CSCI 4131 – Internet Programming Assignment 4

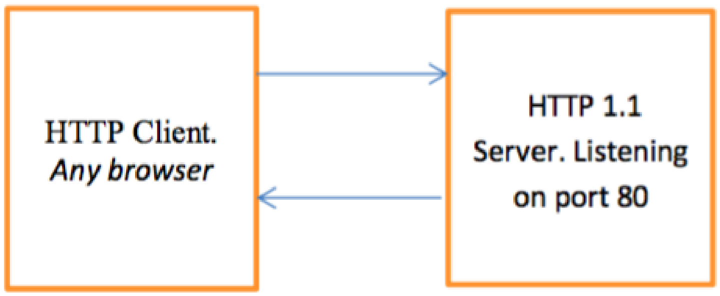
DUE DATE: Friday October 22nd at 11:59pm

Late Submissions accepted until Sunday October 24th at 5:59am (morning) with penalty (see the syllabus for details on late submission penalty)

1 Description

The objective of this assignment is for you to learn the Hyper-Text Transfer Protocol (HTTP) and build a small subset of an HTTP server in Python 3. In this assignment, using Python 3 and TCP sockets, you will program some of the basic functionality of an HTTP (Web) server. You will need to go through RFC 2616 for HTTP 1.1 protocol details. This assignment specification is 11 pages long.

When a web client (such as Google Chrome) connects to a web server (such as ***https://www.google.com***) the data that is exchanged between them (e.g., HTTP messages, HTML, CSS, JavaScript, pictures, audio, video, etc.) is transmitted using the Hyper-Text Transfer Protocol.



An outline of how a basic HTTP server works for an HTTP request message is specified below.

1. An HTTP client connects to the HTTP-server and sends an HTTP request message requesting a resource to the HTTP server.
2. The request can be of type HEAD, GET, POST, etc.
3. The server parses the request header fields.
4. For a GET request, the server identifies the requested resource (e.g., an HTML file) and checks if the resource exists and if it can access it. If this is the case, the server proceeds to step 5 below. Otherwise it proceeds to step 6 below.
5. The HTTP server then generates an appropriate HTTP response message. If the requested resource is found and is accessible, the HTTP server reads in the resource and builds a response message. The response includes successful (2xx) status code in the response headers along with other metadata such as the Content Type and Content Length, and includes the resource data as the message body. The server then sends the message to the HTTP client (e.g., a browser, such as Chrome) which sends the GET request.
6. Otherwise, if the resource requested by the HTML client is not found, then the HTTP Server composes and sends a response message with an error response status code to the HTTP client (e.g., a browser). See section 4.3 below for a discussion about the errors your server must recognize and respond to (i.e., compose a proper response message and send to the HTTP client).

2 Preparation: Required and Provided Files

You will need following files for this assignment:

- ***403.html*** - this file should be sent to the client if permissions do not permit its access (Provided).
- ***404.html*** - this file should be sent to client if the server cannot find the requested file (Provided)
- ***private.html*** - this file is the private file that triggers 403 forbidden code, you can use it to test.
- ***MySchedule.html*** – use your own MySchedule.html file from Assignment 3. (not provided)
- ***MyForm.html*** – use your own MyForm.html file from Assignment 3. (not provided)
- ***MyServer.html*** – html file containing links to the following webpages. (File provided).
- ***Coffman.html*** – html file containing an image to use for testing your server's capability to respond to a request for an image (Provided).
- ***OuttaSpace.html*** – html file containing an audio controls element to use for testing your server's capability to respond to a request for an audio file (Provided).
- ***Coffman_N_OuttaSpace.html***– contains both an image and audio control element to use for testing your server's capability to respond to a request for an image and an audio file (Provided).
- ***coffman.png*** – the image file (a .png file) used by Coffman..html and Coffman_N_OuttaSpace.html (Provided).
- ***OuttaSpace.mp3*** – the audio file (a .mp3 file) used by OuttaSpace.html and Coffman_N_OuttaSpace.html (Provided).

We have provided files listed above in the file named: **Hw4Resources.zip**

You will use your form page to test your server's ability to successfully handle a form that uses a post method.

You are required to test your solution on a Linux or Unix machine such as the Department of Computer Science Computers running the Ubuntu OS to ensure you test your solution on files with permissions set.

