DEPARTMENT OF COMPUTER SCIENCE
ASSESSMENT DESCRIPTION 2016/17
(EXAM TESTS WORTH ≤15% AND COURSEWORK)

MODULE DETAILS:

Module Number:	08246	Semester:	2
Module Title:	Networking and Web Technologies		
Lecturer:	Eur. Ing. Brian Tompsett / Dr Bing Wang		

COURSEWORK DETAILS:

Assessment Number:	1	of	1		
Title of Assessment:	Location Client and Server with Live Web Interface				
Format:	Program	Demonstration			
Method of Working:	Individual				
Workload Guidance:	Typically, you should expect to spend between	60	and	120	hours on this assessment
Length of Submission:	This assessment should be **no** more than: (***over length submissions will be penalised as per University policy – see below)***	N/A - coding exercise			

PUBLICATION:

Date of issue:	31st January 2017

SUBMISSION:

ONE copy of this assessment should be handed in via:	Canvas		If Other (state method)	Lab Demonstration
Time and date for Part 1 submission:	**Time**	14:00	**Date**	7th March 2017
Time and date for Part 2 submission:	**Time**	14:00	**Date**	9th May 2017
If **multiple hand–ins** please provide details*:*	Part 1 will be submitted at 14:00 on the 7th March 2017. The stages of the work should have already been demonstrated in the labs for feedback. Part 2 must be submitted by 14:00 on the 9th May 2017. Demonstrations will take place during the exam period. A detailed schedule will be published at a later date.			
Will submission be scanned via TurnitIn?	No	If this requires a separate TurnItIn submission, please provide instructions:		

*Late submissions **will be** penalised as per university policy*

The assessment must be submitted **no later** than the time and date shown above, unless an extension has been authorised on a *Request for an Extension for an Assessment* form which is available from: http://www2.hull.ac.uk/student/registryservices/currentstudents/usefulforms.aspx.

If submission is via TurnitinUK within Canvas staff must set resubmission as standard, allowing students to resubmit their work, though only the last assessment submitted will be marked and if submitted after the coursework deadline late penalties will be applied.

MARKING:

Marking will be by:	Student Name

ASSESSMENT:

The assessment is marked out of:	40	and is worth	60	% of the module marks
N.B If multiple hand-ins please indicate the marks and % apportioned to each stage above (i.e. Stage 1 – 50, Stage 2 – 50). It is these marks that will be presented to the exam board.				

ASSESSMENT STRATEGY AND LEARNING OUTCOMES:

The overall assessment strategy is designed to evaluate the student's achievement of the module learning outcomes, and is subdivided as follows:

LO	Learning Outcome	Method of Assessment *{e.g. report, demo}*
3	*Develop an appropriate distributed application which is relevant for a suggested purpose*	Program, Demo
4	Develop an appropriate web user interface for a business application using remote resources	Program, Demo
5	*Complete an analysis and evaluation of network performance*	Demo

Assessment Criteria	Contributes to Learning Outcome	Mark
Part 1 (The maximum possible mark of 22 is capped at 20)		
Client Operation	3, 4	8
Server Operation	3, 4	8
Network Error Handling	3, 4	2
Coding Style, Design and Documentation Quality	3, 4, 5	2
Optional Features	3, 4	2
Part 2		
GET/POST functionality	3	4
Functionality of web pages	3,4	6
Database integration	3,4	3
Application robustness	3,4,5	4
Website presentation and innovation	3,4,5	3

FEEDBACK

Feedback will be given via:	Verbal (via demonstration)	Feedback will be given via:	Canvas
Exemption (staff to explain why)			
Feedback will be provided no later than 4 'semester weeks' after the submission date.			

This assessment is set in the context of the learning outcomes for the module and does not by itself constitute a definitive specification of the assessment. If you are in any doubt as to the relationship between what you have been asked to do and the module content you should take this matter up with the member of staff who set the assessment as soon as possible.

You are advised to read the **NOTES** regarding late penalties, over-length assignments, unfair means and quality assurance in your student handbook,
which is available on Canvas - https://canvas.hull.ac.uk/courses/17835/files/folder/Student-Handbooks-and-Guides.

