# Homework 4:
# HTTP Requests & Responses

CSCI 4131: Internet Programming (Fall 2022)

**Deadline**

The deadline for this assignment is November 18 at 11:55pm. It can be turned up to 24 hours late for a 10% deduction.

## 1 Change Log

- None yet.

## 2 FAQS

- None yet.

# 3 Introduction

## 3.1 Learning Goals

The objective of this assignment is for you to learn about the Hyper-Text Transfer Protocol (HTTP) and build a small subset of an HTTP server in Python. In this assignment, using Python 3 and TCP sockets, you will program some of the basic functionality of an HTTP Web Server. A lot of the protocol information you need for this assignment we have talked about in-class, however some we have not. You may find that you need to read online documentation about the HTTP protocol, or even the formal protocol (RFC 2616) itself.

By doing this assignment you will:

- Learn about HTTP request / response structure
- Get hands-on experiance with the HTTP protocol
- Work in partially-built server code. This will practice your software engineering and python skills in general.
- Experiment with several HTTP "mini protocol" (I.E. redirection, HTTP basic auth, Form-posts etc.) and learn how to quickly absorb and understand the details of these interactions.

# 4 Included Files

- `myServerStudent.py` contains an initial starting point of the web server.
- `MyServer.html` serves as an entry point to resources for testing.
- `Coffman.png`, `Coffman.jpg`, and `Coffman.html`
- `OuttaSpace.mp3` and `OuttaSpace.html`
- `401.html`
- `404.html`
- `private.html`

# 5 Recommended references

- `https://docs.python.org/3/library/socket.html` – Python socket code.
- `https://docs.python.org/3/library/base64.html` – Python's base64 decoding utility. This is a way of encoding binary values into text.
- `https://docs.python.org/3/library/urllib.parse.html` – Pythons URL processing library – the unquote_plus method will be particularly useful
- `https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html` – A "Cheat sheet" on python's optional "type hinting" syntax – used in much of our provided code.
- `https://developer.mozilla.org/en-US/` – MDN has MANY pages that would be relevant.
- `https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html` – an HTTP tutorial.

# 6 Requirements

The requirements for this homework are slightly different than in the past. In the past requirements could be done separately, and earlier requirements were not needed to do the later ones. In this assignment, however, you will frequently need earlier parts of the assignment done before you can attempt the later ones.

A few further notes:

- Use of any libraries or packages not already included in the handout file is not permitted without explicit approval. Slack is the best way to request permission to use python libraries we didn't explicitly import.
  - We want to be reasonable here, and will probably accept most reasonable requests, we just want to make sure you're not importing code from python's built-in web-server or other libraries that trivialize this problem.
- While the provided myServer.py code is ultimately not required it is **strongly recommended** Do not let it's size dissuade you – you're better off with this starting point than without it.
- The provided myServer.py code uses "TODO:" in either a string or comment before any point in the code you should need to modify.

## 6.1 Basic GET server

Your first requirement is the simplest to describe, but represents the majority of initial development for this assignment. Starting with the provided myServer.py you must build a working web-server that responds to GET requests for local files. The provided myServer.py file is, ultimately, optional – you can change it as you see fit – however, it represents the class staff's personal opinions about the best way to go about structuring and organizing this code. It also contains pre-built python functions and objects for managing the socket-level network interactions.

To get points for this requirement, your code must

- Be able to be run with the command `python3 mySever.py`.
  - Doing so will cause it to bind to port 9001.
  - If your program cannot bind to port 9001, it is ok to temporarily rewrite your code to use another port. HOWEVER, make sure your code uses port 9001 in it' final submission.
  - Your program should them listen to, and handle, incoming TCP connections under the HTTP protocol
- Be able to respond to GET requests for all of the files involved in your HW3 code (javascript, HTML, CSS, images, font, etc.)
  - The files should be accessible in urls matching the required folder structure from HW3. For example, a request to `http://localhost:9001/myContacts.html` would get the file myContacts.html, and a request to `http://localhost:9001/resources/js/passwordWidget.js` should get the passwordWidget javascript code, etc.
  - The header of the HTTP response should contain an accurate status code along with headers: `Connection`, `Content-Type`, and `Content-Length` if they are applicable.
  - The body of the HTTP response should contain the related file.

