CSCI 4131 – Internet Programming Assignment 4

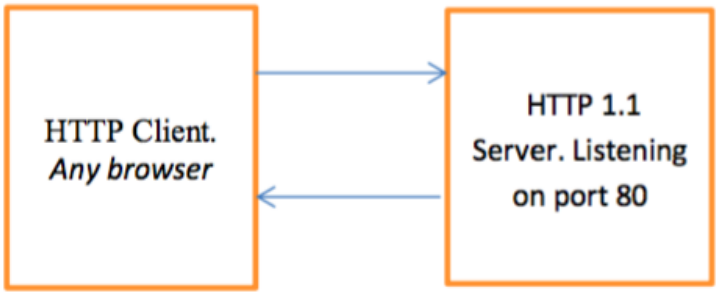
DUE DATE: Friday March 12th at 11:59pm

Late Submissions accepted until Sunday March 14th 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 12 pages long.

When a web client (such as Google Chrome) connects to a web server (such as ***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 GET 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 HTTP Sever 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 meta data 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 sent 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 4.4 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.
- ***MyContacts.html*** – use your own MyContacts.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. It also includes a search form which can be used to test your server's capability to respond with a redirect code. (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 and 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, please 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 MyContacts.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
```

Finally, 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.

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 Contacts from your web browser by typing:

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

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`.

NOTE: if you are developing your solution on a computer running the Windows Operating System, setting your port and/or default port to 9001 may result in an error – some Intel devices use that port. Try using port 9003 instead.

Your code should not use Python's `httplib` module, or http.server or http.client in Python's http module/library . You should do your own socket programming on this assignment.

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.

When you run your server with no arguments, 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). Your server should also accept one optional command line argument that specifies a port to which it should bind. Two example calls to start your server on the CSELabs computers are as follows:

1. *python3 myServer.py*
2. *python3 myServer.py 9001* (server then binds to and listens on port 9001)

On Windows 10, if you are using the EchoServer.py code we provided as the basis to build your server, you can run the executable by double-clicking the file.
Finally, your server should log any incoming requests to STDIN.

4. The Hyper Text Transfer Protocol (HTTP)

The HTTP is a protocol of non-trivial size. You will only be implementing a small, functional subset of HTTP GET, HEAD, and POST requests. You code will also recognize and respond to a limited subset of errors discussed in section 4.4 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 URL address bar:

http://localhost:9001/MyContacts.html

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

GET /MyContacts.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) or an audio file (.mp3). 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 MyContacts.html is html and the resource type of OuttaSpace.mp3 is mp3). Note that your html file (e.g., MyContacts.html page) may contain links to external .css and .js files. You are not required to handle these two resource types, so if you desire, you can embed your JavaScript and CSS files styling in your MyContacts.html file.

You will receive bonus points if your server can successfully return externally linked .css and .js files (successful means they are sent by your server and successfully obtained and applied to your MyContacts.html by the browser you are using).

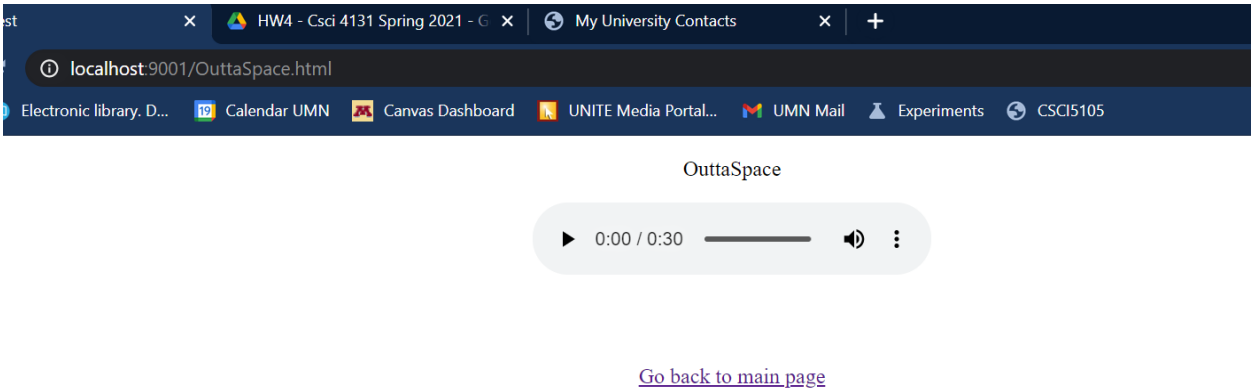
As specified above, your server should be able to access the image and audio resources and return them to the browser as part of a GET request.