In particular, please be aware that:

- Your work has a 10% penalty applied if submitted up to 24 hours late
- Your work has a 10% penalty applied and is capped to 40 (50 for level 7 modules) if submitted more than 24 hours late and up to and including 7 days after the deadline
- Your work will be awarded zero if submitted more than 7 days after the published deadline.
- The overlength penalty applies to your written report (which includes bullet points, and lists of text you have disguised as a table. It does not include contents page, graphs, data tables and appendices). Your mark will be awarded zero if you exceed the word count by more than 10%.

Please be reminded that you are responsible for reading the University Code of Practice on the use of Unfair means
(http://www2.hull.ac.uk/student/studenthandbook/academic/unfairmeans.aspx) and must understand that unfair means is defined as any conduct by a candidate which may gain an illegitimate advantage or benefit for him/herself or another which may create a disadvantage or loss for another. You must therefore be certain that the work you are submitting contains no section copied in whole or in part from any other source unless where explicitly acknowledged by means of proper citation. In addition, **please note** that if one student gives their solution to another student who submits it as their own work, **BOTH** students are breaking the unfair means regulations, and will be investigated.

In case of any subsequent dispute, query, or appeal regarding your coursework, you are reminded that it is your responsibility, not the Department's, to produce the assignment in question.

Assignment Details

Part 1

The coursework requires you to implement two C# Windows console applications which can be invoked by other programs; one is a client and the other is the server. These applications will eventually be components in a larger software system and their interfaces and names need to be implemented according to this specification to permit the correct interoperation between components. The client shall be called `location`, and the server called `locationserver`. They should use TCP sockets by means of the appropriate classes in System.Net and System.Net.Sockets. The task is to implement a simple student locating facility. Students will normally be identified by their computer login name, but the server need not be specific to Hull University, so any command argument string can be a student identifier. The client will simply use the Microsoft Command Prompt interface to run. The client and server will communicate using simplified forms of the internet whois protocol and the HTTP protocol as described below. The work can be implemented in stages and lectures will have the information needed to get you started. The second coursework assignment will build on the features of this one, so the more you are able to complete of the features of this program, the more helpful you will find it for later.

BASIC CLIENT INTERFACE SPECIFICATION

The client (which must be called `location`) can have two arguments, a user name and a string giving location information. In the examples below, the text `G:\08241\>` represents the command prompt.

For example:

```
G:\08241\> location cssbct
cssbct is in RB-336
```

In this example, `location` is the name of the executable of the client program, which is given one argument (`cssbct`) and the client shows the response "`cssbct is in RB-336`". This string is composed by the client from the argument supplied and the data returned by the location server. The client must output only this for a successful location lookup.

```
G:\08241\> location cssbct "in RB-310 for a meeting"
cssbct location changed to be in RB-310 for a meeting
```

In this example `location` is given two arguments, the first `cssbct` is the user name being updated, and the second is the string `"in RB-310 for a meeting"`. The quote characters (") are not passed to the client as part of the second argument, they are used to indicate to the windows command interpreter that the sequence of words separated by spaces are part of a single argument and are not five consecutive arguments to the client.

The response output by the client shows that the server has acknowledged the update was successful and is composed from the arguments to the client.

```
G:\08241\> location cssbct
cssbct is in RB-310 for a meeting
```

This example shows an enquiry to the server after making the previous update, which illustrates the new location is returned for further queries. It also allows you to determine the specified syntax of replies.

In summary these examples show first the call of the client indicating the current location of the user cssbct. The second was used to change the location and the third showed the changed response.

PROTOCOL FOR COURSEWORK

INTRODUCTION

Your coursework must implement the following protocol for communication between the client and server; this is based on a subset of features from the internet protocols for whois (RFC3912), HTTP/0.9 (w3.org), HTTP/1.0 (RFC1945) and HTTP/1.1 (RFC2616). Students should not invent their own protocols as the aim of the coursework is to demonstrate interoperability between different implementations as well as existing clients and servers. A working server that implements this protocol is provided by the lecturer on the machine `whois.net.dcs.hull.ac.uk`. You can initially develop a client using the lecturer's server, and later develop your server when you are happy with the correctness of your client. It is requirement of this coursework that your client can communicate with the reference server in addition to your own server in order to achieve successful client operation for assessment.