- Accomplishing this should be enough for you us interact with all HW3 features using your python server as a web-server. **NOTE** – there are many headers (such as content-type) that browsers will infer if not sent. Therefore do not assume your code works solely because the browser can "make sense" of your server's responses. We will be checking for a correct HTTP response – and will be more demanding than the average browser.
- (We understand you may have lingering HW2/HW3 bugs in your code – to be clear, we should be able to load files from past assignments at this point – but it's ok if their features *Continue not to work*. If these features worked before, they should also work through your new python server.

To keep submissions, and URL structures consistent throughout assignments, we will again require a consistent folder structure for all submissions.

```
hw4
├── *.html
└── resources
    ├── audio
    │   └── *.mp3
    ├── css
    │   └── *.css
    ├── fonts
    │   └── *.{woff|ttf|etc}
    ├── images
    │   └── *.{jpg|png|etc}
    └── js
        └── *.js
```

This is a natural extension of the HW3 requirements for file structure.

#### 6.1.1 Hints and Notes

- This part of the assignment may seem very overwhelming initially. There is a lot of code that needs to exist to make this happen. Fortunately for you, we've actually provided much of the basic structure for this code, as well as several key parts of the program that make this happen. The actual amount of new code needed to make this happen is relatively small – and focuses directly on your understanding of the HTTP protocol at a low level.
- This step of the assignment will likely involve a large amount of code-reading. This may even feel like an exercise in *software engineering* rather than *web development*. In our professional experience we've found that there is a surprising amount of overlap between software engineering and web-development. If you're *code reading* skills are a bit rusty, remember your goal isn't necessarily to understand every pre-built function, but instead to understand the provided *software structure*. We can try to support this in office hours.
- You will mostly need to make changes in the ResponseBuilder class, and the get_request function – although you should expect to need a few other changes throughout the code base to manage all of the required details..
- Your server will need to handle a large range of file types, setting appropriate HTTP headers (namely `Content-Type`) to indicate the various file types.

- Don't forget to make sure your various linked files work as well – CSS, images, fonts, etc. should all load correctly via a GET request.

### 6.2 Extra files

Uploaded with this assignment are a few extra files. We would like you to add these to your files in the appropriate locations (resources in the resources folder, html in the root directory). These play two roles:

- Ensure that you've designed your server to accommodate a range of files, both text- and binary-files, as well as files with different content types.
- Allow equitable grading on our end – By making sure everyone support a fixed set of resources grading can be done more objectively.

You do not need to make these extra files be linked from your main website files. To be clear, while we will be doing *primary testing* on these files, we will also be looking at your other homework files as part of testing.

### 6.3 Head Request type

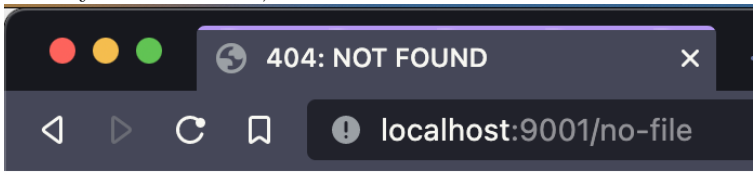
Your server should support head requests. Some code is already in place for this. The head request type is expected to return all the same headers as a GET request type for the same URL, without returning a body. You might wonder what goal this serves: there are a few. First-and-foremost this allows pro-active cache handling. You can also use this to get the size of a file (via the content-length header) before getting the file itself!

This can be done *very easily* if you use good software engineering principals. You could, for example, run the same internal code for a GET response, but modify the response so it doesn't have a file attacked.

### 6.4 404 errors

A 404 error should be returned if the path provided is not found. Initially you can simply check if path of a GET request matches any file on disk. However, as you develop the server more and we add special paths you may need to adjust this so special paths do not return a 404 error.

With your 404 error, the content of 404.html should be returned.




# 404: NOT FOUND

# 7 Advanced Features

Once you complete the previous set of core features, we have a few "advanced" features to add. These can be done in any relative order.