To give the files above proper permissions, execute following `chmod` commands to correctly set permissions on your files after downloading them to your folder:

```
chmod 640 private.html
chmod 644 403.html
chmod 644 404.html
chmod 644 MySchedule.html
chmod 644 MyWidgets.html
chmod 644 MyServer.html
chmod 644 Coffman.html
chmod 644 OuttaSpace.html
chmod 644 Coffman_N_OuttaSpace.html
chmod 644 coffman.png
chmod 644 OuttaSpace.mp3
chmod 644 MyForm.html
```

We have provided the executable Python 3 files `EchoClient.py` and `EchoServer.py` which you are free to use and refactor in order to construct your server.

Finally, we also provide a skeleton of a python server in `myServer.py`. You are free to use the skeleton of the python server code we provide as a starting point for your server. In any case, you should review it carefully because describes much of the requirements within comments. ***Do not rely on the skeleton to provide prescriptive requirements.***

Note, you should also change the permissions on any directories/ folders or files used by your form to **644** (that includes external JavaScript and CSS files used by your form or other pages)

3 Functionality

When you start your server, it will establish a socket and bind to a port, listening for connections. You can send a request to the server to get your Schedule from your web browser by typing:

```
http://<host>:<port>/MySchedule.html
```

in the browser's address (i.e., URL) bar. For this assignment, when developing and testing your server, you will run the server on your local machine, so `<host>` will be `localhost` and `<port>` should default to `9001`.

Also, you are required to use python 3 to develop and test your server – use of frameworks that generate Python, like Flask, are not permitted. Moreover, you are not permitted to use python 3's http.server module, or Python's SimpleHTTPServer, or any Python module that provides HTTP functionality to construct your server. Any we code provide is permitted.

When you run your server on Unix and Linux Operating Systems your server should bind to port 9001 and serve requests. Note, if you are developing and testing on Windows OS, and you run into errors, bind to another port (for example, 5050). Regardless, ensure it binds to port 9001 upon submission. An example call to start your server on the CSELabs computers is:

```
python3 myServer.py
```

4. The HyperText Transfer Protocol (HTTP)

HTTP is a protocol of non-trivial size. You will only be implementing a small, functional subset of HTTP GET and POST requests in your HTTP server. Your server should also recognize and respond to a limited subset of errors discussed in section 4.3 below.

4.1 GET Requests

GET requests are the most commonly used HTTP requests. For example, if you enter the following address in your browser's address bar:

http://localhost:9001/MySchedule.html

the browser will issue a GET request to the server to fetch MySchedule.html file from the directory in which the server code resides. Below is an example get request received by your server:

GET /MySchedule.html HTTP/1.1
Host: localhost:9001
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:33.0) Gecko/20100101 Firefox/33.0
*Accept***:** *text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

In this assignment, the resources that a client can request include an html file (.html), an image file (.png), an audio file (.mp3), JavaScript (.js), or css (.css). When such resources are requested in the request message, your server should identify the resource type and return the resource via an http response message. The resource type can be identified via the suffix of the resource name (e.g., the resource type of MySchedule.html is html and the resource type of OuttaSpace.mp3 is mp3).
Note that your html file (e.g., MySchedule.html page) may contain links to external .css and .js files. If it does, then to properly load and render the MySchedule.html page, your browser must request (via HTTP requests) the external css and js files as well. So, for testing, you may embed your JavaScript and CSS from the external files in your mySchedule.html file.

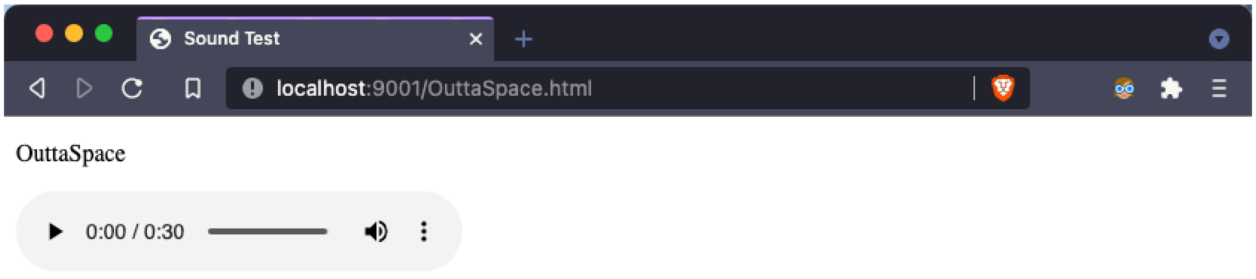
As specified above, your server should be able to access the image and audio resources and return them to the browser as part of a GET request.

