Warning: Parameter 1 to GadgetHooks::registerModules() expected to be a reference, value given in **/mnt/data/www/html/coepwiki/includes/Hooks.php** on line **195**

HTTP Server Project

From COEP Wiki
Jump to: navigation, search

Contents

How to have standard HTTP Server Running on Localhost to cross-check your project?

- Open Terminal

- Navigate to the directory you want to have the root directory.

- Run python -m http.server 8000

- Go to http://127.0.0.1:8000/ (localhost)

You can perform tests on HTTP server running on localhost from browser or Postman. (depending on you.)

```
@ Postman: https://www.postman.com/
@ http.server docs : https://docs.python.org/3/library/http.server.html
```

Common guidelines for reading and interpreting the RFC's

Backus Naur Normal Form will be followed to describe all headers.

Some examples of the following are:

1. HTTP-VERSION = "HTTP" "/" 1*DIGIT "." 1*DIGIT

2. http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]

3. Content-Encoding = gzip | deflate | compress | identity

The literature here is similar to the grammar that have been studied in Theory of Computation. Pay attention to the keywords that are applied with each description.

How to render images (or similar files) on browser ?

1. Divide response in two categories :

```
Response headers(must end with blank line)
Response body
```

2. Set content-type (response header) accordingly (e.g. image/jpeg for .jpeg image). You may want to check this out: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types

3. Set content-length (response header). You can use os.path.getsize() for this purpose.

4. Read file as 'rb' (response body)

5. Response header is a string so after you created all necessary response headers encode them.

6. Response body is nothing but file read in 'rb' which is 'bytes' object, so do not encode it.

7. Now add response headers and response body and send together conn.send(response_headers+response_body) OR you can send them one after another

Some Useful Locations of Apache

- **/var/www/html/ -** The Document Root of apache for adding HTML pages or files for making requests.
- **/etc/apache2 -** The Directory with all the configuration files of apache most important being apache2.conf
- **/var/log/apache2/ -** The Directory with log-files of apache

to look at requests sent by various HTTP Clients

- Web Browsers

1. Enter the developer console window of your browser (Mostly by pressing F12).

2. Switch to the Network tab of the console. It will start recording network activities.

3. Visit a webpage(or localhost, for that matter) to send a get request to the server.

4. A few logs will be displayed. Click on the very first log entry. You will say multiple tabs (response, headers, etc.)

5. You will get every single detail of your request. (Click on 'view source' to look at the raw text-based http request sent by browser)

- Postman

1. In the postman window, open the console. (View -> Postman console; Alt+Ctrl+C; small console icon at bottom left)

2. Send a request to your server. It will be logged on the console.

3. Expand the request to see all details. (Click on 'show raw log' to look at the raw text-based http request)

Working with Status Codes

- **403 Forbidden -**

This status code indicates that the server understood the request but refuses to authorize it. To implement it you will have to check file permissions on the requested resource(filename) using the **os** library. If it doesn't have the required permissions, we set the status code to 403. For example:

Request

```
GET /try.html HTTP/1.1
```

Response

```
HTTP/1.1 403 Forbidden
Date: Sun, 08 Nov 2020 13:31:51 GMT
Server: Apache/2.4.41 (Ubuntu)
Last_Modified: Wed,  Nov 18:47:18 4 GMT
Accept-Ranges: bytes
Content-Type: text/html
Content-Length: 257
```

- **403 Forbidden vs 401 Unauthorized**

401 Unauthorized is similar to 403 Forbidden, but it is used when authentication is required and it has failed.

References: [1]

Some Useful Libraries

- **datetime -** Used in inter-conversion of date and time in different formats for header fields and some status codes. Reference - https://www.dataquest.io/blog/python-datetime-tutorial/
- **pytz -** Used to get the timezone in GMT. Reference - https://www.geeksforgeeks.org/get-current-time-in-different-timezone-using-python/
- **logging -** Used for creating a log file with custom formats and levels of logging. Reference - https://docs.python.org/3/howto/logging.html
- **threading -** Used for Multi-threading. Reference - https://realpython.com/intro-to-python-threading/
- **requests -** Used for making HTTP requests through your server for testing your code. Reference - https://realpython.com/python-requests/
- **os -** Used for various file permissions and handling purposes. Reference - https://www.geeksforgeeks.org/os-module-python-examples/
- **base64 -** Used for decoding username and password available in the Authorization header. Reference - https://www.geeksforgeeks.org/encoding-and-decoding-base64-strings-in-python/
- **mimetypes -** Used to guess the mime type of a file in Python. References -

```
- https://docs.python.org/3/library/mimetypes.html
- https://www.tutorialspoint.com/How-to-find-the-mime-type-of-a-file-in-Python
```