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):

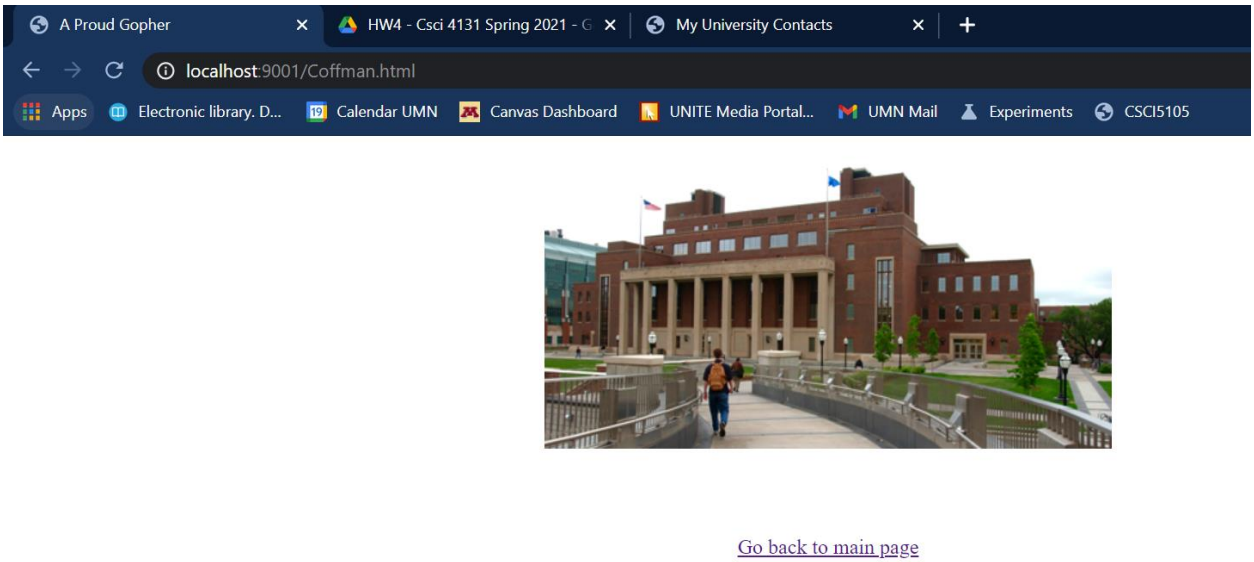


After the page is displayed, you should be able to play the Audio.