You will encode your data to be transferred between client and server as US-ASCII strings terminated by US-ASCII characters 13 then 10 (carriage return then line feed). There are other more readily available data encoding options in C# such as unicode and object serialization, but using US-ASCII makes our communication easily understandable by other non-C# applications and existing clients and servers. Fortunately the default action of network sockets is to do this anyway, so no special coding is normally needed.

The server is prohibited from using file storage when running unless the optional `/f` and `/l` features are implemented to specify the name and location of the files to be used.

PROTOCOL

1. The server waits for clients to contact it.
2. Clients should connect and communicate with the server via sockets and TCP packets.
3. Our protocol will operate over port 43, normally used by the internet whois protocol.
4. The protocol has four styles of commands representing the whois notation, and the HTTP 0.9, 1.0 and 1.1 styles.
5. The protocol contains two types of requests from the client to the server. These are lookup and update.

Whois style requests

1. A request from the client to the server for the server to query the database for a person's location looks like this:

   ```
   <name><CR><LF>
   ```

   Where <name> is the name of the person whose location the client is requesting.

2. A request from the client to the server for the server to update the database with a new location for a person looks like this:

```
<name><space><location><CR><LF>
```

Where: <name> is the name of the person whose location the client wants to update.
<location> is the new location for that person.

3. When the client requests the server to query the database for a person's location and the server finds an entry in the database for that person the server responds like this:

```
<location><CR><LF>
    [drops connection]
```

Where: <location> is the location if found in the database for the person.

4. When the client requests the server to query the database for a person's location but the server is unable to find an entry in the database for that person the server responds like this:

```
ERROR: no entries found<CR><LF>
    [drops connection]
```

5. When the client sends a request to the server to update the database with a new location for a person the server responds like this:

```
OK<CR><LF>
    [drops connection]
```

HTTP 0.9 style requests

1. A request from the client to the server for the server to query the database for a person's location looks like this:

```
GET<space>/<name><CR><LF>
```

Where <name> is the name of the person whose location the client is requesting .

2. A request from the client to the server for the server to update the database with a new location for a person looks like this:

```
PUT<space>/<name><CR><LF>
<CR><LF>
<location><CR><LF>
```

Where: <name> is the name of the person whose location the client wants to update and <location> is the new location for that person.

3. When the client requests the server to query the database for a person's location and the server finds an entry in the database for that person the server responds like this:

```
HTTP/0.9<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
<location><CR><LF>
    [drops connection]
```

Where: <location> is the location if found in the database for the person.

4. When the client requests the server to query the database for a person's location but the server is unable to find an entry in the database for that person the server responds like this:

```
HTTP/0.9<space>404<space>Not<space>Found<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
     [drops connection]
```

5. When the client sends a request to the server to update the database with a new location for a person the server responds like this:

```
HTTP/0.9<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
     [drops connection]
```

HTTP 1.0 style requests

1. A request from the client to the server for the server to query the database for a person's location looks like this:

```
GET<space>/?<name><space>HTTP/1.0<CR><LF>
<optional header lines><CR><LF>
```

   Where <name> is the name of the person whose location the client is requesting . The <optional header lines> are not required to be sent by the clients but other programs using HTTP 1.0 protocol may send them and they should be read and skipped by the server. A blank line marks the end of the header lines.

2. A request from the client to the server for the server to update the database with a new location for a person looks like this:

```
POST<space>/<name><space>HTTP/1.0<CR><LF>
Content-Length:<space><length><CR><LF>
<optional header lines><CR><LF>
<location>
```

   Where: <name> is the name of the person whose location the client wants to update and <location> is the new location for that person. The <optional header lines> are not required to be sent by the clients but other programs using HTTP 1.0 protocol may send them and they should be read and skipped by the server. A blank line marks the end of the header lines. The <length> is the count of the number of characters in the <location>.

3. When the client requests the server to query the database for a person's location and the server finds an entry in the database for that person the server responds like this:

```
HTTP/1.0<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
<location><CR><LF>
     [drops connection]
```

   Where: <location> is the location if found in the database for the person.