- **uuid -** Helps in generating random objects of 128 bits as ids. Reference - https://www.geeksforgeeks.org/generating-random-ids-using-uuid-python/

Some Useful Resources to get started

- RFC-2616
- RFC-7231
- MDN HTTP Docs
- Basics of creating a HTTP Server
- Mulithreading References
- Server Testing Tools

Socket programming as a prerequisite

A basic explanation of socket and socket options:

- Socket module in python
- Port and address reusability

The TCP client-server assignment done as a part of the CN course provides a good base when it comes to socket programming in python. Make sure that the multithreaded TCP server implementation covers all the basic functionalities before actually teaching the TCP server how to use the HTTP protocol. Immediate port and address reusability is a required functionality in the project implementation. If you working on the project across operating systems i.e. Linux and Windows, then different implementations and socket APIs must be taken into account and exceptions should be made accordingly. Look into SO_REUSEADDR and SO_REUSEPORT and how and when socket connections are to be made and closed. Initially you might encounter several socket errors and exceptions while integrating multithreading, but make sure that this part is covered so that it will provide a strong foundation for HTTP implementation.

Logs Implementation

- Refer https://httpd.apache.org/docs/2.4/logs.html this official site of apache.
- If you have apache installed on your device then analyze the format of var/log/apache2/access.log and var/log/apache2/error.log with the format in the website.
- You will get Idea of how to implement.

Two types of logs: 1. Access logs => Keeps track of all the access requests made to the web server and the servers response to those requests

2. Error logs => Keeps a track of the errors encountered during any request response cycle and in general keeps track of the state of the server

Common format of a log file:

1. %>s HTTP status code for a request

2. %U URL of the specified resource

3. %a IP address of the client

4. %T Time required to make the request

5. %t Timestamp either in RFC 1123/1036/ANSI C time format

Decide on a specified format that is read from the config file of the server. Essentially keep two levels of logging => One for the access and the other for the errors like apache2

Also apache has an automatic log compressor that compresses the log file after a certain limit. Can add that to store old logs in a compressed format.

Cookies Implementation

Sources:

- HTTP MDN Cookies
- RFC-2625 State Management Mechanism

An HTTP cookie (web cookie, browser cookie) is a small piece of data that a server sends to the user's web browser. The browser may store it and send it back with later requests to the same server. Typically, it's used to tell if two requests came from the same browser — keeping a user logged-in, for example. It remembers stateful information for the stateless HTTP protocol. It is mainly used for 3 purposes: Session Management, Personalization and Tracking. Cookies are stored on the user's computer by the browser. Same cookies are not shared between browsers, but they may be used for tracking your activity across different browsers on A computer e.g. Session Cookie which signifies the start and end of a specific session. Cookies set by a certain website can be checked in Chrome DevTools or Firefox developer console. They are typically stored as a name-value pair.

For a basic implementation of Cookies in your HTTP server: The server should return a Set-Cookie header in the response which contains the cookie name-value pair. A cookie has several attributes as defined in the RFC like the Expires attribute which defines the time period after which the cookie will expire and the server will have to include the Set-Cookie header again, the Domain attribute which specifies all the hosts to which the cookie will be sent. An existing cookie value can be deleted by setting the Expires attribute to a date in the past. As the user-agent(typically a browser) handles cookies on the client side, the server only needs to make sure that the Set-Cookie header is included in the response if the Cookie header is not included in the request. The cookie name and attributes can be specified in the config file of the server so that the cookie value can simply be set as a uuid.

For further reference: (Not related to project implementation) There are several YouTube videos on authorization using cookies vs web tokens and its impact on security.

Server Configuration

The project expectation is to build an Apache-like server with some of its important functionalities. Apache server configuration by specifying configuration details in a config file like apache2.conf. Be sure to actually install apache2 in your system and explore it's file hierarchy and how basic configuration is done to get started. The config file contains important configurations like Document Root, Server name and other configs like cookies. The main functionality that the server code should provides whn it comes to the config file is parsing and setting the necessary specified configurations. Added support for comments inside the config file allow for a better documentation.

Parsing form data in POST request

A POST request is usually sent by an HTML form and the type of the body of the request which contains the data to be sent is indicated by the Content-Type header. Most of the browsers usually support two MIME types in the Content-Type header, i.e. application/x-www-form-urlencoded and multipart/form-data. The main purpose of both the types is to send a list of name-value pairs to the HTTP server.

In application/x-www-form-urlencoded, the name-value pairs are separated by '&' and the names and values are separated by '='. The spaces between the texts are replaced by a '+' and non-alphanumeric characters are %-encoded. Here, parsing is pretty straight-forward and wouldn't be tough to handle.

