Project 2
Web Application Development

Due date: 23:59pm Sunday 22 October 2023 -- Worth 45%
[Late submission will be penalized, 10% per day for at most 5 days]

NOTE: Extensions must be applied for before the due date by emailing the unit convener. A request for extension after the due time or without a valid medical certificate will NOT be responded. So, make sure you attempt your project early and do not leave the submission to the last minute. Computer failure, network issue, transport problem, travel, house moving and other personal events are not valid reasons for extension.

Project Description

This is an individual project. Students are referred to the Faculty's policy on plagiarism. ***All work must be your own.*** **Submissions are *automatically checked* for similarities. Carefully read the section on plagiarism in the Unit Outline before you proceed.**
The aim of this project is to develop a better understanding of building web applications using **Ajax technologies**. You are requested to use all Ajax techniques (JavaScript/HTML, DOM, XML, XMLHttpRequest, and XPath/XSLT) and PHP.

Project Tasks

The project is to develop a web-based selling and buying system called *ShopOnline*. *ShopOnline* allows sellers to list items for selling and buyers to bid for the items based on the *English Auction* strategy (http://en.wikipedia.org/wiki/English_auction), which is a popular strategy used in many systems such as eBay, real-estate auction. Five components (registration, login, listing, bidding, and maintenance) of *ShopOnline* that must be completed for this project are specified in the sub-sections below. You may explore further functions such as detailed payment processing, notification service and password changing for fun later. *For this project, you must use XMLHttpRequest for client/server communications*.

Task 1. Login/Registration

These two components are used to allow an existing customer to login to the system and to allow a new customer to register, respectively. The system maintains an XML document to record customer information. The specific functions include

1) Design and create an XML document (customer.xml) for storing information of all customers. For each customer (regardless a seller or a bidder), you need to keep the generated customer id, first name, surname, unique email address, and password.
2) For an existing customer, the email address and password are expected and checked against the XML document (see Figure 1). If a customer is matched against the information stored in the XML document, the customer information will be remembered by the system for the following sessions (for either bidding or listing) before log-out and current auction items will be shown up (i.e., switch to the bidding page); otherwise, login failure message will be displayed for invalid email address or password.

(*Hint*: you may use session variable to keep the information about the customer.)

