# CSCI 4131 – Internet Programming Assignment 4

***VERSION 2 – 2/25/2020***

**DUE DATE: Friday March 6th at 3pm**

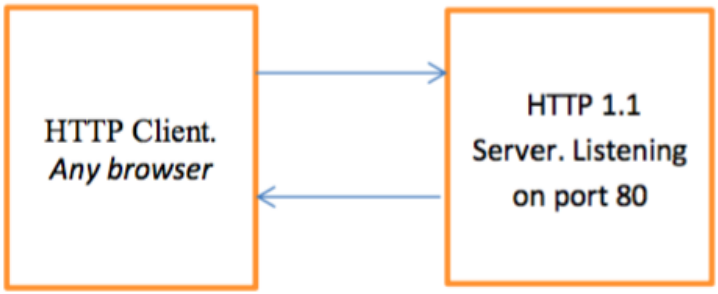
**Late Submissions accepted until Saturday March 7th at 6pm (evening)with 15% penalty**

***Submissions after the Late Submission Date and Time will not be accepted.***

## 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 server. You will need to go through RFC 2616 for HTTP 1.1 protocol details. This assignment description is 10 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.




The 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 which sent the GET request.
6. Otherwise, if the resource requested by the HTML client is not found, then the HTTP Server 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.
- ***MyForm.html*** – use your own MyForm.html file from Assignment 3, updated as described below.
- ***displayPix.html*** – html file containing an image to use for testing your server's capability to respond to a request for an image (Provided).
- ***bellToll.html*** – html file containing an audio controls element to use for testing your servers capability to respond to a request for an audio file (Provided).
- ***Goldy_N_Bell.html*** – contains both and image and audio control element to use for testing your servers capability to respond to a request for an image and an audio file (Provided).
- ***gophers-mascot.png*** – the image file (a .png file) used by displayPix.html and Goldy_N_Bell.html (Provided).
- ***tolling-bell_daniel-simion.mp3*** – the audio file (a .mp3 file) used by bellToll.html and Goldy_N_Bell.html (Provided).

We have provided these files in the file named: Hw4Resources.zip

Additionally, this assignment requires that you update your Form page (MyForm.html) from homework assignment 3:

1. Update the input validation so that your form only accepts Contact Names and Favorite Places that include alpha-numeric characters, spaces, or commas ***without leading or trailing whitespace***. If a user enters an event name or location with characters other than those specified in the previous sentence, or with leading or trailing blanks, your form should report error (before or on submission). You will use your updated form page to test your server's ability to successfully handle a form that uses a post method.
2. Change the method of the form in your MyForm.html to "post".

