog file and examine the format in which catalog data is stored:***

101:AA batteries (pack of 2):2.99

102:AA batteries (pack of 4):5.49

103:Backpack (black): 69.99

104:Money belt with 6 compartments (black):13.49

105:Haversack (red):199.99

106:Swiss Army knife (6 blades including can opener and scissors):24.99

107:Duffel bag (steel gray):28.50

This is fairly easy to understand: each product is listed on a separate line, with colons used to demarcate the product code or SKU, its description, and its price. It’s easy to parse this file and store its contents in a PHP array using the file() and explode() functions. And this next snippet of code does exactly that:

<?php

// look for catalog file

$catalogFile = "catalog.dat";

// file is available, extract data from it

// place into $CATALOG array, with SKU as key

if (file\_exists($catalogFile))

{

$data = file($catalogFile);

foreach ($data as $line)

{

$lineArray = explode(':', $line);

$sku = trim($lineArray[0]);

$CATALOG[$sku]['desc'] = trim($lineArray[1]);

$CATALOG[$sku]['price'] = trim($lineArray[2]);

}

}

else

{

die("Could not find catalog file");

}

?>

The end result of this is an associative array called **$CATALOG**, which uses the product codes as keys. Each key further points to a nested associative array with two keys — **desc** and **price** — which the product’s description and price, respectively. This **$CATALOG** array, once created, becomes available for use by other components within the script. Obviously, in the event that the catalog file cannot be found, the user must be notified with an error message, hence, the ***if(file\_exists(...))*** test and subsequent call to ***die()*** if the test proves false.

Once the catalog data is successfully imported into a PHP variable, the next step is to print it. Because the data is in an array, it’s logical to reach for the ***foreach()*** loop to process it.

***Here’s the code:***

<table border="0" cellspacing="10">

<?php

// print items from the catalog for selection

foreach ($CATALOG as $k => $v)

{

echo "<tr><td colspan=2>";

echo "<b>" . $v['desc'] . "</b>";

echo "</td></tr>\n";

echo "<tr><td>";

echo "Price per unit: " . $CATALOG[$k]['price'];

echo "</td><td>Quantity: ";

echo "<input size=4 type=text name=\"a\_qty[" . $k . "]\">";

echo "</td></tr>\n";

}

?>

<tr>

<td colspan="2">

<input type="submit" name="add" value="Add items to cart">

</td>

</tr>

</table>

Notice that each item in the product catalog contains an empty text field next to it, which can be used to input quantities. The data entered into these fields is submitted back to the same script, by means of a **POST**-ed array called **$a\_qty**. The keys of this array are the product codes, and its values are the corresponding quantities selected.

**Creating the Shopping Cart**

On submission, the items and quantities selected need to find their way into the “shopping cart” — essentially, a session variable that remains available throughout the user’s session. This shopping cart is an associative array called **$\_SESSION['cart']**. Its keys are the product codes of the selected items, and its values are the corresponding quantities entered by the user.

<?php

session\_start();

if ($\_POST['add'])

{

foreach ($\_POST['a\_qty'] as $k => $v)

{

$\_SESSION['cart'][$k] = $\_SESSION['cart'][$k] + $v;

}

}

?>

Note that for items already in the cart, submitting the form with new numbers adds to the existing quantities, instead of replacing them.

**Calculating Costs**

Once items have been stored in the shopping cart, it’s a simple matter to display them. All you need to do is iterate over the **$\_SESSION['cart']** array and print its values. Because **$\_SESSION['cart']** only stores product codes with quantities, it’s necessary to cross-reference the product codes with the data in the **$CATALOG** array to retrieve the human-readable descriptions and prices (these prices are also used to calculate subtotals and the grand total).

<table width="100%" border="0" cellspacing="10">

<?php

// initialize a variable to hold total cost

$total = 0;

// check the shopping cart

// if it contains values

// look up the SKUs in the $CATALOG array

// get the cost and calculate subtotals and totals

if (is\_array($\_SESSION['cart']))

{

foreach ($\_SESSION['cart'] as $k => $v)

{

if ($v > 0)

{

$subtotal = $v \* $CATALOG[$k]['price'];

$total += $subtotal;

echo "<tr><td>";

echo "<b>$v unit(s) of " . $CATALOG[$k]['desc'] ↵

. "</b>";

echo "</td><td>";

echo "New quantity: <input size=4 type=text ↵

name=\"u\_qty[" . $k . "]\">";

echo "</td></tr>\n";

echo "<tr><td>";

echo "Price per unit: " . $CATALOG[$k]['price'];

echo "</td><td>";

echo "Sub-total: " . sprintf("%0.2f", $subtotal);

echo "</td></tr>\n";

}

}

}

?>

<tr>

<td><b>TOTAL</b></td>

<td><b><?=sprintf("%0.2f", $total)?></b></td>

</tr>

<tr>

<td><input type="submit" name="update" value="Update Cart"></td>

<td><input type="submit" name="clear" value="Clear Cart"></td>

</tr>

</table>

**Handling Cart Updates**

This display contains a text field next to each item, for the user to update the quantities of each item in the cart. Values are submitted to the form processor through the **$u\_qty** array (similar in structure to the **$a\_qty** array explained earlier). This update operation differs from the add operation in that submitting the form with new values replaces the existing quantities (instead of adding to them). The user also has the option of “emptying” the cart with a single click; essentially, this destroys the session data and presents the user with an empty **$\_SESSION['cart']** array.