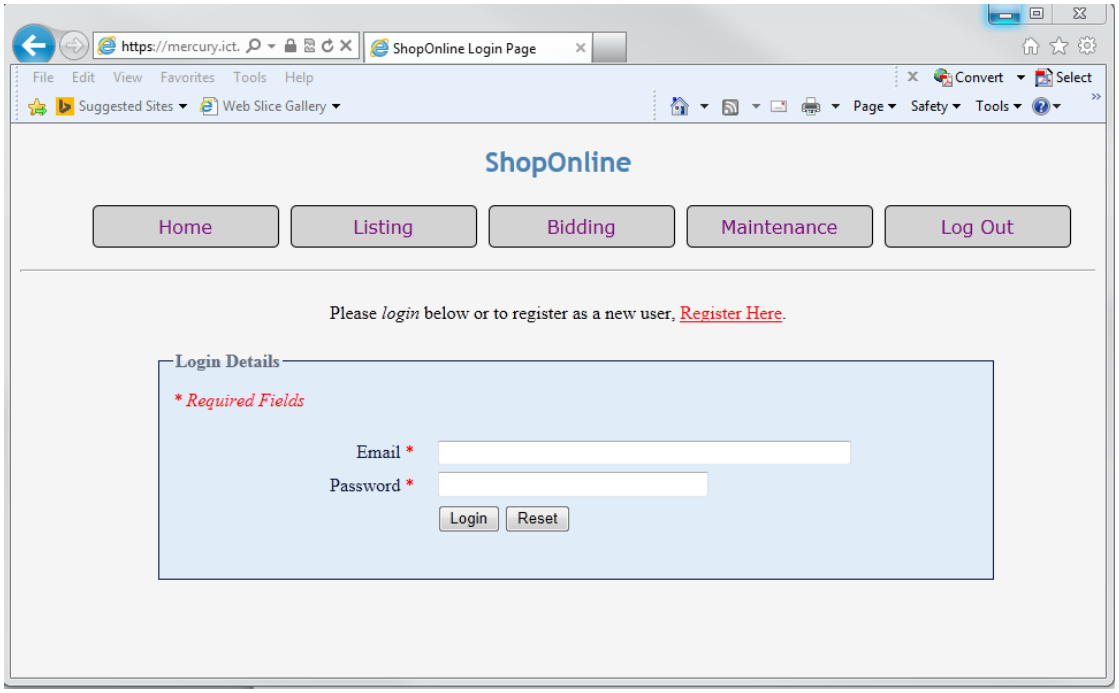

Figure 1: Login for Existing Customers

3) For a new customer, the system will check (a) all inputs including first name, surname, email address, password, re-typed password (for double checking) are given (shown in Figure 2), and (b) the email address is unique (i.e., it has not been used by other customers in the customer document), and (c) the email address is valid. The checking of a valid email address is done by using regular expression. The complete set of rules for a valid email address can be found from Wikipedia http://en.wikipedia.org/wiki/Email_address#Valid_email_addresses. We simplify the rules as follows:
   a) The format of email addresses is local-part@domain-part;
   b) The local part may contain
      i) Uppercase and lowercase English letters (a–z, A–Z) (ASCII: 65–90, 97–122)
      ii) Digits 0 to 9 (ASCII: 48–57)
      iii) Characters !#$%&'*+-/=?^_`{|}~ (ASCII: 33, 35–39, 42, 43, 45, 47, 61, 63, 94-96, 123–126) (You are allowed to simplify this by selecting a few characters)
      iv) Character . (dot, period, full stop) (ASCII: 46)
   c) The domain name must match the requirements for a hostname, consisting of letters, digits, hyphens and dots.

   If there is problem with the above checking, the corresponding error message will be displayed; otherwise, the system will generate a customer id before sending the message.

   Recipient: the provided <email address>
   Subject: "Welcome to ShopOnline!"
   Message: "Dear <first name>, welcome to use ShopOnline! Your customer id is <customerID> and the password is <password>."
   Header: "From registration@shoponline.com.au"

When sending email from php using the mail() function, you should specify an envelope sender who will receive the bounce messages as shown below:

```
mail($to, $subject, $message, $headers);
```

The system will also store all the required information for the customer into the XML document. If the XML document does not exist (i.e. when the first customer is registered in the system), create a new one. Finally, the system will show the successful registration information. Similar to login, the customer information will also be remembered by the system for the following sessions (for either bidding or listing) before log-out.