## 7.1 private.html

The private.html endpoint represents a private file. For this file we are requesting a primitive form of username/password authentication known as the HTTP basic Authentication control flow. A sequence diagram denoting this control flow is provided in Figure 1.

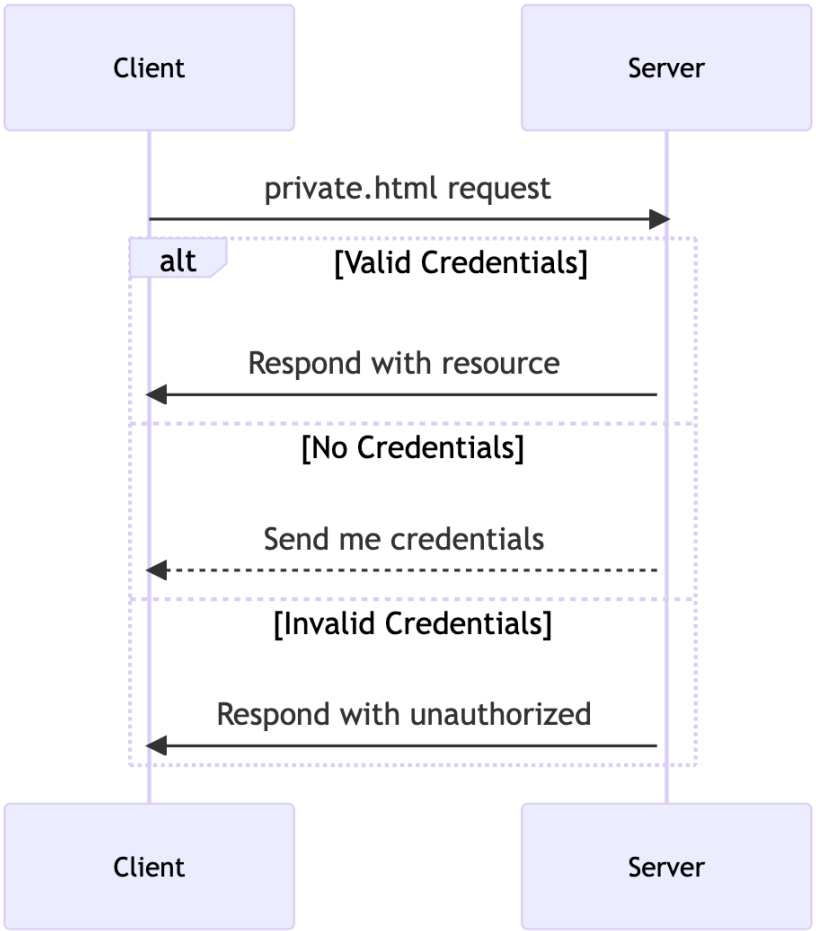



Figure 1: Sequence Diagram of requesting private resource

In brief, when a client requests `private.html`, your server should check if the client used the appropriate header to send along credentials (username and password) for this resource. If this is sent, the username and password should be validated in a manner appropriate to the protocol. If the username and password are correct (username: `admin` password: `password` ), then the file `private.html` should be returned. If the username and password are incorrect, sent incorrectly, or not sent at all, then a 401 response should be sent, along with the content of the file `401.html` When sending a 401 response, a specific header will be required to indicate the need for HTTP basic authentication.

Ultimately, there are many ways of implementing Authentication on the web. To be clear we are not asking you to reinvent the wheel, we very specifically want you to implement the basic access authentication protocol. We have left the PDF deliberately vague about the exact steps and headers involved in this process as we expect you to do some independent research on how this mini-protocol works in the context of the bigger HTTP protocol. The basic workflow is documented above, and as a hint – I will say that the wikipedia article on the subject is quite clearly written.