Next, when you type:

http://localhost:9001/Coffman.html

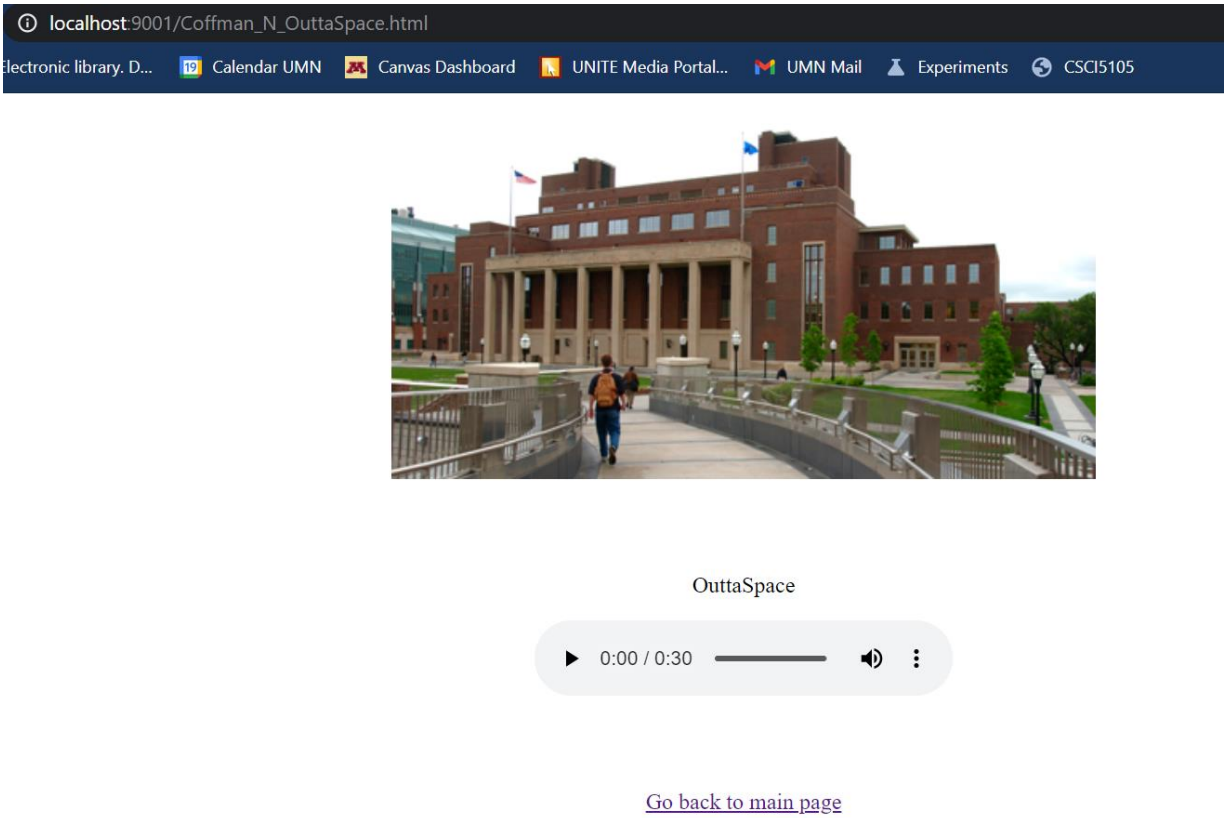
in your browser's address bar, it will request the file *Coffman.html* from your server. Your server should return the contents of the file *Coffman.html*, and your browser window should display the following web page (shown on the next page):



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:



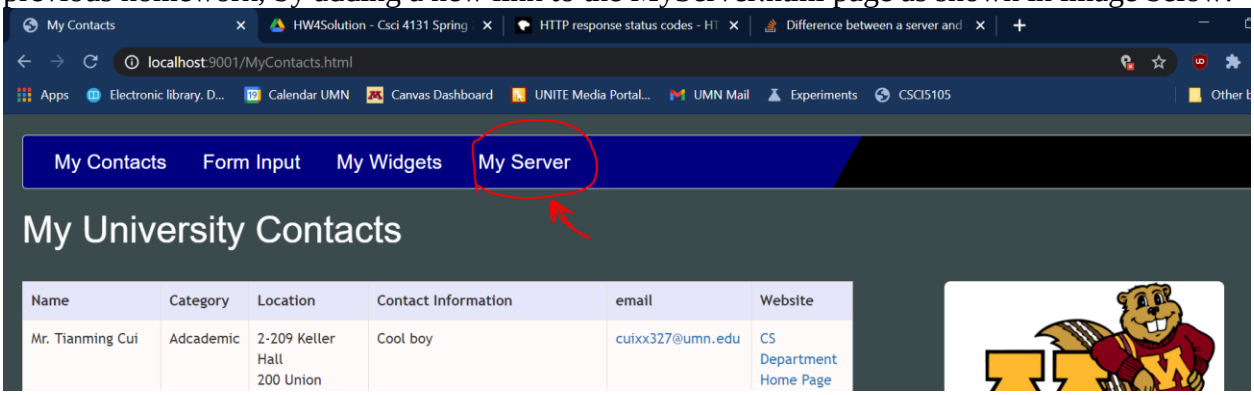
You should be able to play the Audio by selecting the Play button.

