

Anna Helana Maliakal

Student number: 17324430

Web-Proxy Server

Advanced Telecommunications – PROJECT 1

# **Introduction**

The aim of this project was to implement a working solution of a Web Proxy Server. The implementation would run on a local server and retrieve items from the internet on behalf of a Web client instead of the client fetching them directly.

The program should be able to:

1. Respond to HTTP & HTTPS requests and should display each request on a management console. It should forward the request to the Web server and relay the response to the browser
2. Handle websocket connections
3. Dynamically block selected URLs via the management console.
4. Efficiently cache requests locally and gather timing to prove the efficiency of the proxy.
5. Handle multiple requests simultaneously

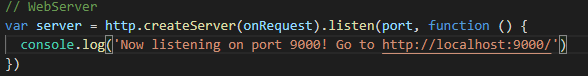
# **Approach**

I chose to implement my solution using NodeJS. NodeJS uses an event-driven I/O model which makes the application extremely light and scalable. Therefore, the code executes faster than most languages. Also, NPM (Node Packet Manager) makes many modules/libraries available which make the programming experience more productive. Given the above reasons and my previous experience with JavaScript, I decided to use NodeJS.

# **Implementation**

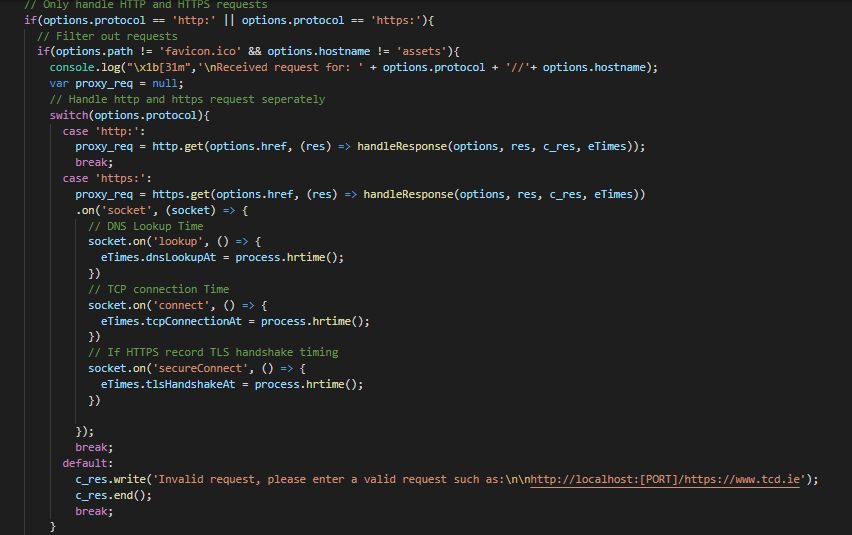
## Web Server

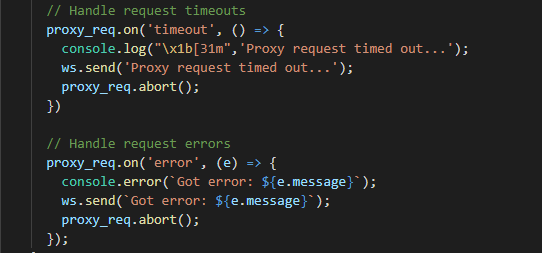
First step was to create a web-server. In NodeJS, this can be done through the use of the HTTP library. This creates a server in a designated port of the local machine. Once created, requests are received.



## Handling Requests

Once requests are received, *onRequest()* is called. As shown in the snippet below, *onRequest()* only handles HTTP and HTTPS requests. Both these are done using the HTTP library (mentioned above) and the HTTPS library. The significant time details of the HTTPS request are noted. Error handling was also added to check for timeout scenarios and error messages received.



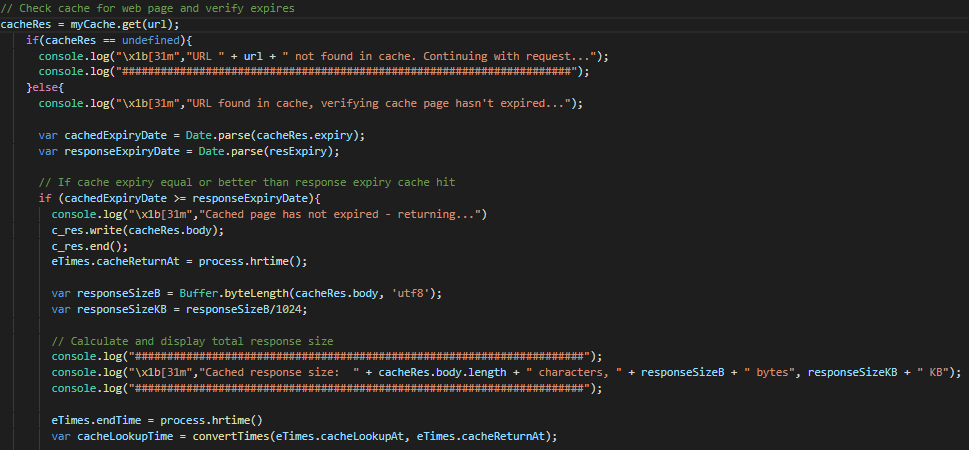


## Handling Responses

All responses are handled by *HandleResponse()*. This checks if the URL is in the blacklist and proceeds if it isn’t. if the response received is a 200 OK, the program proceeds, otherwise it pauses and prints the error. Then the program moves onto the caching.

## Caching

Caching is implemented in the program with the assistance of the node-cache library in NodeJS. It stores each URL as a key in a hashmap. It checks if the URL is in the cache and if it is uses that response. However, if it is not in the cache it forwards the request and caches the response.



## Blocking/Unblocking URLs

Users can block and unblock URLs from the terminal by typing *block [URL]* or *unblock [URL].* Blocking adds the URL into a hashtable. Unblocking checks if the URL is in the hashtable, if it is then it is removed.



## WebSocket server

The WebSocket server provide a persistent connection between a client and server that both parties can use to start sending data at any time. Once the connection is received, user can make a request through the terminal and if it is a valid request, the data is returned to the console.





## Additional Functionality

## 

I have added some extra features, including changing the port and host from console, and also terminating the program from the console.