**NOTES**

- A key part of this protocol has the client sending the username and password to the server using base64 encoding. This is an encoding scheme for representing arbitrary binary sequences as ascii letters – it's commonly used in situations like headers where the need to be ascii compatible prevents sending arbitrary binary values. We've included an import to the python module for decoding base64 encoded strings into binary sequences: `b64decode`. You will almost certainly need to use this as part of your process.
- It should be noted that this form of authentication is commonly understood to be pretty insecure. While it makes for great practice in website design, it's not actually a good choice in securing a website that stores any private data.
- To be clear, the expected username is "admin" and the expected password is "password" – these are, of course, terrible choices if your goal is a secure website, but they will work well for testing the protocol.
- when testing for authentication, your browser will likely try to cache your credentials so you don't have to input them every time. This is great for everyday use cases, but not so much for testing. If you do your testing in a private window, the browser will forget the credentials when that window is closed. If not in an incognito window, closing out of the browser entirely usually forgets them.

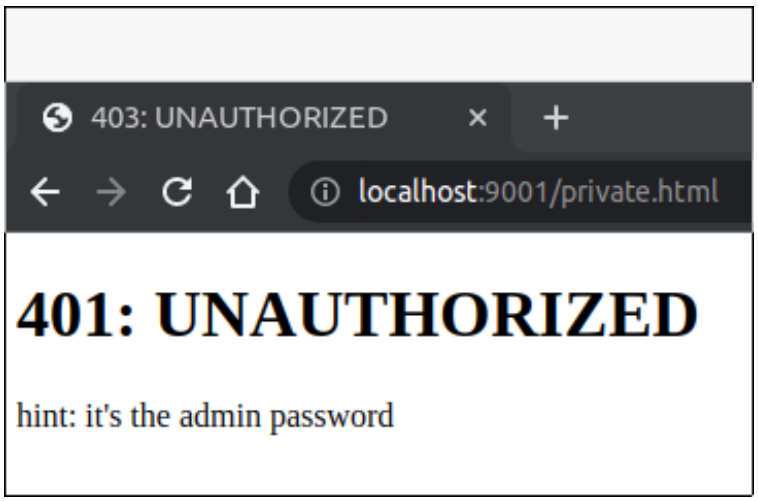
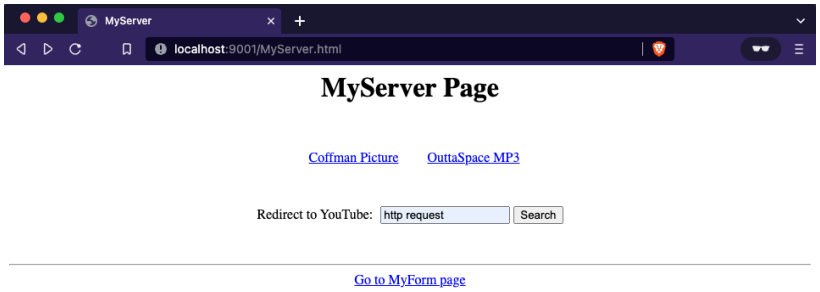