4. When the client requests the server to query the database for a person's location but the server is unable to find an entry in the database for that person the server responds like this:

```
HTTP/1.0<space>404<space>Not<space>Found<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
     [drops connection]
```

5. When the client sends a request to the server to update the database with a new location for a person the server responds like this:

```
HTTP/1.0<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
      [drops connection]
```

HTTP 1.1 style requests

1. A request from the client to the server for the server to query the database for a person's location looks like this:

```
GET<space>/?name=<name><space>HTTP/1.1<CR><LF>
Host:<space><hostname><CR><LF>
<optional header lines><CR><LF>
```

   Where <name> is the name of the person whose location the client is requesting and <hostname> is the host name of the server. The <optional header lines> are not required to be sent by the clients but other programs using HTTP 1.1 protocol may send them and they should be read and skipped by the server. The <hostname> line can also be ignored by the server. A blank line marks the end of the header lines.

2. A request from the client to the server for the server to update the database with a new location for a person looks like this:

```
POST<space>/<space>HTTP/1.1<CR><LF>
Host:<space><hostname><CR><LF>
Content-Length:<space><length><CR><LF>
<optional header lines><CR><LF>
name=<name>&location=<location>
```

   Where: <name> is the name of the person whose location the client wants to update and <location> is the new location for that person and <hostname> is the host name of the server.. The <optional header lines> are not required to be sent by the clients but other programs using HTTP 1.1 protocol may send them and they should be read and skipped by the server. The <hostname> line can also be ignored by the server. A blank line marks the end of the header lines. The <length> is the count of the number of characters in the "`name=<name>&location=<location>`" string.

3. When the client requests the server to query the database for a person's location and the server finds an entry in the database for that person the server responds like this:

```
HTTP/1.1<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<optional header lines><CR><LF>
<location><CR><LF>
      [drops connection]
```

   Where: <location> is the location if found in the database for the person.

4. When the client requests the server to query the database for a person's location but the server is unable to find an entry in the database for that person the server responds like this:

```
HTTP/1.1<space>404<space>Not<space>Found<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<optional header lines><CR><LF>
      [drops connection]
```

5. When the client sends a request to the server to update the database with a new location for a person the server responds like this:

```
HTTP/1.1<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<optional header lines><CR><LF>
     [drops connection]
```

In all examples above <CR> is US-ASCII character number 13, the carriage return character, and <LF> is US-ASCII character number 10, the line feed character and <space> is the US-ASCII space character. We are using both CR and LF together as our line terminator. The notation `[drops connection]` is used to indicate that the server closes the connection to the client at that point, without waiting for any response. In all cases both the client and server should give up and drop connection (timeout) if they are left waiting more than 1 second (1000 milliseconds).

The server should be able to handle enquiries from a number of clients simultaneously, however a client only needs to handle a single enquiry at a time.

ADDITIONAL CLIENT INTERFACE FEATURES

When initially constructed, the client, could have the address of the server hardwired into the C# code (such as whois.net.dcs.hull.ac.uk or localhost). This is not very useful, and the delivered client will need a parameter to specify the server to be used. The default address would be whois.net.dcs.hull.ac.uk if the argument is not used. For example:

```
G:\08241\> location /h 150.237.92.99 cssbct
cssbct is in RB-321
G:\08241\> location /h somepc.dcs.hull.ac.uk cssbct "changed room"
cssbct location changed to be changed room
```

Note that the sequence "`/h <servername>`" could occur at any point in the argument list, so it could be either before the username or after.

It will also be usful to be able to specify the port number to be used, to enable the client to connect to other servers on other ports. The `/p` flag will be used for this:

```
G:\08241\> location cssbct /p 43
```

It is also necessary to specify which protocol request form is to be used. By default the `whois` protocol form should be used. If an HTTP format is required then the `/h0`, `/h1`, `/h9` flags can be used:

```
G:\08241\> location cssbct /h1
```

When using these `/h1` means HTTP/1.1, `/h0` means HTTP/1.0 and `/h9` means HTTP/0.9 styles of server request.

