G52APR Coursework 2 (2014/15)

Please also refer to the specification document for Part 1 of the coursework.

Part 2 – Parsing HTML and File Server (30%)

This part of the coursework comes in two sections and still two Eclipse projects. You should by now have two projects in Eclipse, one for the client and one for the server. The first section of this part (2a) concerns parsing some HTML from your client, and so should be completed in your client project. The second section (2b) builds on the server which you made previously and so requires changes in your server project.

Part 2a – Parsing HTML

For this part you should create a set of tests in your client project. You will run these tests against the server provided. After you have completed all of part 2 (Including 2b) you should be able to run these tests against your own server. You will probably already have a main method for your client (hopefully in a different class from your G52APRClient class!). You should create a new class that will take in a G52APRClient object and run a series of tests on it.

We expect something like this for your test class to start:

```
Public class ClientTests {

        public ClientTests(G52APRClient client) {

        }

        public void runTests() {

        }

}
```

Which will enable you to run these methods from your main:

```
ClientTests ct = new ClientTests(new G52APRClient());
ct.runTests("http://cs.nott.ac.uk/~syn/webfiles/index.html");
```

This allows you to easily change from where your test is retrieving data. It would be sensible to also visit the page in your browser yourself to understand what kind of results you may expect.

Your client will need to run a series of tests and print the output using System.out.println. It is expected that this output will be used only for the tests, and thus you should not do any other printing to the console. It is important that you consider the efficiency of many GET requests and think of ways to reduce the number of these. Given you can minimise the number of GET requests you should then try to minimise the number of other requests needed.

Your tests should do the following:

1- Your HTTP Client should visit http://cs.nott.ac.uk/~syn/webfiles/index.html search the response for links to other html pages. This requires you to parse the HTML which is returned.

Hint: A link tag looks like this:

 Name of link

2- Your HTTP Client should check the availability and the last-modified date of each linked page.

3- Your HTTP Client should then retrieve each existing page which has a last-modified date within the last 6 months. For these pages your client should retrieve the page, find any html headings and print out all of the headings in order.

Hint: Headings will all start with one of three tags <h1>, <h2> or <h3>, and will close with a closing tag </h1>, </h2> or </h3>

Your output should be in the following format:

Your output should to be in this format (unless the page does not exist or was not edited in the last 6 months in which case see the examples below):

<Link URL>

<Last Modified Date>

<Header>

…

<Last Header>

Ignore the angle brackets which are only part of the specification documentation!

If the link is broken or the page is too old then:

<Link URL>

<message>

Example output from your tests (Page exists):

http://www.cs.nott.ac.uk/~syn/webfiles/page1.html
Tue, 30 Oct 2012 13:09:57 GMT
G52APR Web Resources
Things to do on a weekend
Coursework
Dine at a nice restaurant

Example output from your tests (Page does not exist):

http://www.cs.nott.ac.uk/~syn/webfiles/page1.html
This is a broken link.

Example output from your tests (Page last modified over 6 months ago):

http://www.cs.nott.ac.uk/~syn/webfiles/page1.html
Page last modified over 6 months ago.

Part 2b – File Server

For this part of the coursework you should write a file server to be used to retrieve files from local storage. The fileserver will be used by your existing HTTP webserver to implement the GET, HEAD and POST methods.

You will first need to add the IServe interface and exception classes from the G52APR Moodle page to your server project. You should create a new class for your fileserver which implements IServe. The interface has javadoc comments explaining the parameters and return values needed for each of the methods. The exception classes also have javadoc comments which you should read. You will need change your existing server request handler methods to utilise your fileserver in order to provide valid and meaningful responses rather than the current 501 status codes.

The Berners-Lee HTTP/1.0 protocol specifies that the behaviour for POST may differ from one server to another. "The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI." For our server we wish to keep this simple. If the request URI does not exist, it should be created simply with the content of the body as the content of the file, the response should include the 201 status code (created). If the requested URI does exist the response should include the 200 status code and contain the body of the file (this is the same as the GET method).

You will need to specify the root directory for your server to get the files from. To do this, your server (G52APRServer.java) should contain a main which now takes in two (2) arguments. The first argument should be the port number, and the second should be the directory which will be the root of the file store. For example: java –cp <classpath> G52APRServer 4444 C:/files/wwwroot/

This means that if your server gets a request for /index.html, it should read the file: C:/files/wwwroot/index.html

You should not hard code this directory into your server, if you do, your functionality tests are likely to fail.

For testing, there is a ZIP file of html documents available on the Moodle page which you can add to your local storage for your fileserver to use.

It is expected that you will include headers for Date, Last-Modified, Content-Length (If appropriate) and Content-Type for all responses.

Once you have completed this section, if your implementation is well made, it is possible to point your browser to http://localhost.com:4444/filename (Where 4444 is the port number you are using and filename is the file you are wishing to obtain), and view the pages you have on your HTTP server.

Hint: At this point you should be able to further test your HTTP server. You can send POST requests followed by HEAD and GET requests to test these methods. You can test using both the client that you made, and a real browser (Firefox, Chrome, Safari etc). You may choose to use tools such as firebug to show the requests for testing and debugging. There are many more ways to test both valid and invalid requests to your server, you should be doing this and documenting how you have tested your file server. You will be marked on the efficiency of your server, how well it works, proof of testing and the style of your design and code.

Marks

Each part of the coursework carries a weighting. All of the sections of part 2 will add up to 30% of your final G52APR grade.

Assessment Criteria:

This is similar to part 1 of the coursework.

Submission

You should submit a client zip file and a server zip file as in part 1. The server should include a report of up to 500 words, as a text file, explaining how you have tested the Server (see “hints” above). You should NOT produce a report for the client for this part of the work.