INF-2300

Assignment 1, HTTP server

Magnus Dahl-Hansen

September 2022

Introduction

This report describes the implementation of different functions that handles HTTP-requests, the tests for these functions and the attempt to implement a RESTful API. The implemented functions that handles the HTTP-request are *GET, POST, PUT* and *DELETE.*

Technical background

Before reading how the code was implemented, the reader of this report should be familiar with: HTTP, HTTP requests/responses, JavaScript Object Notation (JSON) formatted files and RESTful API.

HTTP stands for Hypertext Transfer Protocol and is the most common protocol to fetch resources and load web pages over the internet. By sending HTTP-requests different internet communication platforms are able to ask for resources and information that they need to load a website. A HTTP-request as implemented in this code consists of five parts: method, a filepath/URL, HTTP version type, request headers and a entity body. The methods described in this report are the GET method which requests some information from a server, POST that posts information to the server, PUT which replaces some information on the server and DELETE which deletes information from the server. The filepath/URL is used to specify which file the the different methods are going to handle, the HTTP-version specifies the version of HTTP, the HTTP headers is used as extra information to the request and lastly the entity body is data that may be appended or used in some way according to the different methods.

A HTTP response is what the client/user of the server receives after a HTTP-request is sent to a server. These responses vary according to the request but mainly consists of a HTTP status code, response headers and a body. The HTTP status code is used to indicate (among other things) whether the HTTP request was successful or not, if there was any errors or if there was any re-directions of some sort. The HTTP response headers gives extra information about the response and the body is the actual information retrieved from the HTTP request.

Moving over to JSON and the RESTful API. JSON is a standard text based file format that represents data based on JavaScript syntax. It is commonly used as a standard to send data between a client and a server so that the data can be displayed on a web-page. The RESTful API stands for Representational State Transfer Application Programming Interface and is basically a standardized software architecture that is used for communication between two applications or between a client and a server.

Design&implementation

There are twelve different functions implemented in the pre-code that test the implementation of HTTP methods. These tests will among other things send a HTTP-request that can be read and used by the implemented HTTP methods. In this implementation, each and every line from the HTTP-request will be parsed and the method, URL and version will be placed inside a dictionary along side with the HTTP-headers and entity body of the request. This is done to make it easier to handle the different parts of the request.

Before the request can be handled by it’s respective method a check is made to see if the string “../” is contained within the URL sent by the request. If this string is contained in the URL it is considered to be a traversal attack and a *“403 - forbidden”* response is written to the client to indicate that this operation is not allowed. If the URL is not considered to be a traversal attack then the HTTP-method is fetched from the dictionary containing information about the request.

In the case where the HTTP-request is a *GET,* the implemented *GET* function is called. The implemented tests specifically test a *GET* method up against three different files. If the requested file is “server.py” a “403 – forbidden” response is expected, while a request to “text.txt” file or “messages.json” file should handle these two files specifically. A call to other files may be done, but is not expected to be handled in this assignment. However, should a *GET* request to “text.txt” file be made (and the file exists), then the file is opened, its content is read and written to the response body. The body is then sent as a response along with the “200 – OK” staus code and the content-length and content-type headers. Should a file not exist then a “404 – not exists” response is written.

If the HTTP-request is a POST request, then a new file will be created and the entity body is written to the file. If The file already exists then the entity body will be appended to the file. If the file-type is “text.txt” the built in python open() function is used to append the given data to the file. If the file is of type “text.json” then the function dump() from the “json” library is used to append the data to the file. In both cases the process is terminated by closing the file and writing a status-line, headers and response body as a HTTP-response.

The *PUT* function is specifically implemented to replace a message based on an ID in a “json” file. This is done by loading data from the “json” file into a list by using the *json.load()* function. The entity body from the request is placed inside a dictionary with values corresponding to its “id” and “text” keys. The dictionary is then appended to the list and written back to the “json” file. Lastly the response-code “200 – OK” is sent as a response. If the “json” file were to be empty and no message could be replaced, then the error message “304 – Not Modified” is responded with.

The last implemented function is *DELETE*. This function opens a given “json” file and asks the user for a specific message ID that is going to be deleted. The dictionaries within the “json” file is then searched until the ID is found and deleted. If the ID is deleted then the “200 – ok” response is sent and the “404 – not found” response is sent if the ID is not found.

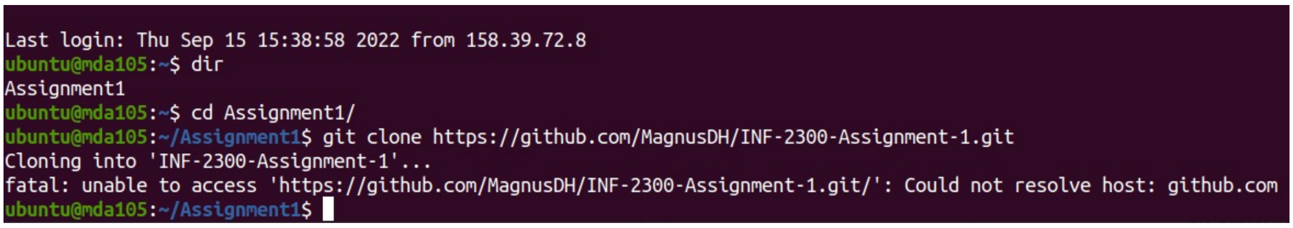
Including to the implementation of the HTTP method handlers some test extra tests have been implemented. These tests are simple and will test if it is possible to fetch content, create, write to, replace or delete content from a “json” file. The tests simply sends a HTTP request but does not check for a valid response. Instead the file have to be checked manually to see if the output is correct.

Results&discussion

The implemented code is able to successfully complete all the required tests from test_client.py. The extra implemented tests from the “RESTfulAPI_tests.py” are also able to get information from a json-file, create and write/append to a json-file and replace/delete messages from a json-file. However there are some minor bugs with the interface of these tests. Some methods have to be repeated for them to correctly handle a file. The reason for this is still unknown but it may be cause by two functions waiting for input from a user simultaneously. These methods are the PUT and DELETE functions, which both needs input from the user.

Another minor bug occurs when deleting a message a json-file. When an ID is deleted the other ID’s are not organized and replaced with correct ID’s. For example: if there are 3 messages in the json-file and message with ID: 2 is deleted, then the messages in the json-file will have ID’s 1 and 3, instead of 1 and 2.

Running *server.py* works and will load the *INF-2301 Web Server* web-page. Entering text on this site is also completed correctly. However, this is only works locally since the code was not copied properly to the actual server. Attempts to copy the code to the server has been done by using the Secure Copy (scp) function and “git clone”, but none has worked. An attempt to make the github repository as “public” before cloning has also been proven unsuccessful and responds with this error message:



```
Last login: Thu Sep 15 15:38:58 2022 from 158.39.72.8
ubuntu@mda105:~$ dir
Assignment1
ubuntu@mda105:~$ cd Assignment1/
ubuntu@mda105:~/Assignment1$ git clone https://github.com/MagnusDH/INF-2300-Assignment-1.git
Cloning into 'INF-2300-Assignment-1'...
fatal: unable to access 'https://github.com/MagnusDH/INF-2300-Assignment-1.git/': Could not resolve host: github.com
ubuntu@mda105:~/Assignment1$ 
```

Conclusion

This assignments task was to create a HTTP server that could handle *GET*, *POST*, *PUT* and *DELETE* requests. The required tests from the pre-code are all passed correctly and the local host server is working. However, due to troubles with copying the implemented code to the private course server the application must be tested locally.