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 sever 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 webpages above i.e., *Coffman.html*, *OuttaSpace.html* and *Coffman_N_OuttaSpace.html*.

The requirements for the search field on *MyServer.html* page shown on the previous page are specified in section 4.4 of this document.

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 HEAD Requests

HEAD requests are almost identical to GET requests. However, instead of returning a status code followed by the requested URL, your server should only send the status response line (for example HTTP/1.1 200 OK) and an empty message body.

The HEAD request is the easiest request to implement, but you cannot test it with your browser. You must test it with the CURL command, Telnet, or a utility to send messages such as POSTMAN (POSTMAN is our preferred utility)

Your server should respond to HEAD request by checking to make sure the requested resource is present and has the proper permissions, and then compose a response message – which, HEAD request will consist of the status response line and an empty message body. We will review a sample server implementation of the HEAD request during lecture.

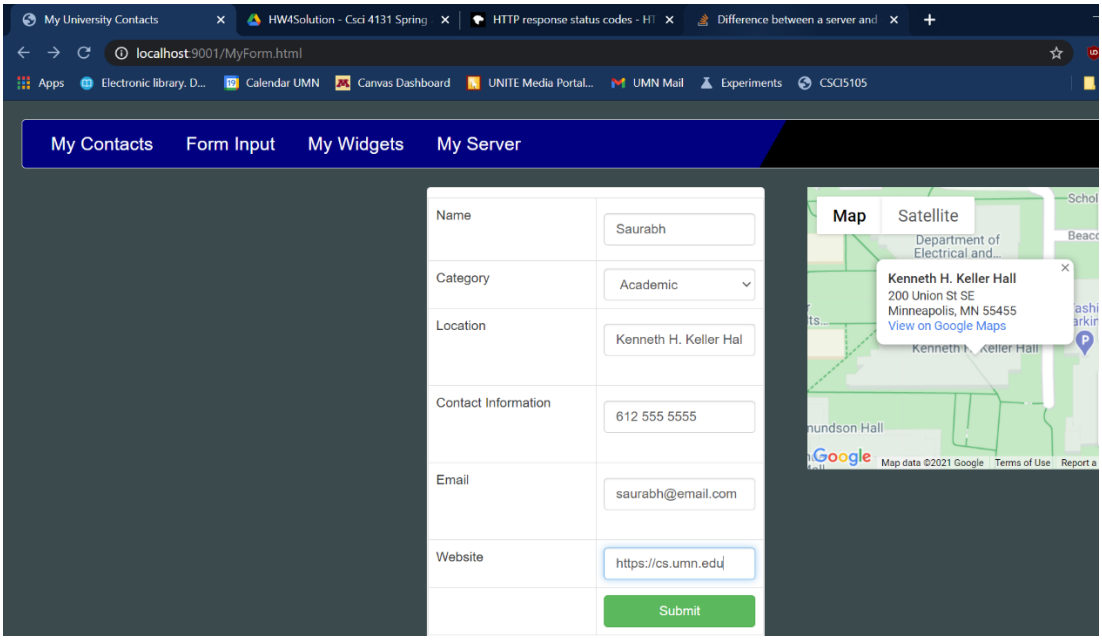
4.3 POST request