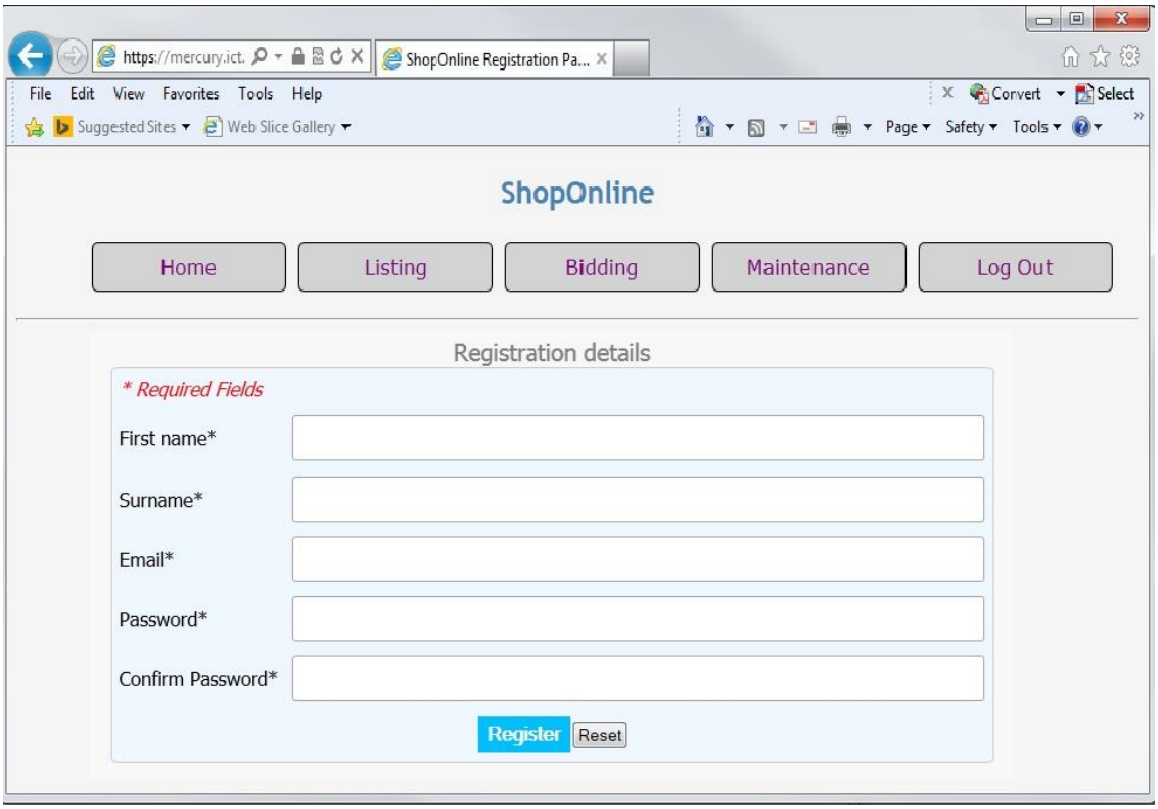

Figure 2: Registration for New Customers

Task 2. Listing

This component allows a customer (as a seller) who logged in the system to add an item into *ShopOnline* for selling. The inputs include item name, category, description, reserve price, buy-it-now price, start price (default 0), duration. Once you get these inputs, you need to validate the inputs, generate an item number, and add the item listing information together with the customer id of the seller and other system generated information in another XML document. The specific functions of this component include

1) Design an XML document (auction.xml) for storing information about all listed items. For each listed item, apart from the inputs from the seller, you also need to keep the customer id of the seller, system generated item number, the current system date and time that the item is put up for auction (startDate and startTime), the status (which takes *in_progress*, *sold*, and *failed* as value, and *in_progress* as the initial value), and

the information about the latest bid which include customer id of the bidder and the current bid price (with the start price as initial value).

2) As shown in Figure 3, design the user interface to take a seller's inputs for an item. For category, use a drop-down list with its item values retrieved from existing categories in auction.xml plus an "other" item allowing the seller to list an item belonging to new categories. When the Listing button is pressed, the system will validate the inputs. In particular, the start price must no more than the reserve price and the reserve price must be less than the buy-it-now price.