OPTIONAL FEATURES

Implementing these features is not essential for the coursework, but may be useful in developing your client or server. They can earn extra marks in the marking scheme, but no one can get more than 100%, even if all the options are implemented.

Parameters of `/t` to change the timeout period (and setting the timeout period to zero could disable timeouts) and `/d` to enable any debugging might be useful additions to the client and the server.

The server could, at a minimum, do its job silently. It would be helpful, both for you to debug it, and for assessment, if it reported activity, either to the command window, or to a specially designed administration interface. For example:

```
G:\08241\> locationserver
request for cssbct
replied in RB-321
```

The more information that is recorded the better, and a good server should keep log files:

```
200.4.239.13 - - [28/Dec/2003:03:14:56 +0000] "GET cssbct" OK
150.237.92.99 - - [28/Dec/2003:03:15:23 +0000] "PUT cssbct some room" OK
150.237.92.99 - - [28/Dec/2003:03:19:24 +0000] "GET cshwvv" UNKNOWN
```

This example is shown in "Common Log Format".

The supplied server whois.net.dcs.hull.ac.uk keeps a log of all activity, which can be checked for debugging your client.

If your server is able to handle more than one simultaneous client enquiry at a time, you need to be careful that you do not attempt to write more than one log entry to the file at the same time. Further, you should ensure that if the log file is not writable that the server does not fail. When the programs are assessed, the server will not necessarily have write permissions to the machine to save a log. You should also make the destination of the log file an optional argument to the server to allow for different machine file structures, such as:

```
G:\08241\> locationserver /l c:\log\logfile.txt
```

It is also useful to be able to save the server database when it exits to some form of file, and re-load it when it restarts. You could use the option `-f` for this feature:

```
G:\08241\> locationserver /f G:\08241\locations.txt
```

If you have implemented all your client and server features properly, then it will be possible to use your client to talk to a web server, or get a web browser to talk to your client. For example:

```
G\08241\> location /h www.hull.ac.uk /p 80 /h9 index.html
```

might fetch a web page. You can also try using Intenet Explorer on the URL `http://localhost:43/cssbct` to do a database lookup from your server.

There are also further features of the standard protocols that could be implemented, but are not necessary for this assignment. Students may wish to read those standards.

DELIVERABLES

You must submit the following:

An electronic copy of your source code, compiled class files of both applications as two distinct visual studio projects compiled into a ZIP file. This must only be a ZIP file and has the file type ZIP. No other form of archive or compression should be used (i.e. no RAR or 7z etc). The submission should also contain a file, not within a folder, named `readme.txt` which contains the summary of the zip file contents.

The client and server code may be contained within folders. If the client code is delivered in a folder, perhaps as a visual studio project, then this folder should be called `location`. If the server code is delivered in a folder, perhaps as a visual studio project, then this folder should be called `locationserver`. There must be only one copy of the client and server source and executable contained within the zip file submitted, otherwise it would not be clear which one should be marked.

TIMESCALE FOR COMPLETION OF WORK

This timetable shows how the extensive coursework assessment can be completed, by making regular progress. Initially you would want to call your programs manually, but you may wish to also try the test scripts which show how your programs can be called from other code for testing.

Step 0. The first laboratory task introduced you to the sample client used in the lectures. You may need to check about the method `stream.flush` which is not included in the example code.

Step 1. Develop and complete your client to perform a whois request by having it communicate with the working server provided by your lecturer (whois.net.dcs.hull.ac.uk) or an external third party internet whois server. This should be based on the code template given in the lecture powerpoint. It is often sensible to be sure of the correctness of your client before proceeding to develop your server. It might be a good idea to start using the SVN server to keep track of the development of your solution.

Step 2. Develop a minimal server based on the template in the powerpoint slides that uses the whois protocol, using your own client to interface with it. You would need to write code to store the person locations in the server memory, and you might find the classes for `Hashtable` or `Dictionary` particularly useful for this. We suggest that you initially develop a server that does not use threads, to ensure the protocol and communications work properly. This can then be augmented to handle multiple clients and multiple protocols later.