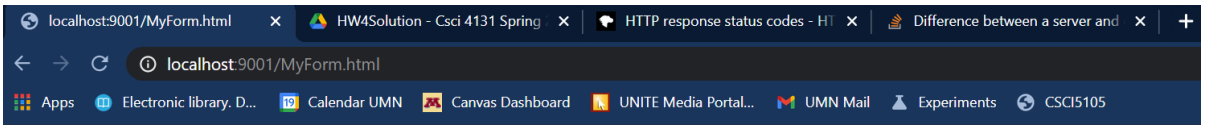
You will use the form that you developed for first homework. In the first three homework assignments, your form simply submitted the data to our server which returned an HTML for the browser to display. For this homework, you will instead submit your form to **your** python HTTP server. You achieve this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

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 and 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





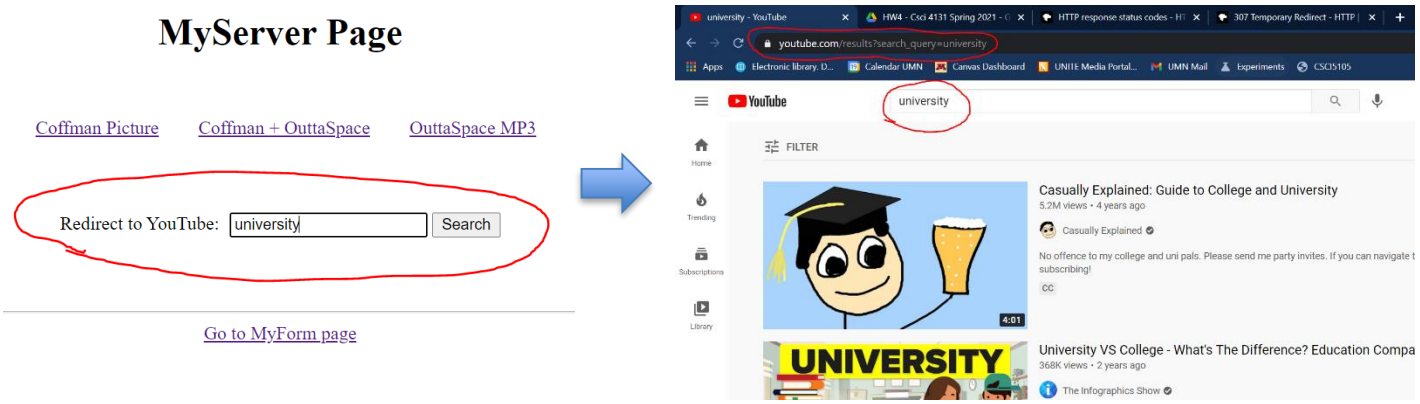
Following Form Data Submitted Successfully:

name	**Saurabh**
category	**Academic**
location	**Kenneth+H.+Keller+Hall,+200+Union+St+SE,+Minneapolis,+MN+55455,+USA**
contact	**612+555+5555**
email	**saurabh@email.com**
website	**https://cs.umn.edu**

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

4.4 Server response to Redirect conditions

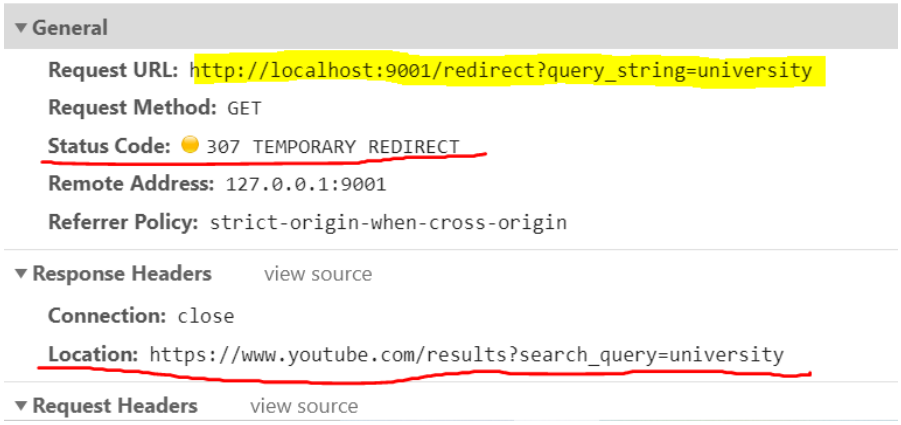
There is a search field on the page *MyServer.html* shown in the first (leftmost) image:



When you enter some text in the input box and click submit, the page should redirect to the YouTube search results for the text you entered. This is shown in second (rightmost) image above which displays where YouTube search results for 'university' are displayed.

