# Client/Server Web Application Development

Mehdi Raoufi
Kimberly Spoa
Zachary Wiggins

Department of Information Systems and Computer Programming
Purdue University Calumet
Hammond, IN, 46323, USA

## Abstract

**Client Side Web Application Development** refers to coding in HTML and/or scripting languages. When a user opens a web page, which is stored in a server, the file is transferred from the server computer to the client computer and viewed in the client computer. If it consists of scripts, execution of these programs is done in the client computer. In **Server Side Web Application Development**, when a program stored in a server is accessed (usually written in some scripting language and/or Java along with HTML code), the program is executed in the server computer; HTML code is generated, which is then transferred to the client computer to be viewed. This tutorial starts with a review of HTML and how scripting languages as JavaScript/ VBScript can be used together with HTML code to add interactivity to web pages as Client Side Programming. Server Side Programming is then presented using Microsoft's dynamic document technology, ASP (Active Server Pages). The paper ends with an e-commerce application; Internet shopping center. The objective of the paper is to present client/server web application development not any scripting language, the reader is assumed to have some familiarity with HTML and a modern programming language.

**Keywords:** Dynamic document technology, active server pages (ASP), client side programming, server side programming, JavaScript, VBScript, HTML

## 1. REVIEW OF HTML

### 1.1 What is HTML?

HTML (Hypertext Markup Language) is a language used to layout documents. It is called a markup language because it consists of special tags to mark up the beginning and end of different parts of a document. The browser does not display the tags; only content between the tags is displayed according to the type of the tags. The language also allows a document connect with other documents through special hypertext links.

### 1.2 An HTML Document

To be consistent with most authors, we start our first example in HTML with a document, which displays "Hello World!". Use your text editor to create a file named helloworld with extension of .html or .htm and save it in your local disk. We use notepad; if you are using Microsoft Word make sure that save it in ASCII format, Microsoft Word's .doc files save hidden characters that may confuse the browser.

**Example 1.2.1:**

```
<!--This is a comment -->
<html><head><title>Hello World</title>
</head>
<body>
<h7>Hello World!</h7><br>
<h6>How Are You?</h6><p>
<a href = "http://www.calumet.purdue.edu>Click Here to See My School's Web Page</a></body>
</html>
```

Use your browser to view the file helloworld.html, the browser displays:

Hello World!

**How Are You?**

Click Here to See My School's Web Page

**Figure 1.2.1**

As you notice, the browser does not display the "<" and ">" symbols and everything between them.

### 1.3 Tags, the HTML Building Blocks

The symbols less-than "<" and greater-than ">", and everything between them is called a **tag**. The region starts where the tag starts for example <html> and con-

tinues until the corresponding end tag is counter </html>. Some tags do not require end tags like tags; <br>, which means break to the beginning of next line or <p>, which means start a new paragraph. Every HTML document begins and ends with <html> and </html> tags. The entire document is contained between these tags. Also every HTML document consists of two parts: a head and a body each bounded by begin and end tags. Information about the document goes to the head and the content of document goes to the body. One important header tag is title. Select a meaningful title and enclose it between the <title> and </title> tags as in our Example 1.2.1.

### 1.4 Comments, URLs, Hyperlinks, Anchors, and Images

Every string between <!-- and ending tag --> considered as a comment. **URL** (Unified Resource Locator) is an address or path for a file, web page, or even an email address. An example of a web page URL is: http://calumet.purdue.edu/public/index.html. The **http**:// (Hypertext Transfer Protocol) is the set of rules for communication between client and server. The www.calumet.purdue.edu is the **domain name** or location of a particular server where a web page resides. The /public/ is the directory name and index.html is a file name. **Hyperlinks** are links or paths to other web pages. A hyperlink is like a short cut, you click on it and you are transferred to another web page. **Anchors** allow hyperlinks possible; for example in example 1.2.1, the tags <a>…</a> are used to create a link to the Purdue University Calumet campus as <a href = "http://www.calumet.purdue.edu>Click Here to See My School's Web Page<a>. An **image** is an icon or a picture. It can also be used as a hyperlink to another web page. For example in our example 1.2.1, we could replace the message "Click Here to See My School's Web Page" by an image as <a href = "http://www.calumet.purdue.edu><img src = "picture1.gif"><a>.