To test your server's ability to return audio and image files, we have included the following files:

i) `Coffman.html`
ii) `OuttaSpace.html`
iii) `Coffman_N_OuttaSpace.html`
iv) `MyServer.html`

that you should "get" by issuing a **get** request to your server via your browsers address (url) bar.

More specifically when you type: http://localhost:9001/OuttaSpace.html
in your browser's address bar, it will request the file *OuttaSpace.html* from your server. Your server should return the contents of the file *OuttaSpace.html*, and your browser window should display the Web page displayed below (on the next page):

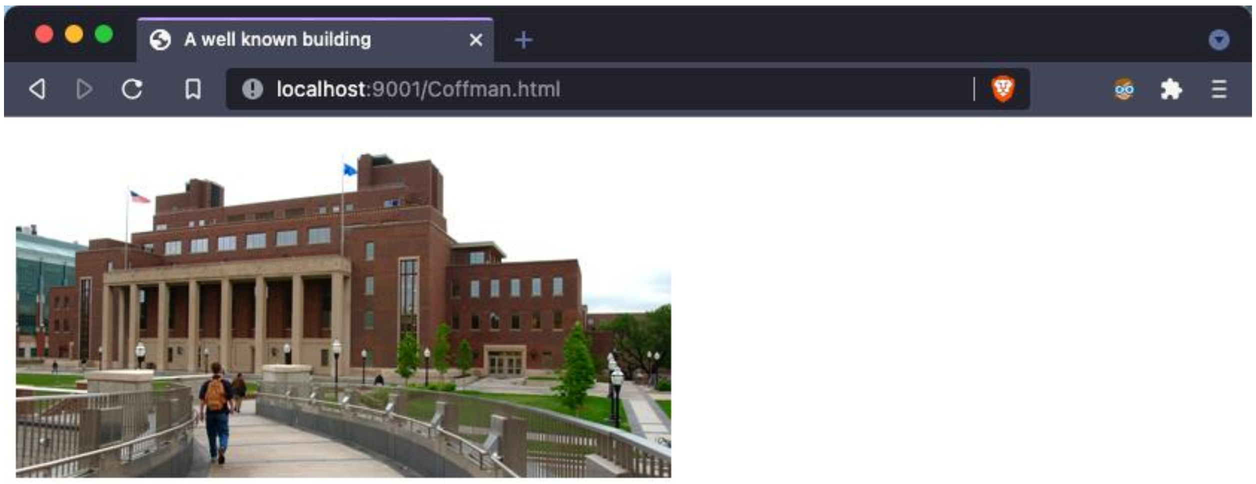


After the page is displayed, you should be able to play the Audio.

Next, when you type:

http://localhost:9001/Coffman.html

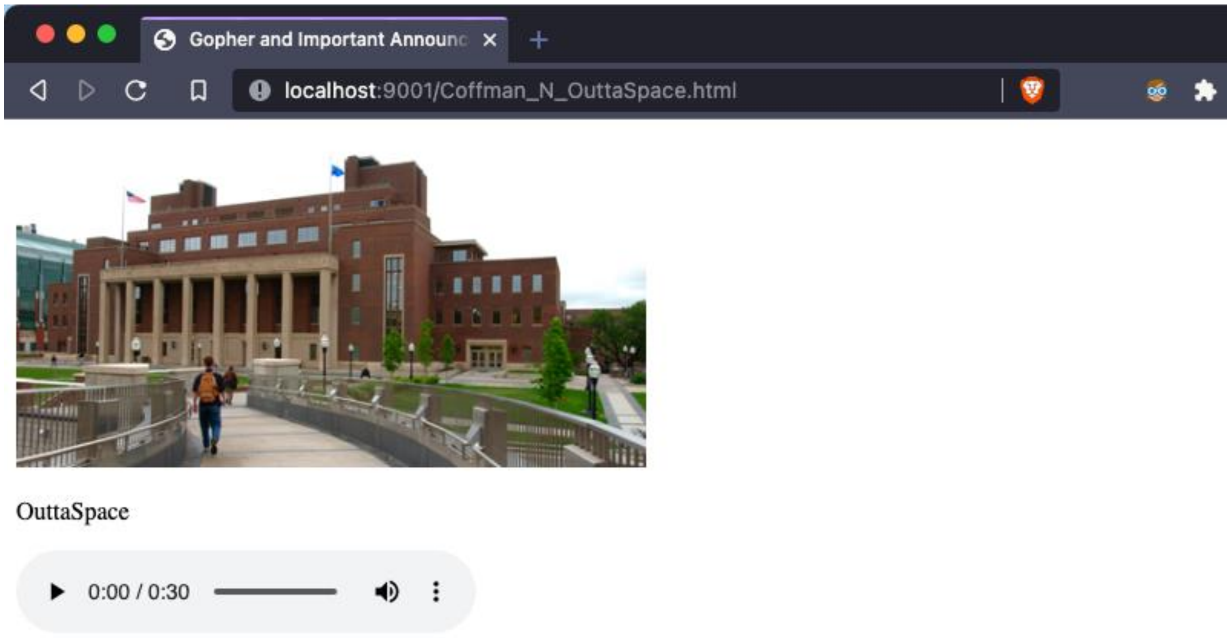
in your browser's address bar, it will request the file *Coffman.html* from your server. Your server should return the contents of the file *Coffman.html*, and your browser window should display the following web page (shown on the next page):