***Here’s the code to perform the previous logic:***

<?php

if ($\_POST['update'])

foreach ($\_POST['u\_qty'] as $k => $v)

{

$\_SESSION['cart'][$k] = $v;

}

}

// if this is a clear operation

// reset the session and the cart

// destroy all session data

if ($\_POST['clear'])

{

$\_SESSION = array();

session\_destroy();

}

?>

**Putting It All Together**

***And now that you’ve seen how the various pieces interact with each other, here’s the complete script:***

<?php

// start session

session\_start();

// initialize session shopping cart

if (!isset($\_SESSION['cart']))

{

$\_SESSION['cart'] = array();

}

// initialize session shopping cart

if (!isset($\_SESSION['cart']))

{

$\_SESSION['cart'] = array();

}

// look for catalog file

$catalogFile = "catalog.dat";

// file is available, extract data from it

// place into $CATALOG array, with SKU as key

if (file\_exists($catalogFile))

{

$data = file($catalogFile);

foreach ($data as $line)

{

$lineArray = explode(':', $line);

$sku = trim($lineArray[0]);

$CATALOG[$sku]['desc'] = trim($lineArray[1]);

$CATALOG[$sku]['price'] = trim($lineArray[2]);

}

}

// file is not available

// stop immediately with an error

else

{

die("Could not find catalog file");

}

// check to see if the form has been submitted

// and which submit button was clicked

// if this is an add operation

// add to already existing quantities in shopping cart

if ($\_POST['add'])

{

foreach ($\_POST['a\_qty'] as $k => $v)

{

// if the value is 0 or negative

// don't bother changing the cart

if ($v > 0)

{

$\_SESSION['cart'][$k] = $\_SESSION['cart'][$k] + $v;

}

}

}

// if this is an update operation

// replace quantities in shopping cart with values entered

else if ($\_POST['update'])

{

foreach ($\_POST['u\_qty'] as $k => $v)

{

// if the value is empty, 0 or negative

// don't bother changing the cart

if ($v != "" && $v >= 0)

{

$\_SESSION['cart'][$k] = $v;

}

}

}

// if this is a clear operation

// reset the session and the cart

// destroy all session data

else if ($\_POST['clear'])

{

$\_SESSION = array();

session\_destroy();

}

?>

<html>

<head></head>

<body>

<h2>Catalog</h2>

Please add items from the list below to your shopping cart.

<form action="<?=$\_SERVER['PHP\_SELF']?>" method="post">

<table border="0" cellspacing="10">

<?php

// print items from the catalog for selection

foreach ($CATALOG as $k => $v)

{

echo "<tr><td colspan=2>";

echo "<b>" . $v['desc'] . "</b>";

echo "</td></tr>\n";

echo "<tr><td>";

echo "Price per unit: " . $CATALOG[$k]['price'];

echo "</td><td>Quantity: ";

echo "<input size=4 type=text name=\"a\_qty[" . $k . "]\">";

echo "</td></tr>\n";

}

?>

<tr>

<td colspan="2">

<input type="submit" name="add" value="Add items to cart">

</td>

</tr>

</table>

<hr />

<hr />

<h2>Shopping cart</h2>

<table width="100%" border="0" cellspacing="10">

<?php

// initialize a variable to hold total cost

$total = 0;

// check the shopping cart

// if it contains values

// look up the SKUs in the $CATALOG array

// get the cost and calculate subtotals and totals

if (is\_array($\_SESSION['cart']))

{

foreach ($\_SESSION['cart'] as $k => $v)

{

// only display items that have been selected

// that is, quantities > 0

if ($v > 0)

{

$subtotal = $v \* $CATALOG[$k]['price'];

$total += $subtotal;

echo "<tr><td>";

echo "<b>$v unit(s) of " . $CATALOG[$k]['desc'] ↵

. "</b>";

echo "</td><td>";

echo "New quantity: <input size=4 type=text ↵

name=\"u\_qty[" . $k . "]\">";

echo "</td></tr>\n";

echo "<tr><td>";

echo "Price per unit: " . $CATALOG[$k]['price'];

echo "</td><td>";

echo "Sub-total: " . sprintf("%0.2f", $subtotal);

echo "</td></tr>\n";

}

}

}

?>

<tr>

<td><b>TOTAL</b></td>

<td><b><?=sprintf("%0.2f", $total)?></b></td>

</tr>

<tr>

<td><input type="submit" name="update" value="Update Cart"></td>

<td><input type="submit" name="clear" value="Clear Cart"></td>

</tr>

</table>

</form>

</body>

</html>

Pop it into your browser, and see how it works. When you first load it up, you’ll see a list of items. Select a few items by attaching quantities to them, and submit the form. The page will refresh and display those items to you in your shopping cart, together with unit and total costs.

Because the shopping cart is maintained in a session, your selection will be “remembered” even if you visit another site, and then come back to the page. The session will only be destroyed if you close your browser window or if you explicitly empty your cart by calling the **session\_destroy()** function.