### 1.5 Form

We encounter them frequently in our daily life. We have to fill a **form** to open a new bank account, or apply for a driver license, and we have to fill a form to purchase a product through Internet. Creating a form, using HTML is very simple. We explain it with a simple example. Please type the following example using notepad or your favorite editor.

**Example 1.5.1** Edit the following program:

```
<!-- File name: example1_5_1.html -->
<html>
<head><title>A Simple Form Example</title></head>
<body>
<form method = post action = "mailto:raoufi@calumet.purdue.edu"
enctype = "text/plain">
First Name:
<input type = text  name = firstname size = 20
maxlength = 80><p>
Last Name:
<input type = text name = lastname size = 20
maxlength = 80><p>
Street Address:
<input type = text name = streetaddress size = 20
maxlength = 80><p>
State:
<input type = text name = state size = 2
maxlength = 2><p>
Zip Code:
<input type = text name = zipcode size = 5
maxlength = 5><p>
Telephone Number:
<input type = text name = tel size = 20
maxlength = 80><p>
Sex:
<input type = radio name = sex value = male checked>Male
<input type = radio name = sex value = female>Female
<p>Age:<br>
<select name = age size = 2 >
<option>Under 18
<option>19 to 25
<option>26 to 35
<option>36 to 49
<option>50 or higher</select><p>
Select the degree(s) that you hold (check one or more)
<p>
<input type = checkbox name = highschool value = hs>High School Diploma<br>
<input type = checkbox name = as value = as>AS Degree<br>
<input type = checkbox name = bs value = bs>BS Degree<br>
<input type = checkbox name = ms value = ms>MS Degree<br>
<input type = checkbox name = phd value = phd>Ph.D. Degree
<p>
<input type = submit value ="Submit Entries"><p>
<input type = reset value = "Clear Entries">
</form></body>
</html>
```

If you view this file using a browser, you will view the following form. Fill up the form as instructed below; in a moment we will examine each segment in detail:

First Name: John

Last Name: Doe

Street Address: 121 1st St

State: IN

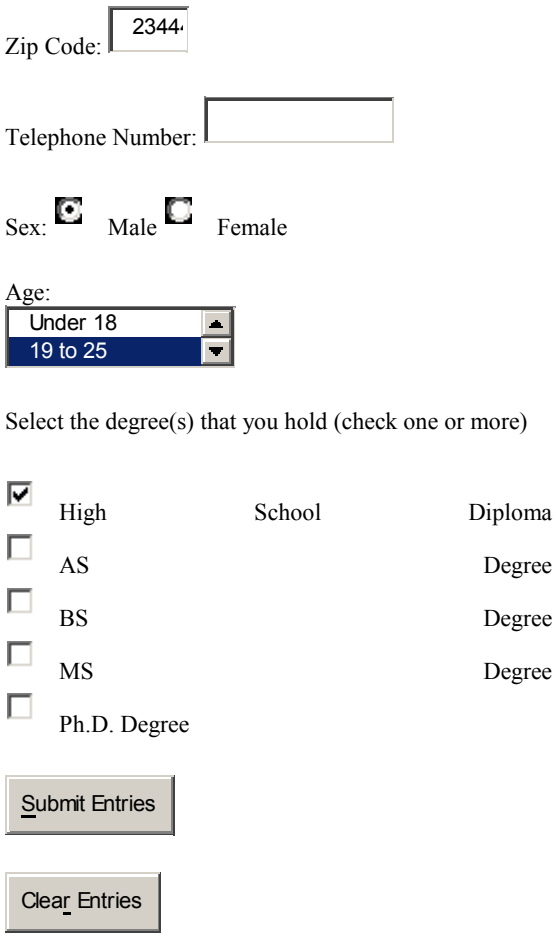


Figure 1.5.1

Let's examine example 1.5.1:
Consider the code:

```
<form method = post action =
"mailto:raoufi@calumet.purdue.edu"
enctype = "text/plain">
```