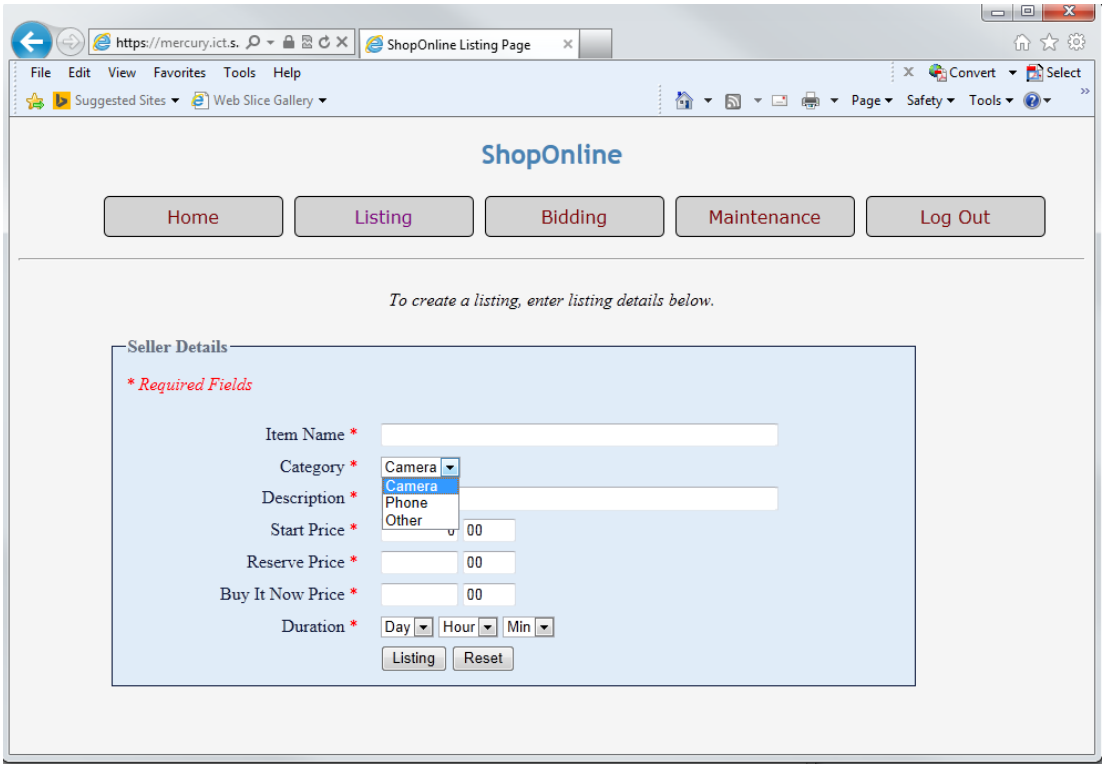

Figure 3: Listing an Item for Selling

3) If there is no problem with the inputs, the system will generate an item number, startDate and startTime, and add them together with the customer id of the seller and the seller's inputs to the XML document. If the XML document does not exist (i.e. when the first item is entered in the system), create a new one. We assume that the auction process starts immediately after the item is listed so the status is set to *in-progress*. The current bid price is also set to the value of the start price. Finally, you need to return the generated item number, start date and start time to the client and display under the inputted data "Thank you! Your item has been listed in ShopOnline. The item number is <itemNumber>, and the bidding starts now: <startTime> on <startDate>.

Task 3. Bidding

This component allows a customer (as a bidder) to view all the listed items and to put in a bid. The specific functions of this component include

1) As shown in Figure 4, design the user interface to periodically (say, every 5 seconds) retrieve all items in the XML documents and neatly display them. Instead of

displaying duration, startDate and startTime, show the time left. For each item, display the item number, name, category, description (first 30 characters), buy-it-now price, current bid price, and time left. Don't display the reserve price. For each displayed item, if the item has not been sold and the time left is greater than zero, add two buttons: Place Bid and Buy It Now button; otherwise, show either the item has been sold or the time expired.
(Note: if you implement these two buttons under all the displayed items, then the item number is required as part of the inputs. If you choose to implement it like this, a penalty of 2 out of 72 marks will apply)