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 displayPix.html
chmod 644 bellToll.html
chmod 644 Goldy_N_Bell.html
chmod 644 gophers-mascot.png
chmod 644 tolling-bell_daniel-simion.mp3
chmod 644 MyForm.html (Your form from HW 3, updated)
```

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)***

## 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`.

**Your code should not use the `httplib` module of the Python standard 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.**

When you run your server with no arguments, your server should bind to port 9001 and serve requests. 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 are as follows:

1. *python myServer.py*
2. *python myServer.py 9002* (server then binds to and listens on port 9002)

***Additionally, 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. When your web browser requests a webpage from a server, it is issuing a GET request. Below and at the top of the next page is an example GET request that will be 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*

For example, if you enter the following address in your browser's URL address bar:

http://localhost:9001/MyContacts.html

the browser should display a version of the MyContacts.html file.

The GET request from the client should be constructed so it includes accept headers which accept at least three resource types: *.html*, *.png*, and *.mp3*

- For an *.html* resource, the accept header in the request message from the client should contain **text/html** or ***/***.
- For a *.png* resource, the accept header in the request message from the client should contain **image/*** or ***/***.
- For *.audio* resource, the accept header in the request message from the client should contain **audio/*** or ***/***.

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.

Thus, the GET requests you compose should be constructed so they include accept headers which accept at least three resource types: *.html*, *.png*, and *.mp3*

- For an *.html* resource, the accept header in the request message from the client should contain **text/html** or ***/***.
- For a *.png* resource, the accept header in the request message from the client should contain **image/*** or ***/***.
- For *.audio* resource, the accept header in the request message from the client should contain **audio/*** or ***/***.

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 three files:

i) `displayPix.html`

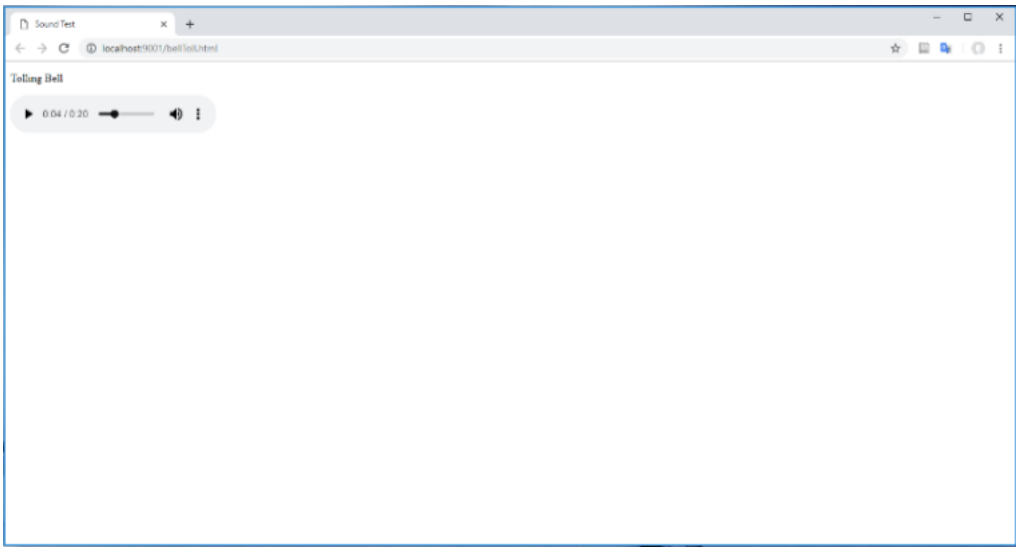
ii) `bellToll.html`
iii) `Goldy_N_Bell.html`

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/bellToll.html

in your browser's address bar, it will request the file *bellToll.html* from your server. Your server should return the file *bellToll.html*, and your browser window should display the Web page displayed below:




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

Next, when you type:

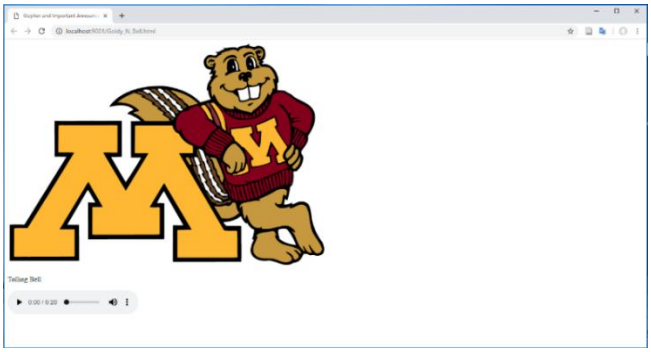
http://localhost:9001/displayPix.html

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

Finally, when you type:

http://localhost:9001/Goldy_N_Bell.html

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



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

## 4.2 HEAD Requests

HEAD requests are almost identical to GET. Instead of returning a status code followed by the requested URI, 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.

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.

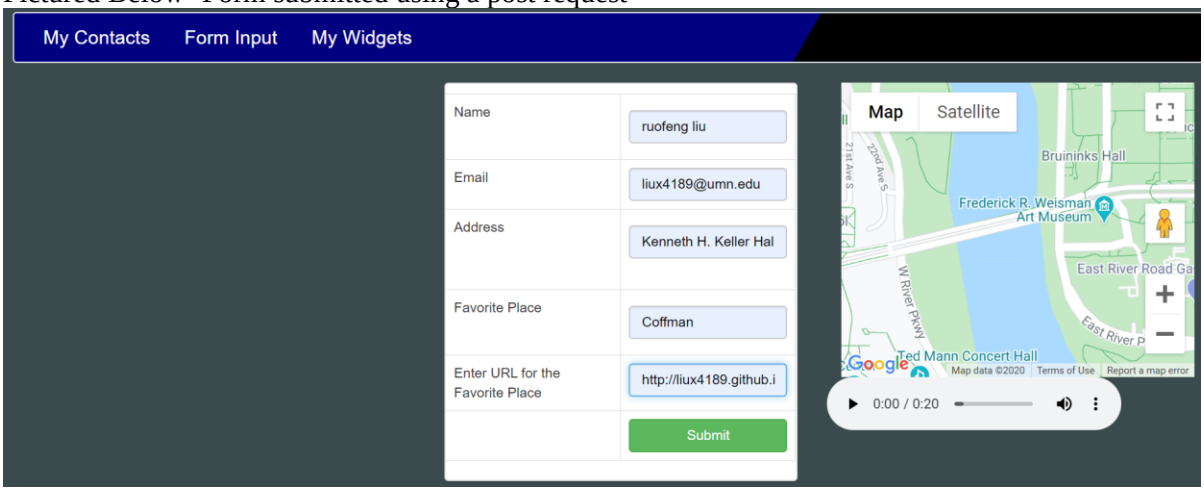
## 4.3 POST request

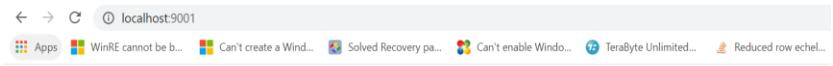
You will use the form that you developed for first homework. In the first three homework assignments, your form simply submitted the data to our server which returned and HTML for the browser to display. For this homework, you will instead submit your form to **your** python HTTP server. You achieve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