Multipart/form-data is where it gets a little complicated. It should be used for submitting forms that contain files, non-ASCII data, and binary data. Here, each value is sent as a block of data with a boundary string separating each part. We can get the boundary string for each request from the Content-Type header in the HTTP request. Each part has its own set of headers, mainly a Content-Disposition header with disposition type as 'form-data'. The field name is given in the Content-Disposition header's 'name' parameter. The headers and the value for each part are separated by a blank line. If file contents are submitted in a form, the media type of the file is specified in the Content-Type header. In this case, the filename is specified in the 'filename' parameter of the Content-Disposition header.

To test both the types, you can set the 'enctype' attribute to any one of these types in the HTML <form> element along with the 'method' attribute set to 'POST'.

Sources -

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST
- https://tools.ietf.org/html/rfc2388

to handle file upload in PUT/POST request

Text or simple encoded files are easy to handle, whereas files like pdf or images are a little tricky. The key to ensure that files are uploaded without corruption, is to NOT decode them. Most file formats use special bytes to hide/encrypt certain information and normal encoding/decoding schemes might not work. This results in corrupted files being uploaded. Following instructions can ensure no data is lost while uploading files to the server:

1. Get the request from the client. Make sure the size you are receiving is big enough, else you will lose data.

2. Figure out the content-type from the http headers.

3. For text(or simple encoding) based files:

- Decode the complete request.
- Split the request in 2 Categories :-

```
- Request headers
- File data (separated by a new line)
```

- Open the file to be created in 'w' mode.
- Write the file data in the newly created file.

4. For images(any binary) files :

- Split the request first into 2 Categories :-

```
- Request headers
- File data (separated by a new line)
```

- Decode the request headers.
- Create a byte string for the encoded file data.
- Get the size of file to be created and find the length of file data already received.
- Now, receive the data from client until the original file size is greater than the received data.
- Open the file to be created in 'wb' mode(bytes).
- Write the encoded data directly into the newly created file.

General tips on handling binary files:

1. Never decode raw data you get from socket. Some information might be lost.

2. Most of the times the 'Content-Length' header will be accurate, but do not rely on it. Sometimes, the 'Content-Length' in request will not match the actual size of the file.

3. Instead, a better way to do that is to parse the headers, calculate the length of headers in the request and then subtract it from the total length of the entire request. This will give the exact number of bytes in the file.

4. While writing into the file, use that part of the raw request data (which is not decoded) which contains the file bytes, directly.

to handle file If-Modified-Since header in GET request

If-Modified-Since header indicates the time at which the browser had last downloaded the webpage.

While parsing get request, if this header is received then follow these steps:

1. This header contains date in the <day-name>, <day> <month> <year> <hour>:<minute>:<second> GMT format. It can be parsed using the datetime library.

2. Find out the last modification date of the requested webpage using os.path.getmtime(filename).

3. Add the last-modified header(contains the last modification date of webpage) in the response headers.

4. If the requested webpage has been modified after the given date, send a response body with status code 200.

5. If the requested web page has not been modified , send an empty response body with status code 304.

In case you do not receive the if-modified-since header while using the browser, Postman can be used. Add If-Modified-Since header and date under the Headers section of Postman.

to start and stop the HTTP server

- Must work just like the way Apache server starts and stops, i.e., with a **single command.**
- Must run in **background**, i.e., even after closing the terminal.
- Can make two shell scripts; start.sh and stop.sh
- To stop the server from running in the background, kill the process using the process ID.
- Useful resource for running a python program in **background** : https://www.geeksforgeeks.org/running-python-program-in-the-background/
- Useful resource for killing a process : https://www.geeksforgeeks.org/kill-a-process-by-name-using-python/

Automated Testing

Two libraries that are helpful for automatic testing of your code are:-

- https://realpython.com/python-requests/#other-http-methods
- https://docs.python.org/3.5/library/webbrowser.html

-> Using the requests module, we can implement all HTTP methods and test our code.

```
For eg. For a GET request, requests.get('https://api.github.com') is used.
```

-> Webbrowser module is helpful if we want to test our code on the browser. We can give URL as the argument and it will open a new browser tab with that URL.

```
  For eg.  webbrowser.open_new_tab('https://api.github.com')
```

Retrieved from "http://foss.coep.org.in/coepwiki/index.php?title=HTTP_Server_Project&oldid=2328"

Navigation menu

Personal tools

- Log in

Namespaces

- Page
- Discussion

Variants

Views

- Read
- View source
- View history

More

Search

Search Search Go

Navigation

- Main page
- Recent changes
- Random page
- Help

Tools

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link
- Page information

-