Figure 2: Example of 401.html being returned by the server

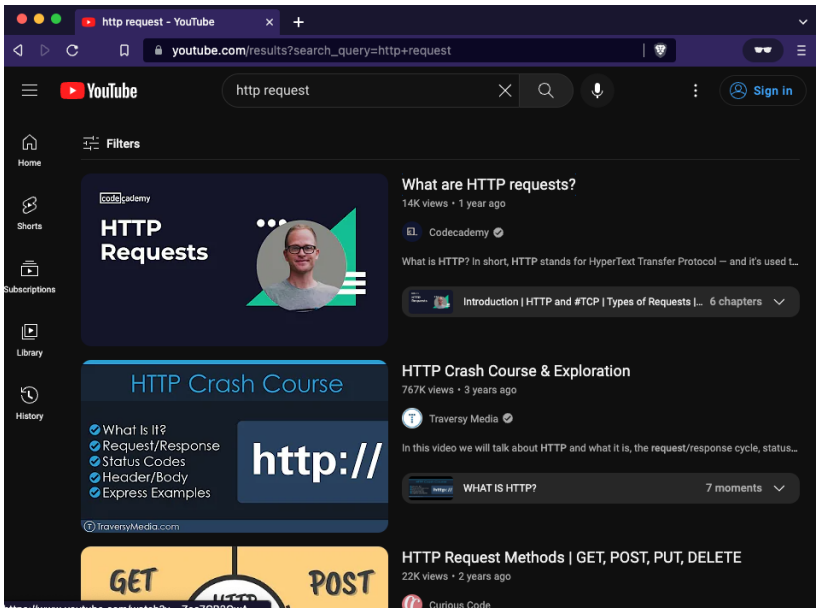
### 7.2 redirect

In addition to serving resources from your server, you will handle a redirection route. In `MyServer.html` is a `form` which when submitted sends a GET request to the `/redirect` path. Your server should be able to accept this GET request then redirect the client to the YouTube search results page where the user's input is already entered for them by using a temporary redirect.

An example of this interaction is shown in Figure 3.




(a) Client filling out form




(b) Client's window after submitting the form

Figure 3: Example of the server handling a redirect request

Note – doing this will involve parsing the query-string part of the URL to extract what the user searched for. You are expected to handle searches with spaces and other "special characters". The function (provided via import at the top of the file) `unquote_plus` can be used to parse the URL-encoding used by GET requests.

### 7.3 POST request

Your server should handle post requests. For simplicity, we're going to make all post requests behave the same way (I.E. your server will ignore the path when processing a POST request and behave the same regardless)

To generate the POST request, you will use the form that you developed for the previous homework. Your first step will be updating the `action` property on your form to send data to you server – again, the url doesn't matter as we will handle all post requests the same way. Therefore you might want to change the `action` of your form to `http://localhost:9001` or `/form` or some other value.

When your server receives a POST request, it will imitate an "echo" program and create a new HTML page with a table reflecting the content of the request. Specifically, each row of the table will be a key value pair of the request such that every key and value sent in the request is present within the table. An example of a resulting table is shown in Figure 4. Note – this should work with **any** data set, not just the fields from your contact-me form (although you will find your contact-me form to be a great testing tool).

Recall that your form's data will be sent form encoded. To decode this data, we've included the `unquote_plus` function in the provided starter code. Take care in decoding this data to extract key value pairs, as prematurely decoding may make extracting them impossible.

| | |
|---|---|
| postTitle | How do you set up an HTTP Server? |
| email | joh13266@umn.edu |
| username | joh13266 |
| link | http://www.example.com |
| category | concern |
| message | Hey, how's it going? |
| otherKey | With some other value |

Figure 4: An example resulting table from a POST request. Note the support for arbitrary keys by the presence of "otherKey"

### 7.4 Testing Guidelines

You can test the HTTP server using any client, such as a browser, cURL, telnet, or Postman. Note that many modern tools will put in their best effort to correct any mistakes in responses, meaning the response may be invalid in parts, yet the user would remain mostly unaware of this. Since the error correcting ability varies widely from client to client, and a goal of the course is one of understanding the HTTP protocol itself, we will be grading the response, and not the functionality in isolation.