If we are working with one of the sites with no access to server, we may decide that we like to collect data through our email address. The data gets emailed to the email address provided through **action** attribute of form tag. With the **post** method the form data transferred to the server in separate transmission. The attribute value of post for method may be replaced with **get**, which causes the form to be transferred to the server, appended to URL. The attribute enctype specifies that the form data is being submitted unencrypted "text/plain", which is unsecured.
We use **<input>** tag to define any one of number of common form controls.
The frequently used input control is **text** entry field. The text entry field appears in the browser window as an entry box in one single line. We can specify size of field with size attribute, and size of maximum length of field with maxlength attribute.

```
<input type = text  name = firstname size = 20
maxlength = 80>
```

Next input type is **radio** button. Radio button gives the user choice to select an item quickly. In radio button user can select only one in the group. Using the checked attribute, we may initially check one of the radio buttons. In our example the Male button is selected to have value checked initially. Name and value attributes are required for radio buttons. The content of the name attributes of a group of radio buttons must be the same. The code below will enable the user to select only one of the items.

```
Sex:
<input type = radio name = sex value = male checked>Male
<input type = radio name = sex value = female>Female
```

In **<select>** tag, we place a list of **<option>** tags inside the <select> tag of a form. This creates a pull-down menu of choices. The name attribute is required for <select> tag. Unlike the radio button, one may select single or **multiple** options. To have multiple option selection enabled, we must specify the <select> tag with multiple attribute "<select name = age multiple>". To select multiple options, one must use control key and click on the selected option simultaneously. The size attribute specifies the number of options must be visible to the user at one time. The following code has five options; only two will be visible at a time.

```
Age:
<br>
<select name = age size = 2 >
<option>Under 18
<option>19 to 25
<option>26 to 35
<option>36 to 49
<option>50 or higher
</select>
```

The **checkbox** form gives user a way to select one or more items from a group of items quickly and easily. Name and value attributes are required for checkboxes. The code below will enable the user to select one or more of the options.

```
Select the degree(s) that you hold (check one or more)
<p>
<input type = checkbox name = highschool value = hs>High
School Diploma
<br>
<input type = checkbox name = as value = as>AS Degree
<br>
<input type = checkbox name = bs value = bs>BS Degree<br>
<br><input type = checkbox name = ms value = ms>MS De-
gree<br>
<input type = checkbox name = phd value = phd>Ph.D. De-
gree
```

The final two buttons are simple submit, and reset button. Submit button submits the entries and reset button clears all of the entries.
The actual data received by submitting the above form is:

```
firstname=John
lastname=Doe
streetaddress=121 1st St
state=IN
zipcode=23444
tel
sex=male
age=19 to 25
highschool=hs
```

## 2. CLIENT SIDE WEB APPLICATION DEVELOPMENT USING JAVASCRIPT

### 2.1 What is Client Side Programming?

**Client side programming** refers to web pages that use processor of client computer to execute the scripting program. The client computer is a desktop or a laptop computer that is used to view a web page. Using client side programming to do some of the processing improves efficiency, because it shifts some of the burden away from server computer to client computer. If your form requires calculations, or input validation you can do them in JavaScript or VBScript on the user's machine without a need to use processor time of your server. In this chapter we also cover **window** and **document** objects of the browser, which are used in client side programming.