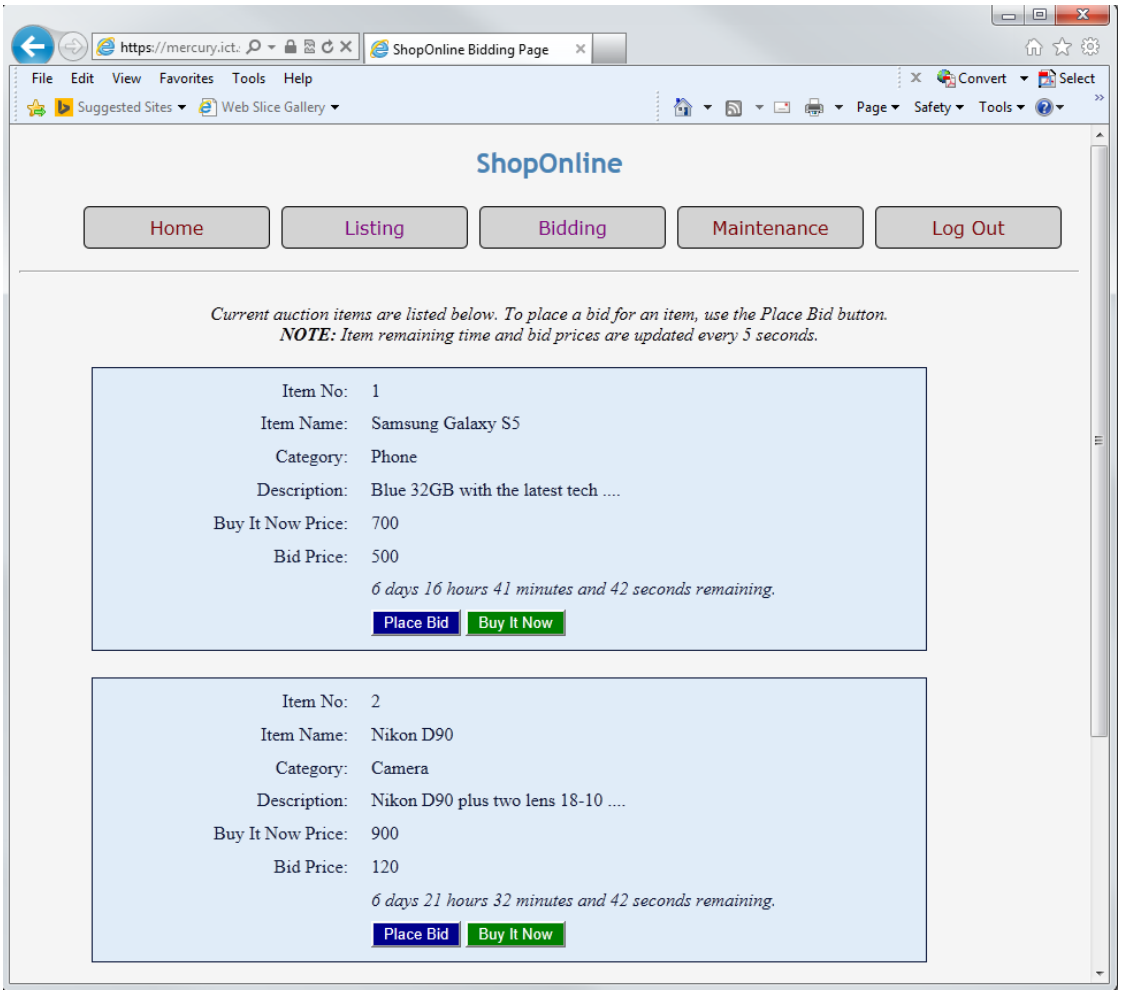

Figure 4: Auction Items for Placing Bid or Buying It Now

2) Once the Place Bid button is clicked, a pop-up window will show up to take the new bid price as input, and the bid request with the new bid price, item number and bidder's customer id will be sent for processing. You need to check if the new bid for the item is acceptable (i.e. the new bid price is higher than the current bid price of the item and the item is not sold), and if so, update the item in the XML document with the new bid price and the new bidder's customer id, and send back an acknowledgement "Thank you! Your bid is recorded in ShopOnline." to the customer; otherwise send back a message "Sorry, your bid is not valid." to the customer.

3) Once the Buy It Now button is clicked, the buy-it-now request will be sent for processing with the item number and customer id. You need to update the item in the XML document by changing the current bid price by the buy-it-now price, the bid by the customer id, and setting the status immediately to "sold". Then you send back an acknowledgement "Thank you for purchasing this item." to the customer.

Task 4. Maintenance

This component allows the broker to process items that are sold or expired before the current date and time, and to report the "sold" and "failed" items. It should only allow system administrators rather than normal customers to use these functions. In this project, we do not place this restriction.

1) Design the user interface that has two functions: "Process Auction Items" and "Generate Report".
2) If the "Process Auction Items" button is pressed, the system will check each item with "in_progress" status in the XML document to see if it is expired by calculating the time left (using current date/time, start date/time, and duration). If the time left is zero (or negative), then check the current bid price and reserve price to determine the status of the item ("sold" or "failed") and change the status of the item in the XML document accordingly. Once the processing is finished, display a message to show the process is complete.
3) If the "Generate Report" button is pressed, the system will retrieve all sold or failed items and compute the revenue (assume that you charge 3% of the sold price from each sold item and charge 1% of the reserved price from each failed item) from these items. We also choose to remove these items from the XML file. The list of the sold or failed items (all information about the item except the description) formatted as a table will be displayed. The total number of *sold* and *failed* items and the revenue will also be displayed under the table (You will find it easy to implement this function using XSLT).