### 7.4.1 Images

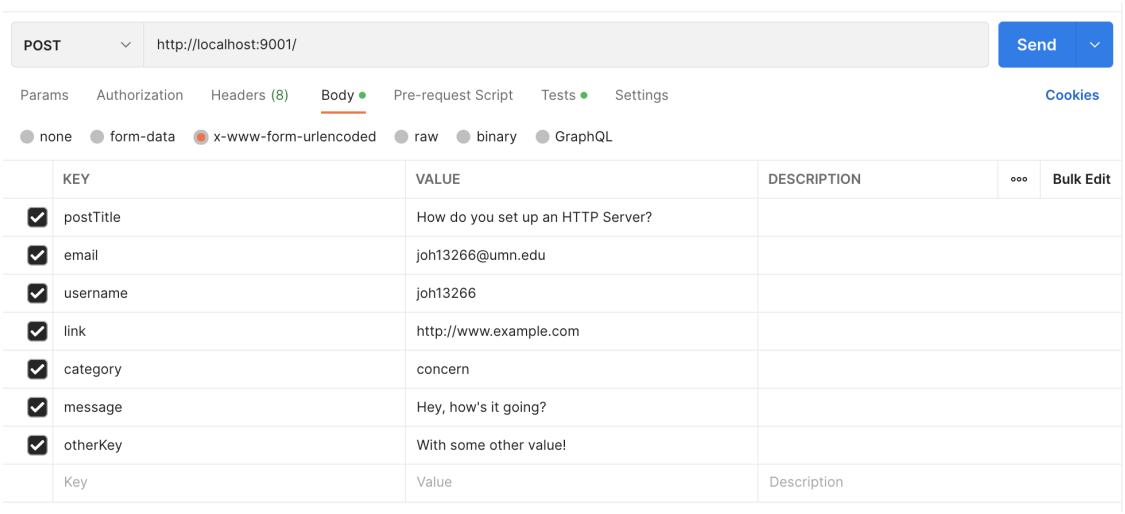



Figure 5: Example of POST request test with Postman

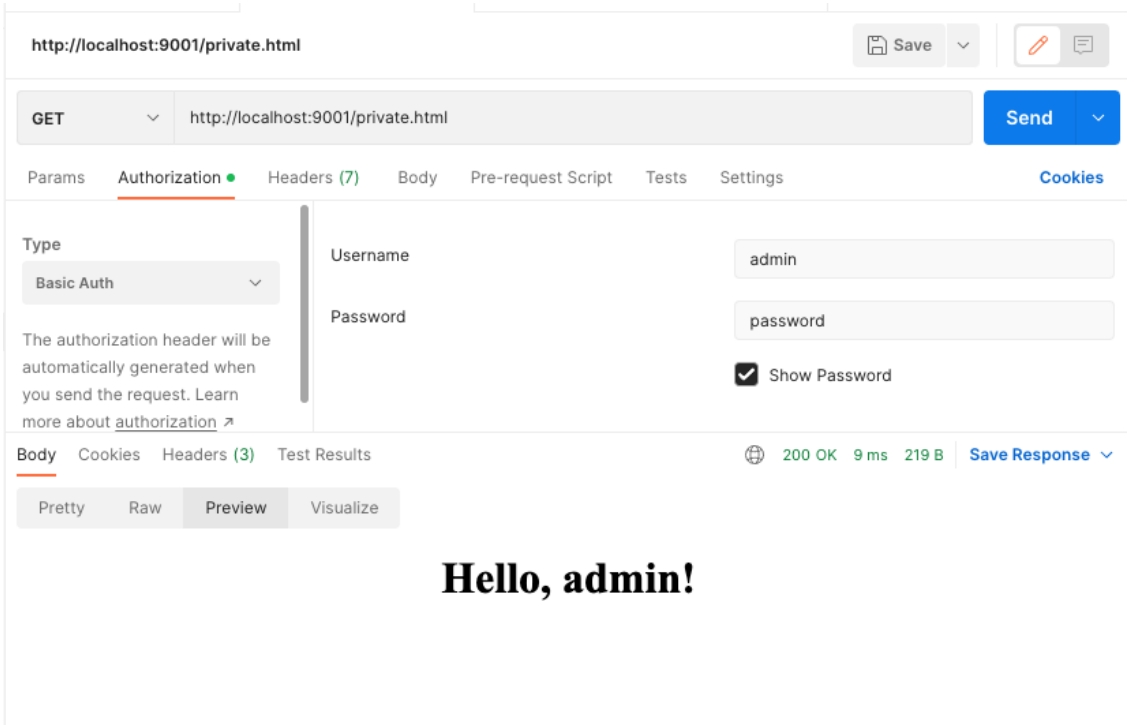



Figure 6: Example of an authentication test

# 8 Grading information and requirements recap

- 10 overall submission expectations

  - We may develop an autograder – if we do points will be taken from this to handle those poitns
  - File organization
  - Code style for your python code
  - (Note – while we expect your HW3 server files to be present here – we understand if they have lingering unresolved bugs or code style issues, we will not be grading those files themselves)
- 30 basic get requests
  - When looking at this we will be looking at a wide variety of URLs and content-types
  - Your server is expected to return content-type and content-length headers appropriately
- 15 head requests, 404, and 405
  - Each feature will be worth about 5 points
- 15 redirect url
- 15 POST responses
- 15 private.html authorization workflow