### 2.2 A simple client side program

**Example 2.2.1:** Write a client side program in JavaScript, which simulates toss of a coin n times. Use your editor to edit the following program. Save the program in a file, name it example2_2_1.html.

```
<html>
<head><title>Toss a coin n times</title></head>
<body><center>
<h1>This is my First Client Side Program</h1>
</center>
<script language = "JavaScript" >
// the function coin() generate a random integer 0 or 1
// note that javascript does not require return type
// specified in function definition
function coin()
{
        if( Math.random() < .5 )
        return 1;
        else
        return 0;
}
tails = 0;
heads = 0;

number = parseInt(window.prompt("Enter an integer number:
","",""));
for ( i = 1; i <= number; i++)
{
        if( coin() == 0 )
        heads = heads + 1;
        else
        tails = tails +1;

}
document.writeln("<p><p>");
document.writeln("The coin is tossed: " + number + " times " +
"<p>");
document.writeln( "The number of tails is: " + tails + " <p>");
document.writeln( "The number of heads is: " + heads + "
<p>");
</script></body></html>
```

The scripting portion of the program must start with the tag **<script>** and terminate with the tag**</script>**. You can inset JavaScript code in different locations of a document as far as the script tags are used to specify beginning and end of the code. The attribute, **language** is optional in new versions of explorer. Unlike Java and C++, JavaScript does not require that a variable be declared prior its usage. The user has option to declare the variable if he desires so. The syntax to declare a variable in JavaScript is:

```
var tails; // declares a variable in JavaScript
```

The function prompt() of the window object is used in following statement to prompt the user to enter a value

```
number = parseInt(window.prompt("Enter an integer number:
","",""));
```

The function writeln() of the document object is used to writes the content of the string parameter on the HTML document using document object. document.writeln("The coin is tossed: " + number + " times " + "<p>");

Both window and document are objects of **BOM** (Browser Object Model).

Use your browser to view the file example2_2_1.html, the output, for the input value of 111 is:

**This is my First Client Side Program**

The coin is tossed: 1111 times
The number of tails is: 545
The number of heads is: 566

### 2.3 Using client computer to validate input before a form is submitted:

In this section, we create a form where a user enters his first name, last name, zip code, and we write a JavaScript program to do some basic validation of the form client side, and only post the form if it's actually filled in, and zip code field consists of exactly five digits. To validate the zip code field we use a JavaScript object called **RegExp**. The statement: var myRegExp = /\d\d\d\d\d/; creates an object of RegExp object that can test if an string has exactly five digits using the member method **test().** The statement myRegExp.test(str)); returns true if string str has exactly five digits and false otherwise. The function **alert()** of the object window alerts the user to re-enter an incorrect input.

**Example 2.3.1:** Edit the following program and view it using your browser. Test the program with invalid entries.

```
<html>
```

```
<head><title>Validate Input</title>
</head><body>
<script>
// The function isValidZipcode() uses Regular //Expression object to test
// if str is a valid zip code; if str is a valid zip code, it //returns true otherwise false.
function isValidZipcode(str)
{
   var myRegExp = /\d\d\d\d\d/; // exactly 5 digits
//  You can also replace the above statement with  var myRegExp = /\d{5}/;
   return ( myRegExp.test(str));
}
</script>
<script>
// The function validInput() returns true if first name and //last name fields
// are not null and zip code is valid, otherwise prompts //the user to reenter.
function validInput()
{
   // if input text is null string
   if ( document.regform.firstname.value == "")
            {
// the method alert() is a member  method of object //window
// the default object is window, can also be written as //window.alert()
               alert("Please enter your first name: ");
               document.regform.firstname.focus();
               return false;
            }

 if ( document.regform.lastname.value == "")
            {
               alert("Please enter your last name: ");
               document.regform.lastname.focus();
               return false;
            }

 if ( !isValidZipcode(document.regform.zipcode.value) )
            {
               alert("Please enter your zip code: ");
               document.regform.zipcode.focus();
               return false;
            }
  return true;
}
</script>
<form method = post name = regform
// The form is submitted only if function validInput() //returns true
onSubmit = "return validInput()"
action = "mailto:raoufi@calumet.purdue.edu"
enctype = "text/plain">
First Name:
<input type = text  name = firstname size = 20
maxlength = 80><p>
Last Name:
<input type = text name = lastname size = 20
maxlength = 80><p>
Zip Code:
<input type = text name = zipcode size = 5
maxlength = 5><p>
<input type = submit   value ="Submit Entries">
<p>
<input type = reset value = "Clear Entries">
</form></body>
</html>
```

The program will not submit the entries as long as the validating functions validInput() return false. The function validInput() should be embedded in the tag form as:

```
<form method = post name = regform
// The form is submitted only if function validInput() //returns true
onSubmit = "return validInput()"
action = "mailto:raoufi@calumet.purdue.edu"
enctype = "text/plain">
```

The definition of function validInput() is explained in the body of the program:

## 3. SERVER SIDE WEB APPLICATION DEVELOPMENT USING ASP/VBSCRIPT

### 3.1 What is Server Side Web Application Development?

As we mentioned earlier, the client side program must be downloaded to the client, to be executed by the client computer. Server side programs run directly on the server and generate HTML code to be viewed by the browser on client computer. The common places to encounter these programs are search engines, up-to-date stock reports and shopping through Internet. There are different technologies for server side Web Application Development; these technologies are called **DDT**s (dynamic document technologies). Other dynamic document technologies beside ASP are **CGI** (Common Gateway Interface), **JSP** (Java Server Pages), and **PHP** (Personal Home Pages). ASP is a Microsoft's technology and is used in this paper.

### 3.2 Request and Response objects: A web page (.html or .asp extension), which asks a user to enter information and returns the user's response to another ASP page to display on a document

Before we start our first example in server side programming, we need to learn a little about ASP files and their contents. Every ASP file has extensions .asp. A file in ASP may consist of scripts written in VBScript, JavaScript and/or HTML. When browser requests a file with extension .asp, if the file contains any scripts, it is executed on the server, HTML code is generated, and then browser is used to view the generated HTML code.

An ASP program can contain one or more of the following codes; client side script, server side scripts, and HTML. For the server to distinguish the client side scripts from server side scripts, some rules must be followed.

The client side scripts must begin with the tags <script> and end with the tag </script>:

<script language = “VBScript”> client side script here </script>
or
<script language = “JavaScript”> client side script here
</script>.
The attribute, language is optional in later versions of the explorer browser. The server side scripts also use tags <script> and </script>, but attribute runat = “server” must be included with the script tag as
<script runat = “server”> server side script </script>
Also, the tags <% and %> can be used to mark the beginning and end of a server side script as
<% server side script here %>
We prefer the tags <% and %> for server side scripts and will use them in our examples.

**Example 3.2.1:** Let’s modify example 1.5.1 to send the form to an ASP program instead of an email address. The form sent to ASP program may be a registration form, an airline reservation form, an order form to purchase a product through Internet, etc.
In this example we replace the email address attribute of the form with the address of the ASP program that must receive and process the information in the form. In this example, we name the ASP file processreg.asp. Please note that for simplicity, we have only retained the following fields from example 1.5.1: first name, last name, street address, state, and zip code. Edit the following program and save it under example3_2_1.asp.

```
<html>
<head>
        <title>Registration Form</title></head>
<body>
<form name = registrationform method = post action = "proc-
essreg.asp">
First Name:
<input type = text  name = firstname size = 20
maxlength = 80><p>
Last Name:
<input type = text name = lastname size = 20
maxlength = 80><p>
Street Address:
<input type = text name = streetaddress size = 20
maxlength = 80><p>
State:
<input type = text name = state size = 2
maxlength = 2><p>
Zip Code:
<input type = text name = zipcode size = 5
maxlength = 5><p>
<input type = submit value ="Submit Entries">
<p>
<input type = reset value = "Clear Entries">
</form></body>
</html>
```