Step 3. Extend your basic working client to include the command line arguments of `/h`, `/p`. Add the HTTP protocols and the arguments `/h0`, `/h1`, `/h9` and test against a web server.

Step 4. Extend your basic server to recognise HTTP protocol requests and test with your client and later with a web browser.

Step 5. Upgrade your server to include handling of multiple threads so concurrent enquiries from multiple clients is possible. If you find this difficult to achieve, remember to retain a copy of your working non-threaded server to hand-in for assessment. Do not damage your only working server when trying to add more capability, or you risk losing all the marks you earned to this point. You might find that using SVN from the first week has become valuable is assisting you in recovering earlier versions!

Step 6. Add any remaining optional features to your client and server, such as logging, and finish testing. Create the ZIP file and `readme.txt` for submission for marking. Check that you can locate the Canvas submission page.

Part 2– Live web interface

In Part 1 of this assignment, you programmed your own simple command line client and server using the HTTP protocol to implement a simple person locator. In part 2, you will build on the knowledge you have gained, by programming similar interactions with server side scripting on a full scale webserver. You will also expand the functionality, to turn a simple command-line system into a usable live website for recording locations and finding people.

Requirements

You are to write a web application to provide greater functionality for the person location service. In addition to storing the current location of each person, it must also keep track of each person's previous locations. Your web application must implement the following:

1) A database in which to store locations and personal details of users
    a. For each user, the database must store a username (which must be unique), a first name and surname.
    b. Each time the user's location is updated, the database must store the new location and the date and time at which it was recorded. Old locations and timestamps should not be removed from the database.

2) In the root folder of your RDE website, there must be a page called 'location.php' OR 'location.aspx' which must accept GET and POST requests. GET requests to this web page must return the most recent location of the person in question from the database, in the same format as the server you wrote in part 1. It must not return HTML as a result of a GET or POST request. POST requests to this page url must allow a person's location to be updated. On receiving a valid POST, your website must update the database record for that person. If the person does not exist in the database, they must be added. Note that PUT requests need not be supported.

3) In the root folder of your RDE website, there must be a simple index page called 'index.php' or 'default.aspx' from which the following human-readable pages (which you must also create) can be accessed:
    a. A page which will show the current locations of all staff
    b. A page which will allow a user to change the location of a staff member
    c. A page which will allow a user to edit the personal details of a staff member
    d. A page which will allow a user to choose a staff member and list their locations for the last 24 hours
    e. A page which will allow the user to select from a list of all locations used by people and show a list of the people currently in the selected location

4) All data entry in your web pages must be validated to prevent the user entering faulty values.

Request formats

Your web application must respond to GET and POST requests sent in the standard HTTP request formats. Sample GET and POST requests are shown below (note that these are only examples to show the format – the data in them will need to be different for your system).

Typical POST request:

```
POST /123456/location.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: webdev.net.dcs.hull.ac.uk
Content-Length: 22

jsmith14=in+a+meeting
```

Typical GET request:

```
GET /123456/location.php?person=jsmith15 HTTP/1.1
Host: webdev.net.dcs.hull.ac.uk
```

Development tools

You may use either PHP or ASP.NET to implement your website. You may use javascript in your web pages. You may use the jQuery library if you wish, but apart from that you may not use any third-party libraries (including Bootstrap) or controls other than those provided in the standard University installation of Visual Studio and RDE. You may develop your web site in any environment you like, but it must be deployed in your RDE web space and must use your RDE database. All assessment will be based only on this installation.

Note that presentation of your web pages does not carry very many marks. While you are free to style your pages how you wish, you are advised to use your limited time on ensuring your web site's functionality and robustness rather than concentrating on a shiny user interface.

Database files

Your web application must store data in a database about the staff whose locations it is tracking. You must use your RDE SQL Server instance to store the database. At the very least, this database must store the user name, first name, surname, locations and times of update for each person. You may find it worthwhile to store other information as well.

Deliverables

By the submission deadline, you must submit via Canvas, a single .ZIP file containing all files required to create your web site, with the correct folder structure . It must be possible to simply unzip this file into an empty web space to recreate your web site. Note that Visual Studio solution files are not required. Do not submit files in any other archive format than .ZIP.