When you type:

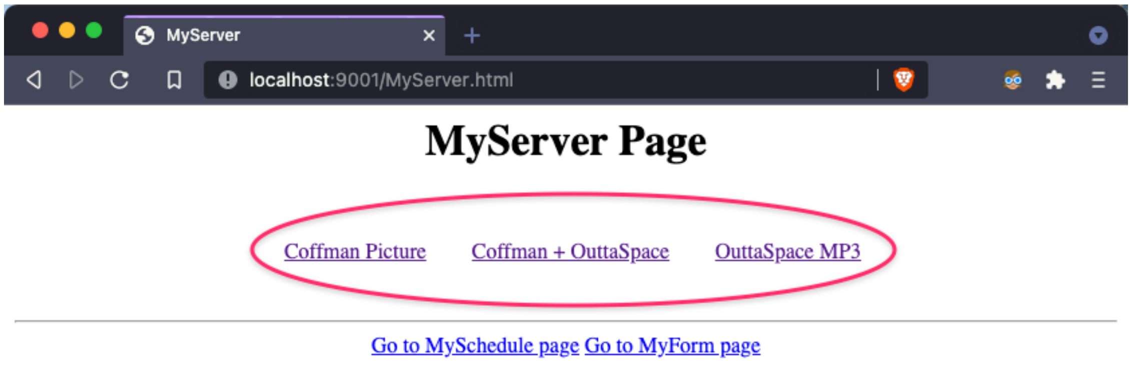
http://localhost:9001/Coffman_N_OuttaSpace.html

in your browser's address bar, it will request the file *Coffman_N_OuttaSpace.html* from your server. Your server should return the file *Coffman_N_OuttaSpace.html*, and your browser window should display the following web page:


Gopher and Important Announc
localhost:9001/Coffman_N_OuttaSpace.html
OuttaSpace
0:00 / 0:30

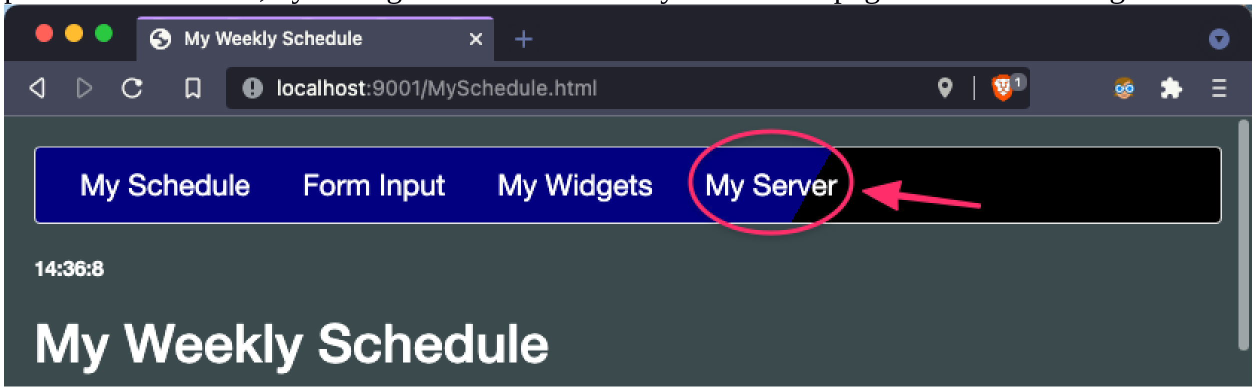
You should be able to play the Audio by selecting the Play button.