Upon submission of this form the content of the form will be transferred to the server, and can be returned using **Request** object of ASP. In our example the content of the form sent by the user is available to the file processreg.asp using Request object. The content of each field of the form can be returned by the function Form() of the object Request, for example the value of the field firstname can be returned using:
Request.Form(“firstname”);
The function Write() of the object **Response** of ASP can be used to display the content of the field:
Response.Write ”Request.Form(“firstname”)”
In general the program processorreg.asp can process the information in the form as instructed by the programmer. Another application may be to modify content ofa Database based on user’s input.
Please also note that the symbol “=” can be used to display field firstname instead of Response.Write:
= ”Request.Form(“firstname”)”
Use your editor to create the following file and save it under the name processreg.asp in the same directory as example3_2_1.asp:

```
<!--file name processreg.asp -->
<html><head>
<title>Registration Response</title>
</head><body>
<center><h1>Registration Response</center>
<p>
The information that you entered:</h1><p>
First Name: <%Response.Write ”Request.Form(“firstname”)”
%><p>
Last Name: <%Response.Write ”Request.Form(“lastname”)”
%><p>
Street Address: Last Name: <%Response.Write ”Re-
quest.Form(“streetaddress”)” %><p>
State: <%Response.Write ”Request.Form(“state”)” %><p>
Zip Code: <%Response.Write ”Request.Form(“zipcode”)”
%><p>
</body>
<html>
```

Please note that the code

```
<%Response.Write ”Request.Form(“firstname”)” %>
```

will be executed by the server. The statement Request.Form(“firstname”) returns the content of the field firstnsme and statement Response.Write displays the returned value on HTML document.

The output viewed by the user after the file example3_2_1.asp is submitted is:

First Name:
Last Name:
Street Address:
State:
Zip Code:

**3.3 An e-commerce application for an Internet shopping center.**
In this section we implement an e-commerce store at which a user can shop for items, put them into a cart, and check them out. Since we need an inventory of some kind we have prepared a file “shopdata.txt”. The items

in the file are tab separated and have the following form: product id, product name, supplier name, category, quantity, price and description.

**Example 3.3.1**: The **Shop Till Drop** Internet store has following main areas:

1. The shopping center (shop.asp): this is where the inventory is displayed dynamically from text file. The user is allowed to select the categories using checkboxes and submission button that loads a page with only items in the checked categories. Next to each item is a checkbox that allows the buyer to place an item into the cart. In submission your choice of items are added to the shopping cart and a link on web page takes you to the shopping cart (cart.asp) web page that displays the confirmation of the buyer's choices.
2. The shopping cart (cart.asp): The shopping cart web page displays the current content of the cart and total cost. There are also an input item next to each item where buyer can enter the quantity desired and a recalculate button to recalculate the total cost.
3. The item page (item.asp): This will dynamically display all of the available items at the store.
4. The checkout page (checkout.asp): There is a login for this page that asks information (name, credit card number, address, etc.). After logging in, the buyer will be taken to a page bringing up the order from the cart, and a submit button will submit the order.

All of the files of examples3.3.1: shop.asp, cart.asp, itempage.asp, checkout.asp, shopdata.txt, and all examples of this tutorial are available at:

http://www.cis.calumet.purdue.edu/raoufi/isecon01

First we go over the procedure that is needed to make the content of the file shopdada.txt available for reading:

```
<% CONST ForReading = 1, ForWriting = 2, ForAppending =
8
  CONST TristateUseDefault = -2, TristateTrue = -1, Tristate-
False = 0
'       Procedures openFile()
sub openFile(objFileTextStream, strVirtualPath)
DIM strPhysicalPath
strPhysicalPath = Server.MapPath(strVirtualPath)
DIM objFSO, objFile
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.GetFile(strPhysicalPath)
Set objFileTextStream = ob-
jFile.OpenAsTextStream(ForReading, TristateUseDefault)
end sub
```