You must also install on the RDE server a functioning version of the same files you have submitted via Canvas. The demonstration will be based on your files in the RDE server.

Important notes

1. HTTP server: For this part of the ACW, you will not be using the HTTP server you wrote earlier. You will be using a commercial web server (Microsoft Internet Information Server). You only need to write scripts and web pages which control its behaviour, as you have done in the laboratory tutorials. You may write these in ASP.NET or PHP. Both are supported by the server.

2. HTTP Client: You will not be able to use the client you wrote in the first part of this ACW with the commercial HTTP server, because your client does not implement the required authentication methods. You may use the HTML client developed in the first web technology lab as shown below (for week 7).

3. Precision: Some of the assessment will be assisted by automated tools. This simulates the situation in the real world where your software forms a component of a larger system, and the 'user' is in fact another software component. This means that tests will fail (and you will lose marks) if you do not follow the defined specification. Where the specification tells you to give a file a particular name, do as it says. Where a particular output or input format is defined, make sure that you use it. In the real world, failure to do so will mean that the software relying on yours will not be able to work. In the context of ACW, it will mean that you will lose marks (even if your software works). This is an easy trap to avoid, so don't fall into it.

4. If you wish to use your laptop or home computer to develop on, you can do so. Just remember that you will not be allowed to use either for the assessment. Data must be stored in your RDE database, and your web pages must be located on your RDE web server. You can use these from off-campus locations if you

use the University's VPN client. If you develop from home, make sure that everything works on campus well before your demonstration time.

Mode of working

This is an individual assignment. Though you are encouraged to discuss methods and problems with colleagues, you may not share code with other students. During assessment, you will be expected to be able to explain any part of the code to the assessor's satisfaction.

The following schedule of development is suggested, which is supported by the laboratory exercises:

Week 7. Ensure you have created your RDE site and add a suitable `index.html` page and view it using the URL http://webdev.net.dcs.hull.ac.uk/yourID/. This should have already been done in the initial lab exercise for this module, but doing it at this stage checks that your permissions are working correctly. You should now create some subsidiary pages and link them from the main page to check you can create and use web links that you will need later. Make a simple web enquiry form called `location.html` that does a GET request to the whois.net.dcs.hull.ac.uk:43 server or localhost:43 server so that you can make enquiries to servers from web pages. This will be useful when you later test your PHP or ASP implementation. Note, that later you will need to remove or rename the index.html file when you create your index.php or Default.aspx files.

Week 8. Create an SQL database to store usersnames and the users locations on the RDE server, following the lab instructions. You may try adding to your basic HTML client to provide POST requests and better functionality, like selecting servers. This may require some Javascript.

Week 9. Create an location.php file to contain code to lookup a user in your SQL database using an HTML form to provide the data for a GET request, following the example from the lab instructions. Add the ability to update the information using a POST request, also with data from an HTML form.

Week 10. Create a location.aspx file to contain code to lookup a user in your SQL database using an HTML form to provide the data for the GET and POST requests, following the examples from the lab instructions. Don't forget to install the necessary files from Visual Studio to your RDE folder location to ensure they operate properly.

Week 11. Decide if you want to complete your assessment using PHP or ASP now you have completed exercises in each. Removed or rename your initial index.html file and create an index.php or Default.aspx file, as appropriate for your chosen method. Expand your SQL database to store any extra information, such as update date and time and so on. Implement the subsidiary pages which permit the 5 specified enquiries to the database.

Week 12. Ensure your pages are installed properly, test the various functions of the different pages. You may fine the location.html page that you made earlier useful in testing the GET and POST functionality. Bundle your web site into a ZIP and submit the assessment.

Assessment

The assignment will be assessed by demonstration in the lab. You will be notified of your demonstration time. You will be expected to show all the required functionality of your web site using only a web browser (without Visual Studio or any other IDE). Your web site will be assessed based only on the installation in your RDE web space using your RDE database. You may not demonstrate it running from a laptop or any other server. If your system does not work on the university RDE web server and database at the time of demonstration, you will lose marks.

Feedback

You will receive individual verbal feedback on your work when it is demonstrated.