Then, upon successful submission of the form, your server should return an HTML page with all the information that was submitted on the form. So your server must return an HTML construct to display the data that was submitted to your server via the 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.

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 | ruofeng+liu |
| email | liux4189@umn.edu |
| address | Kenneth+H.+Keller+Hall+200+Union+St+SE |
| place | Coffman |
| url | http://liux4189.github.io |

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

## 4.4 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, 405, and 406. It should send appropriate error responses as specified on the next page.

1. 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.
2. 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.
3. If the request from the client contains accept headers (e.g., *Accept: text/html*), and the content characteristics of the requested resource are not acceptable according to the accept headers, then the server should compose a response message with a 406 error code and send it. For example, if the accept header(s) in the request message specify only html files (*Accept: text/html* in the http request message), and the requested entity is an image file (.png), your server should compose and send an HTTP response message indicating a 406 error to the requesting client.

## 4.5 Redirection Response

Finally, server should also compose and send redirection responses for certain URLs. Specifically, for this assignment, if the http GET request is for a resource named "**mytube**", then the server should redirect the client to the following location: https://www.youtube.com

For example if a browser sends an http GET request composed from the URL: http://localhost:9001/mytube, then the browser should be redirected to the URL https://www.youtube.com . To enable your client (e.g., your browser) to redirect correctly to https://www.youtube.com, your server should compose and send the appropriate http redirect response message with required headers and send it to the client (e.g., your browser). Your server should include the appropriate location headers in the response. The redirection response should include the "`Permanently moved`" status code. Please refer to section 10.3 of RFC 2616 for details.

# 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
```

## 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, 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, 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. **10 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**
- Server correctly responds to html file requiring external JavaScript and CSS files. ***(5 point bonus)***
- Form page from assignment 3 updated to admit event names and locations with numbers, letters, spaces, and commas (but not leading or trailing with space). **5 points**
- Server correctly processes POST requests for your updated form. **15 points**
- Server correctly responds with 200 and the html file (page) requested. **5 points**
- Server correctly responds with 406 and plain text message. **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** ***(Evaluation Criteria continued on next page)***
- Server correctly responds with a redirection request response message (301). **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 . (**5 point penalty – you lose 5 points)**

**Reminder: All assignments will be graded on the cselabs machines - so make sure to test your server on a cselabs machine to ensure it functions correctly as specified in the evaluation criterial above.**