INF-2301 Computer Communication and Security
INF-2301

Alexander Saaby Einshøj

September 11, 2015

1 Introduction

This assignment is divided into two components; Part A and Part B. Part A is to implement a simple web server, whereas Part B, implementing a RESTful application, which is a client server based software architecture.

2 Technical Background

For both parts of this assignment, Python 2.7[6] was used. RFC 7230-7235 were also essential parts of this assignment.

3 Design

Whenever I speak of 'this task' or 'this assignment' I refer to both Part A and Part B of the assignment. If any part of my report includes only one of the parts, this will be declared explicitly.

Before even reading Part B of this assignment, I tried to understand as much as I could of what was required of me in Part A, and not to confuse myself more than needed by the natural difficulty of the task itself.

In Part A, we were required to implement a simple web server. My implementation is required to be able to support both the 'GET' and 'POST' HTTP request codes. After reading up on RFC's, I decided on the current design for my program.

The design choices for the different parts ended up being quite unlike each other, due to the fact that I didn't really prepare any design choices at all for Part A. In this part, my thoughts were to understand the specifics of the task I was given, in addition to reading as much as possible before I wrote a single line of code. Actually writing the code resulted badly, atleast for Part A, when I realized there was still a lot of syntax and libraries to read up on, and not least, understand.

For Part B on the other hand, I had learned my lesson from the first part of the assignment, knowing much more of the needed syntax, and different thoughts on design.

3.1 Part A

This part I ended up not making object oriented, which in general might defeat the purpose in a language like Python, but since this required quite little code, I believe it is justified. The code consists of the server initiation, and reading the correct information from the client at the appropriate point in time. That in it self can be a difficult task, if one doesn't know the HTTP request and response protocols. After reading up on those, it was really more of a syntax-issue, and was quickly resolved.

The server needs to be able to identify whether the client makes a 'GET' or a 'POST' request, and this was resolved within two if's, one for each request method.

The 'GET'

All i had to do for this part, was writing the status code to the client, followed by trying to open requested path. If the path requested was '/index', the cleint would recieve a readable version of the 'index.html' file.

The 'POST'

For this method, I also had to recieve and append the 'Content-Length' header to the html-file. After that being done, the server could send the 'ok' to the client, along with the newly posted content.

3.2 Part B

This part of the assignment ended up being more complicated, which is also the reason why this is made object oriented, as opposed to what was my solution of Part A. The program consists of one class and several methods

Init

The initialization of the program sets up the server, in addition to creating some useful variables.

The 'GET'

The GET method for this part simply checks if the requested xml-file is empty, and sends an 'ok' status message with the exception that the xml file

is empty. If it's not empty, the xml file is opened no matter what path is requested.

The 'POST'

The POST for Part B turned out to be quite difficult, and required a more deep understanding of the HTTP protocols, and the syntax used to extract the wanted information. As the POST in this part had a payload which needed to be given a unique id based on the id's currently existing in the xml document used, the xml document needed to be parsed, allowing us to do just that. With the xml document being parsed, it was easy to find out the id's currently in the document, and assigning a new one for the posted message. The message in the payload needed to have the '+' symbols replaced with spaces, and the quotes removed before being put in the document. When all this was done, and the new 'element' or the payload was added, all that remained was the response message to the client. Along with the 'ok' message, the content-length was sent, as well as the newly created id for the payload recieved.

The 'PUT'

The PUT had an id and a message field as payload, and was to replace the message at the given id, with the new posted message. This required me to parse the payload to first find the id, then the payload. With this being done, all I had to do was to find the given id in the xml document, and set the posted value to replace the old value.

The 'DELETE'

The delete request took an id field as payload, and was to delete the message at the given id. This was really quite simple when I found the id in the document, which I had already done once in the 'PUT'

4 Implementation

This program has been developed to run on PC's, and Python has been used to make it. Linux Mint 17.2 has been used to test and develop the program.

5 Results and Discussion

5.1 Part A

I will start out with saying that Part A could have been significantly improved by making it object-oriented so it would become more readable, and it would also serve better as a general solution. As of now, it looks quite messy, but works well for the purpose it was created. All of us who did this assignment, were told not to recieve a specific amount of bits from the client when handling requests, but rather creating makefiles that could handle whatever was sent to the server, making it a more solid solution overall. Other than that, I think my solution of Part A is just fine. I tested it using the browser, both with Google Chrome and Mozilla Firefox, and it worked for both. I used the 'socket' and the 'os' library for this part, which I think was sufficient, with the 'socket' library being the most significant, and was suggested for us to use in the assignment text.

5.2 Part B

With this part, we were given a 'test.py' file to test if our program ran as it should. I chose not to use this test-file, as it told me my solution was wrong, even though it works great when using my own methods for testing. I used 'Requests', which is a python library, to make my own requests to the server and see if I got the results I wanted, and I did.

I chose to make this part object-oriented, which in retrospect seemed almost a necessity, as it added a nice overview and flow to the program. So as far as the results go, I would say that Part B works as it should. I don't know how I could have made this any better, as I made it the best way I knew how. It might not be a very general solution though, as it is created to serve the specifics of the task given. If I were to make a more general approach to the assignment, I might have dug deeper into the parsing syntax, enabling me to make something that would always work, not only for this assignment.

6 Conclusion

All in all, both parts of the assignment do work, though Part B doesn't work when testing against the 'test.py' file. The requirements of the assignment were fullfilled, as far as I know.

7 References

http://tools.ietf.org/html/rfc7230

http://tools.ietf.org/html/rfc7231

http://tools.ietf.org/html/rfc7232

http://tools.ietf.org/html/rfc7233

http://tools.ietf.org/html/rfc7234

http://tools.ietf.org/html/rfc7235

https://docs.python.org/2/library/socket.html

https://docs.python.org/2/library/xml.etree.elementtree.html

https://docs.python.org/2.6/library/os.path.html