Finally, to make testing of these three pages easier, we have included an MyServer.html page. When you type: http://localhost:9001/MyServer.html in your browser's address bar, it will request the file *MyServer.html* from your server. Your server should return the contents of the file *MyServer.html* to the browser, and the browser should display the following web page:



By clicking the indicated links (red circled) on this webpage, you should be able to access the three web pages above i.e., *Coffman.html*, *OuttaSpace.html* and *Coffman_N_OuttaSpace.html*.

To make navigation between your webpages easier, extend the navigation bar from your previous homework, by adding a new link to the MyServer.html page as shown in image below:



Clicking on this link should take you to the MyServer.html page that we provided (you are free to restyle it as you deem necessary).

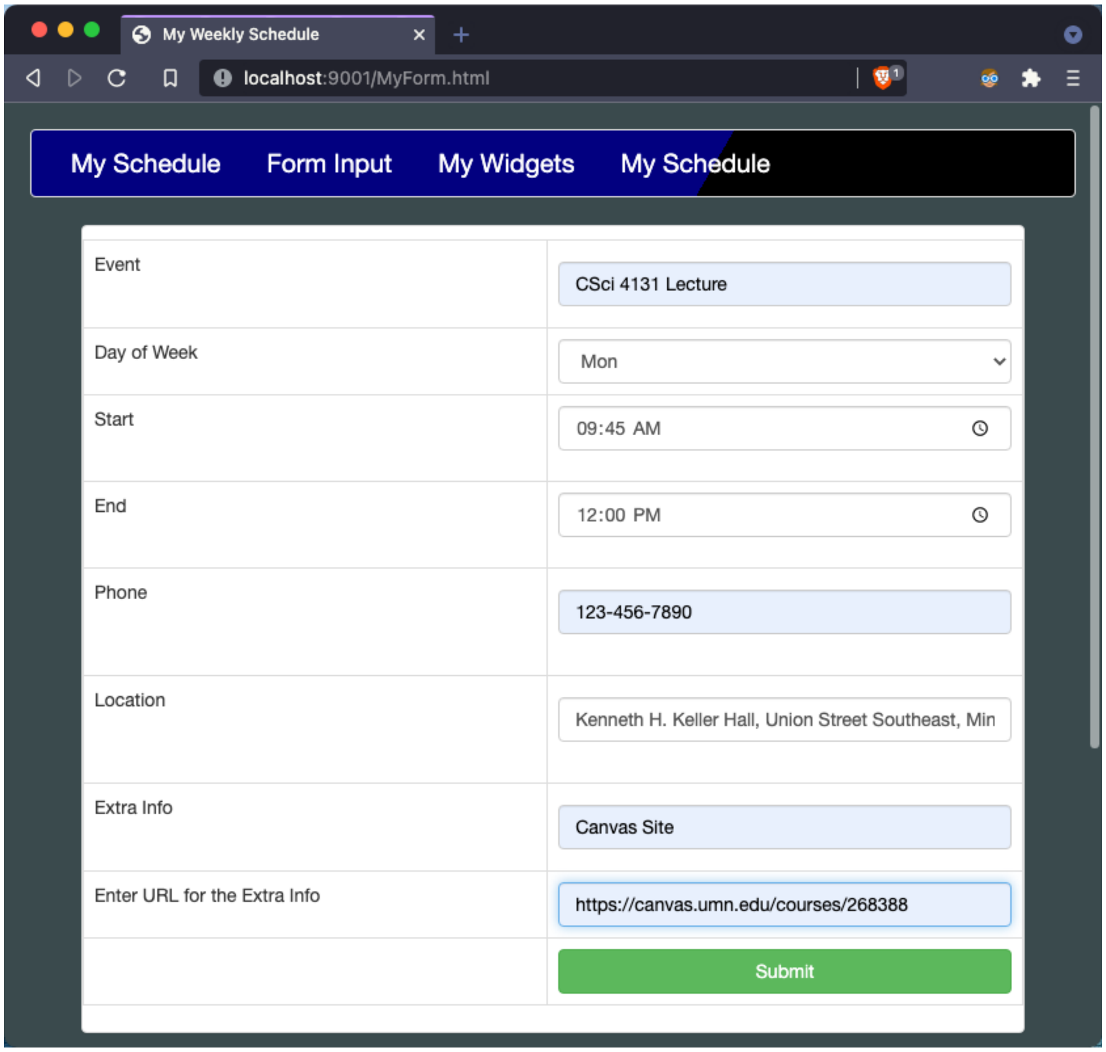
4.2 POST request

You will use the form that you developed for first homework. In the first three homework assignments, your form simply submitted the data to our server which returned an HTML for the browser to display. For this homework, you will instead submit your form to **your** python HTTP server. You achieve this by first changing the *method* of the form to **"post"** and *action* to **http://localhost:9001/MyForm.html**

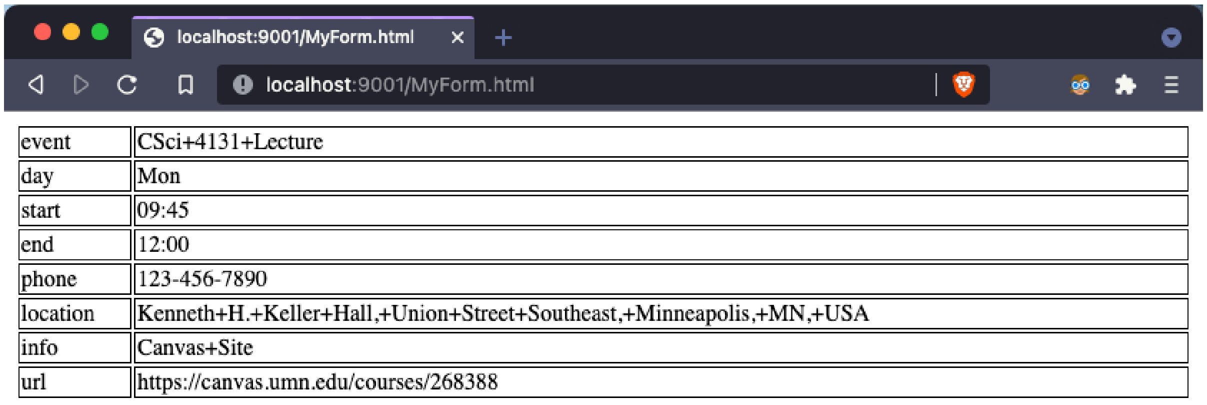
Upon successful submission of the form, your server should return an HTML page with all the information submitted on the form. So your server must construct and return an HTML document with the data values that was submitted to your server via the POST request message sent by the client. One way to do this is to use Python to build a string with HTML tags and the data embedded in it (concatenate HTML strings & the data values to form an HTML document).

Below are sample screen-shots of the expected results displayed in your browser when your form is submitted with the *method* set to **"post"**.

Pictured Below- Form submitted using a post request



Event	CSci 4131 Lecture
Day of Week	Mon
Start	09:45 AM
End	12:00 PM
Phone	123-456-7890
Location	Kenneth H. Keller Hall, Union Street Southeast, Min
Extra Info	Canvas Site
Enter URL for the Extra Info	https://canvas.umn.edu/courses/268388



event	CSci+4131+Lecture
day	Mon
start	09:45
end	12:00
phone	123-456-7890
location	Kenneth+H.+Keller+Hall,+Union+Street+Southeast,+Minneapolis,+MN,+USA
info	Canvas+Site
url	https://canvas.umn.edu/courses/268388

Pictured Above - Response rendered in browser to Form Submitted with a Post request (note the Address/URL bar)

4.3 Server response to Error conditions

Your http server will also need to handle error conditions, as specified below.
Your server should include an appropriate error message in the response message body, specific to the error condition.
NOTE, if we do not provide an html file specifying the error code to include in the error response message your server composes to send to the client, your server should insert an appropriate plain text error message in the response message it composes to send the client (*e.g., the client can be your Browser, Curl, Telnet, POSTMAN, etc.*).

Your HTTP server should handle the following error conditions: 403, 404, and 405. It should send appropriate error responses as specified on the next page.

1. If the web server does not have the permission to access requested resources (e.g., private.html), your server should create a response message with a 403 error response code and 403.html which is sent to the requesting client.
2. If the requested resource is not found, your server should create a response message with a 404 error response code and 404.html which is sent to the requesting client.
3. If the request from the client is anything other than GET or POST, the server should compose a response message containing a 405 error – method not allowed. This requirement is provided to you.

5. Testing Utilities and Guidelines

You can test the HTTP server using a HTTP client such as a browser, curl, telnet, or Postman as follows:

a) *Testing with a browser or the Linux Curl command(refer to the document CurlTesting.docx)*
i) To test your server by sending a GET request message using your browser, enter the following in your browser's address bar after starting your server so it is listening on port 9001:

http://localhost:9001/MySchedule.html

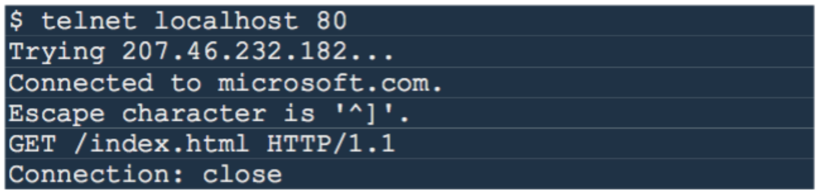
If your server composes and sends a response message correctly, your browser should display The MySchedule.html file you created for homework 3.

ii) To test your server by sending a GET request message using the Linux **Curl** command, type the following at your Linux command line prompt after starting your server so it is listening on port 9001:

```
curl -i -H "Accept: text/html" http://localhost:9001/MySchedule.html
```

The server will send its response back in a string that is displayed in your Linux terminal window.

b) *Using telnet in the Linux terminal to GET a file named index.html from the directory in which your server is running (in the example shown below replace 80 with the port number your server is listening on – 9001, for example). e.g.,:*



```
$ telnet localhost 80
Trying 207.46.232.182...
Connected to microsoft.com.
Escape character is '^]'.
GET /index.html HTTP/1.1
Connection: close
```

c) Test via Postman (this is our preferred method of testing). You can download the app here: https://www.postman.com/downloads (or use the web version).

   Instructions for issuing and capturing responses to HTTP requests, see the following link: https://learning.postman.com/docs/sending-requests/capturing-request-data/capturing-http-requests/

d) Python also has a http requests module, so you can create Python3 programs to formulate and send http requests to your http server and print out the responses returned by your server.
   See: https://www.w3schools.com/python/module_requests.asp for details.

6. Submission Instructions

Submit your updated Form page with any JavaScript and CSS files it requires and your server program (named: **myServer.py**) in a compressed zip with the name:

`<YourUMNx500id>.zip`

For example, user *`john1234`* should submit a file named: `john1234.zip,` with a server file named: **myServer.py**

• In addition to the files specified above, please include all the files, e.g., 403.html, 404.html and private.html, MySchedule.html, MyForm.html, MyServer.html etc. that you used for testing your server.

7. Evaluation Criteria

The HTTP server you submit will be graded out of 50 points on the following items:

- Server successfully establishes a socket and binds to 9001 by default. **5 points**
- Server correctly responds to GET requests for HTML file. **5 points**
- Server correctly responds to GET requests for a file with an image or audio. **5 points**
- Server correctly responds to GET requests for a file with JavaScript or CSS. **5 points**
- MyServer and the navigation links on MyServer page (MyServer.html) works correctly. **5 points**
- Server correctly processes POST requests for your updated form. **10 points**
- Server correctly responds with 200 and the html file (page) requested. **5 points**
- Server correctly responds with 403 and the html file included with this assignment. **5 points**
- Server correctly responds with 404 and the html file included with this assignment. **5 points**

- The ResponseBuilder is completed and used to build responses in the Server. **5-point bonus**
- Source code is documented and readable. **(5-point penalty – code that does not have helpful documentation and/or is unreadable will be penalized 5 points)**
- Submission instructions are not followed correctly. (**up to 15-point penalty – you can lose up to 15 points, the specific penalty will be determined at our discretion.)**

Reminder: All assignments will be graded on the machines running a Unix operating system or Linux variant - so make sure to test your server on a machine running a Unix operating system or Linux variant to ensure it functions correctly as specified in the evaluation criteria above. Note, the CSELABs computers run Ubuntu, a variant of Linux.