On submitting this search form, the MyServer.html page is designed to make a GET request to your local server at the URL: http://localhost:9001/redirect. It will include the inputs from the text box as request arguments.

For example, when you type 'university' in search box and click submit button, the page makes a request with the full URL as: http://localhost:9001/redirect?query_string=university

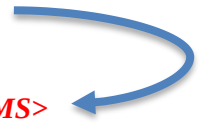


Your server should handle these requests and return a response with HTTP 307 (Temporary Redirect) status code. You can read more about this status code in section 10.3.8 of RFC 2616.

This response will have an empty body. However, in the response headers you should specify a 'Location' header with the value equal to a YouTube URL. On receiving the response, the browser reads this Location header and navigates to the given YouTube URL.

To prepare the YouTube URL you will need to parse the request sent by the browser, extract the search terms and use them in the YouTube search URL as shown at the top of the next page.

Browser Request:
http://localhost:9001/redirect?query_string=<***SEARCH_TERMS***>



YouTube URL for the Location header:
https://www.youtube.com/results?search_query=<***SEARCH_TERMS***>

4.5 Server response to Error conditions

You will 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 serve does not have the permission to access requested resource (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, HEAD, or POST, the server should compose a response message containing a 405 error – method not allowed.

5. Testing Guidelines

To run your HTTP Server, you should pick a port number above 5000. 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/MyContacts.html

If your server composes and sends a response message correctly, your browser should display The MyContacts.html file you created for the 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/ MyContacts.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/

6. Submission Instructions

You should submit your updated Form page with any JavaScript and CSS files it requires and your server program (named: **<YourUMNx500id>.py**) in a compressed file (tar, gz, etc.) with the name:

`<YourUMNx500id>HW4.tar (or gz, or zip, etc.)`

For example, user *john1234* should submit a file named: `john1234HW4.tar,` with a server file named: **john1234.py**

• In addition to the files specified above, please include all the files, e.g., 403.html, 404.html and private.html, MyContacts.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 105 possible points on the following items:

- Server successfully establishes a socket and binds to 9001 by default. **5 points**
- Server successfully accepts (receives and uses) a parameter to assign the server to a non-default port number. **5 points**
- Server successfully accepts connections from clients and reads incoming messages. **5 points**
- Server correctly identifies GET requests, HEAD requests. **10 points**
- Server correctly responds to GET request for HTML file. **5 points**
- Server correctly responds to get request for a file with an image. **5 points**
- Server correctly responds to get request for a file with audio. **5 points**
- The MyServer page (MyServer.html) is included in the navigation bar of your site. **5 points**
- Server correctly sends the redirect response**. 1 points**
- The MyServer page (MyServer.html). YouTube search redirect works. **5 points**
- The navigation links on MyServer page (MyServer.html) work. **5 points**

- Server correctly responds to html file requiring external JavaScript and CSS files. ***(5 point bonus)***
- 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 405 and plain text message. **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**
- Server logs requests to STDOUT. 5 **points**
- 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 25 point penalty – you can lose up to 25 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.