Submission Requirements

You must ensure that all your program files used for the project sit in a directory called "Project2" (use this name exactly, it is case sensitive and no space between Project and 2) under *www/htdocs* directory within your mercury account. This directory should contain no other files and no other sub-directories. However, your XML documents *customer.xml* and *auction.xml* must be placed under *www/data* directory.

All files required by the project description must be submitted as one single ZIP file by using the electronic submission system (ESP). The files should include:

- XHTML files *login.htm, register.htm, listing.htm, bidding.htm, and maintenance.htm*;
- the XML data file *auction.xml* with at least 5 test items;
- the JavaScript, PHP, and XSLT files that you use;
- other files that you used;
- *readme.doc* that includes
    - a list of all the files in the system;
    - brief instructions on how to use the system.

- a file of checklist for tasks completion – see a sample format provided in the last page attached. If your program does not work correctly, you must provide descriptions of all defects under the checklist table. You must indicate which task your program cannot perform (e.g. I cannot perform the email uniqueness checking in Task 2.3), otherwise you will get further penalty -4/72 marks for that task;

For each submitted file, we require the minimum comments including student information and the main function for the file. After submission, you are not allowed to change any of the submitted files in the **Project2** directory and *auction.xml* on your mercury account; time stamps will be checked. **Projects that fail to follow "submission requirements" will NOT be assessed.**
If you use your PC/laptop for the project, you must make sure your completed project is loaded under your mercury account and works fine. We strongly recommend that you give sufficient time to do so!

Demonstration

Demonstration may be needed if the marker has some problems on your code and/or running your program.

Marking Scheme

Work will be assessed based on quality and presentation. The project will be marked out of 72 and will contribute 18% towards assessment of the unit.

Assessment item	Marks
Minimum comment; readme.doc and quality of code	3
Task 1.1: design customer.xml	2
Task 1.2: check email/password, keep login information in session, switch to bidding page or show failure information	5
Task 1.3: simple input validation, check unique email address, check valid email address, send email, generate customer id, create xml, update xml, keep login information in session	10
Task 2.1: design auction xml	3
Task 2.2: UI, drop-down list generation, input validation	6
Task 2.3: generate item number etc., get customer id, initial value setting, create xml, update xml, return information	9
Task 3.1: UI, auto refresh and display, add two buttons	7
Task 3.2: take input, input validation, get item number from UI and bidder id from session, update xml, return information	8
Task 3.3: get item number from UI and buyer id from session, update xml	4
Task 4.1: UI	2
Task 4.2: update xml based on left time and current bid price/reserve price	5
Task 4.3: retrieve required items, display in table, calculate/show total number of sold/failed items, calculate revenue, remove items from	8

xml	
Penalty for not reporting defects / uncompleted tasks	**-4/task-x.x**
Penalty: project cannot run on Mercury AND/OR this project is not your original work	**-72**
Total	72

Checklist for Tasks Completion (see next page)

Name:_____ Student ID:_____

Checklist of Tasks Completion ***(please tick each one as appropriate)***

Assessment item	**Completed**
Comment and readme file	YES ☐ NO ☐ PARTIALLY ☐
Task 1.1:	YES ☐ NO ☐ PARTIALLY ☐
Task 1.2:	YES ☐ NO ☐ PARTIALLY ☐
Task 1.3:	YES ☐ NO ☐ PARTIALLY ☐
Task 2.1:	YES ☐ NO ☐ PARTIALLY ☐
Task 2.2:	YES ☐ NO ☐ PARTIALLY ☐
Task 2.3:	YES ☐ NO ☐ PARTIALLY ☐
Task 3.1:	YES ☐ NO ☐ PARTIALLY ☐
Task 3.2:	YES ☐ NO ☐ PARTIALLY ☐
Task 3.3:	YES ☐ NO ☐ PARTIALLY ☐
Task 4.1:	YES ☐ NO ☐ PARTIALLY ☐
Task 4.2:	YES ☐ NO ☐ PARTIALLY ☐
Task 4.3:	YES ☐ NO ☐ PARTIALLY ☐

You should provide further details if "NO" or "PARTIALLY" is ticked.
Defects / Uncompleted tasks: