PROG2001 – WEB DESIGN AND DEVELOPMENT

A-06 : MY OWN WEB SERVER

OVERVIEW

In this assignment, you (and your partner if you wish) will be investigating web development from a different perspective. This assignment requires that you design and develop your own simple, single threaded **web server**.

This is a partner-based assignment and you can have partner if you wish.

OBJECTIVES

This assignment supports the following course objectives:

- To demonstrate how a web server responds to requests using the HTTP protocol
- To better understand the parsing and creating of the underlying HTTP messages protocol used between a browser and a web server

ACADEMIC INTEGRITY AND LATE PENALTIES

- Please refer to the SET Policies document regarding Academic Integrity Information
- Please refer to the SET Policies document regarding Late Policy

EVALUATION

- Please refer to the assignment weighting in the *Instructional Plan* for the course as well as the assignment's Rubric in the course shell.

PREPARATION

Review Module-09 lesson content as well the module's code samples – they will help you in this assignment.

REQUIREMENTS

1. The server must be written in C# as a console application built upon a .NET framework (i.e. not on .NET Core technologies). Your server has three mandatory command line arguments:
   - `-webRoot` : which will be set to the root folder for the website data (html, jpg and gif files)
   - `-webIP` : which will be set to the IP address of the server
   - `-webPort` : which will be set to the Port number the server will be listening on
2. Ensure that your server when compiled creates an executable called `myOwnWebServer`. So that when you invoke the server, you would enter a command like:
   ```
   myOwnWebServer -webRoot=C:\localWebSite -webIP=192.168.100.23 -webPort=5300
   ```
3. All incoming and outgoing functions supported by the server must comply with the HTTP/1.1 Protocol Specification.
4. The server is to be single threaded, so only 1 browser session needs to be supported at a time.
5. The server only needs to support the **GET** method in any incoming requests. [*But that doesn't mean that I won't try to test it with the POST method*]
6. The server only needs to support the returned content types of plain **text** (specifically the .txt extension), **HTML** files (and their various extensions), **JPG** images (and their various extensions) and **GIF**.
7. The server must be able to parse the incoming HTTP request, and act accordingly. For example - If the request is for a text-based resource (e.g. `default.html`) and this file exists, then the server must be able to open the file, read the file contents and send its contents as a properly formatted HTTP response.
8. If an exception or error condition that arises in the server – you need to ensure that the proper HTTP status code is returned to the client as part of the response.
9. Besides having the mandatory first line in the response header, your server should also populate and return the following items in response header lines : the **`content-type`**, the **`content-length`**, the **`server`** and the **`date`** elements
10. **You are not allowed to use** certain *helper classes* in the `System.Net` namespace.
    - Specifically, you cannot use any of the `System.Net.Http*` set of classes (where * is a wildcard – so basically any helper classes beginning with "`Http`")
    - You are free to use any other classes defined in the System.Net names (e.g. `TCPListener`, `Sockets`, etc.)
    - The goal of this assignment is to have you parse the incoming request, and formulate the full response header programmatically in your server code.
11. Similarly, when it comes to parsing the incoming request and determining the ***outgoing MIME type*** – **you are not allowed to use** the `MimeMapping` class in the `System.Web` namespace.
    - We're going old school! ☺
12. The incoming request for all resources will be guaranteed of being in the server's root folder (or a sub-folder found within it). [even request for non-existent resources]
13. Make sure to comment your classes, methods and code as per the SET Standards.

14. Following best practices when creating any type of server-based application – ensure:
    - That the server outputs **absolutely nothing to the console window** it is running in
    - That the server generates a text-based log file:
        i. The name of the log file should be `myOwnWebServer.log`
        ii. You will produce one log entry for each incoming **`Request`**
        iii. You will produce one log entry for each outgoing **`Response`**
        iv. You will generate a log entry when the application starts
    - Please see the *Additional Notes* section below for log entry formatting requirements.

Additional Ideas:

- Feel free to use the **HTTPTool** (supplied in the course content) to exercise and debug your server, but don't forget to also test your server with both the Internet Explorer v11 and Chrome browsers.
    - Since your web server will not be running on the default port 80 (because your IIS instance is using that port) – you will need to surf to your web server a little differently.
        - For example, let's say you are using the browser and trying to retrieve a file called `index.html` from your server which you've launched and is using port 5300 – you would enter the URL `http://localhost:5300/index.html` to do so.
    - If you wanted to check to see that your image handling is working properly and you had an image called `myImage.jpg` – then enter the URL `http://localhost:5300/myImage.jpg`
- Make sure that you get your MIME types correct in your response HTTP headers – check out this site for a complete list
- Make sure that you understand each of the different HTTP Status Codes and what they mean … here is a good starting reference for you …

FILE NAMING REQUIREMENTS

The only naming requirement is that your VS Solution creates and compiles an executable called `myOwnWebServer.exe`.

SUBMISSION REQUIREMENTS

When submitting your solution to this assignment, hand-in a single ZIP'd file containing:

1. Your cleaned Visual Studio solution.
    a. Do not include your own test files – a set of pre-built test files will be used in the marking process.
2. Also remember that this solution will be tested using Internet Explorer v11 as well as Chrome
3. Please ZIP up these files and submit to the appropriate eConestoga Dropbox by the deadline
    a. Please give your ZIP submission the filename *lastName-firstInitial.zip* (e.g. if you are Sally Jones – then your ZIP should be named `jones-s.zip`
    b. If you are working with a partner, then include both your names in the ZIP filename (e.g. if Sally Jones is working with John Smith – then your ZIP should be named `jones-s-smith-j.zip`

NOTE: If working with a partner, only one partner need submit the solution

ADDITIONAL NOTES

As mentioned, your web server needs to keep track of its activities in a text-based log file. Here are the requirements to keep in mind while generating your log file

1. Each time the `myOwnWebServer` application starts it can create/overwrite any existing `myOwnWebServer.log` file that may exist
2. You can write the log file into the same directory as the `myOwnWebServer` application
3. Each message written into the log file must be date and time stamped in the format as follows
    a. `2021-11-15 14:05:00 <`*`log entry goes here`*`>`
4. Ideally and according to best practices – **each message** written into the log file – **should be** written as a **single line**.
    a. This is done regardless of how long the log entry is … this way, each line in the log file represents a different activity/event. And every line in the log file begins with a date/time stamp
5. I also want each of your log entries to indicate the event that is being logged
    a. As mentioned above, you need to log the **START** of your application as well as each **REQUEST** and **RESPONSE** to and from the server
    b. So a good idea might be to format your logging messages neatly (as follows) to include the event

        `2021-11-15 14:05:00 [SERVER STARTED] - <`*`rest of log entry goes here`*`>`
    c. Starting the web server
        i. Write an entry in your log file when the `myOwnWebServer` application starts. This log entry must include the values for the command-line switches (i.e. indicate the `webRoot`, the `webIP` and the `webPort`)
    d. Receiving a Request
        i. For each request that your web server receives, I want you to write an entry in your log file that indicates both the HTTP Verb that is being used in the request (i.e. GET, etc.) as well as the resource (file) that is being requested (e.g. `/myTest/sample.html`)

- e. Sending a Response
  - i. When your web server issues a **<u>successful</u>** response generation (i.e. when the **HTTP Status** value is **200**) – I want you to write an entry in your log file that contains only the *content-type, content-length, server* and *date* values that you would have placed in your HTTP Response header. **Do not include the HTTP Response body** (i.e. do not include the content of the resource/file that is being sent back) – I only want the successful response log message to include the HTTP Header attributes
  - ii. When / if your web server **<u>finds / detects and error</u>** and is sending back a response where the **HTTP Status** value is **not 200**, then I want your log entry to only indicate the outgoing HTTP Status value

6. Writing messages to the program's console window
   - a. Should **<u>not be done</u>** (generally) …
     - i. But if there is sort of exception or error that you detect / find / catch in your web server **<u>and</u>** that error /exception is something that prevents your web server from running any longer (i.e. an ***unrecoverable*** error / exception) - then
       - You need to log the error / exception to your log file for sure
       - Best practices would also indicate that this is a time when it is okay to write an error / exception message to the console window – so that the user can see it when they realize the web server has exited.