The procedure openFile() used in this example to read input file. Please note that in this procedure we have used the **FileSystemObject** object of ASP. This object provides functionality to manipulate the files; it is at the top of the hierarchy and has several child objects: **Drive** object, **Folder** object, and **File** object, for more discussion of these objects please refer to (Morneau 2001)

We can use the **CreateObject** function of **Server** object of ASP to create an instance of the FileSystemObject object. Once a FileSystemObject object is created, it can be incorporated into VBScript as:

```
DIM objFSO
Set objFSO = CreateObject("Scripting.FileSystemObject")
```

We use the method GetFile() of FileSystemObject object to get a file object, where physical path of the file is provided as parameter:

```
 Set objFile = objFSO.GetFile(strPhysicalPath)
```

Please note that physical path; physical location of a file can be obtained using the logical (virtual) path of the file. In our example the file name shopdata.txt is logical path and we can use the method **Mappath** of Server object to obtain the physical path of the file shopmap.txt as:

```
strPhysicalPath = Server.MapPath(strVirtualPath)
```

And finally the procedure needs to open the file as a text stream:

```
Set objFileTextStream = ob-
jFile.OpenAsTextStream(ForReading, TristateUseDefault)
```

Where first parameter equal to 1 is for reading and second parameter –2 is the default character format.

The **TextStream** object can be used to manipulate the information contained in the file.

After the content of the file is available for reading, the following program segment reads and displays the buyer's choices:

```
<form action="cart.asp" method="post" >
<%    DIM strPathInfo
  strPathInfo = "shopdata.txt"
  DIM objFTS
  call openFile(objFTS, strPathInfo)
 Response.Write "<TABLE  "
strLine = objFTS.Readline
WHILE objFTS.AtEndOfStream <> TRUE
      strResults = split(strLine, vbTab)
      strID = strResults(0)
      strName = strResults(1)
      strSupplierName = strResults(2)
      strCategory = strResults(3)
```

```
      strSizeQ = strResults(4)
      strPrice = strResults(5)
      strDescription = strResults(6)
      if Request.Form("catBeverages") = "on" and strCategory
= "Beverages" then
            call writeLine()
     end if
     if Request.Form("catCondiments") = "on" and strCategory
= "Condiments" then
            call writeLine()
    end if
    if Request.Form("catProduce") = "on" and strCategory =
"Produce" then
            call writeLine()
    end if
    if Request.Form("catConfections") = "on" and strCategory
= "Confections" then
             call writeLine()
    end if
    if Request.Form("catMeat/Poultry") = "on" and strCategory
= "Meat/Poultry" then
            call writeLine()
     end if
    if Request.Form("catDairyProducts") = "on" and strCate-
gory = "Dairy Products" then
            call writeLine()
    end if
    if Request.Form("catSeafood") = "on" and strCategory =
"Seafood" then
            call writeLine()
    end if
    if Request.Form("catGrains/Cereals") = "on" and strCate-
gory = "Grains/Cereals" then
            call writeLine()
    end if
    strLine = objFTS.Readline
 WEND
```

## 1. CONCLUSION

This paper has exemplified client/server application web development for e-commerce application. We recommend the material in this tutorial for every student in the field of Information Systems.

## 5. REFERENCES


Castro, E., 2000, HTML for the World Wide Web, Peachpit Press

Kauffman, J., Llibre, J., Sussman, S., 1999, Beginning Active Server Pages 3.0, Wrox Press Ltd.

Mornear, H., Batistick, J., 2001, Active Server Pages, Course Technology

Musciano, C., Kennedy, B., 2000, HTML & XHTML: The Definitive Guide, O'Reilly

Negrino, T., Smith, D., 1999, JavaScript for the World Wide Web, Peachpit Press

Ullman, C. Buser, D., Duckett, J., Francis, B., Wilton, P., 2000, Beginning JavaScript, Wrox Press Ltd.