NT Assignment HTTP and Web Services

HTTP, REST

Introduction

In this assignment we are going to explore the HyperText Transfer Protocol ([HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)[[1]](#footnote-1)) and the REST architecture.

HTTP is an Internet application protocol. It is the workhorse of the Web. HTTP enables clients to create, retrieve, update or delete web resources. It is used by browsers to fetch web pages (and associated images) from the web sites. It is also used by software programs to access remote Web Services.

HTTP is a client/server protocol. An HTTP server (e.g. Apache or IIS) is listening on a port (usually 80) for requests from clients. It will respond to HTTP requests received on this port. Servers do not take initiative. They are just there to serve. A client that wants a representation of a specific resource like a web page, or wants to get some data processed, has to connect to the right host/port combination, submit a valid HTTP request, and wait for the response from the server.

Some very useful introductory reading on HTTP:

* HTTP Made Really Easy  
  <http://www.jmarshall.com/easy/http/>
* How the web works: HTTP and CGI explained

<http://www.garshol.priv.no/download/text/http-tut.html>

Task 1: Performing HTTP requests manually (putty)

First we will try to communicate with a web server by manually submitting HTTP requests. We will use PuTTY to communicate with a webserver. To obtain PuTTY on your Ubuntu node, just enter the following command:

sudo apt-get install putty

To send a HTTP Request with PuTTY you have to do the following

* Enter ‘Host Name (or IP address)’: e.g. [www.example.com](http://www.example.com)  
  (enter ‘localhost’ if your server is running on your own PC)
* Enter ‘Port’: 80 (could be different on some host)
* Select ‘Connection Type’: Raw
* Select ‘Close window on exit’: Never
* Press ‘Open’

Perform the following tests and answer all questions:

1. Open your favorite browser and enter the URL [www.example.com](http://www.example.com). Look what happens.
2. Let us do what the browser does, but manually, step by step. Connect to [www.example.com](http://www.example.com) (with PuTTY). Type the following two lines in the PuttY window and after that hit the <enter> key twice:  
     
    GET /index.html HTTP/1.1  
    Host: www.example.com

What response do you get? What does it mean?

Text

Description automatically generated

It shows the html information contents in putty

Task 2: Using cURL to do HTTP

In this assignment you are going to use cURL to talk to an HTTP server.

To obtain cURL on your Ubuntu node, just enter the following command:

sudo apt-get install curl

Note: for the last task you’ll also need CURL. So, if you are going to use your Arduino IDE on e.g. Windows, you can better install CURL on Windows too.

Get acquainted with cURL first. Here are some useful links to get you going:

* <http://curl.haxx.se/docs/httpscripting.html>
* <http://www.codediesel.com/tools/6-essential-curl-commands/>
* <http://www.thegeekstuff.com/2012/04/curl-examples/>

Note: cURL built-in help can be activated with the command ‘curl --help’

Now do the following tasks (and test if your answers work!):

* Define a cURL command to do everything (except nice display) that the browser did in task 1.

Write down the used curl command.

Curl http://www.example.com

* Define a cURL command to download and save the Google logo from the main page of the website [www.google.com](http://www.google.com).

What extension (based on image type) should the downloaded file have? How can you find that in the HTTP response header?

Write down the used curl command.

Using ‘curl <http://www.google.com>’ and all the way to the top there is a png file called `googlelogo\_color\_272x92dp.png`.

To download the png you have to input

‘curl googlelogo\_color\_272x92dp.png > image.png`

* Define a cURL command that does the same POST as the HTML-form in the following w3schools.com example:  
  <http://www.w3schools.com/tags/tryit.asp?filename=tryhtml_form_method_post>

Write down the used curl command.

The syntax to post as the HTML-form is `curl -d POST [options] [URL]`

curl -d "fname=First Name&lname=Last Name&submit=submit"

www.w3schools.com/action\_page.php

Task 3: Apache Web Server in the Kathara Lab

For this assignment we’re going to reuse Kathara lab from the previous assignments. We’re going to use *WebServer* node to set up simple Apache Web Server. Our client will be the *Laptop* node.

Find out a start up command for the Apache server and add it to WebServer.startup.

You can check your correct configuration by entering the following command on *WebServer*:

/etc/init.d/apache2 status

The default apache2 setup offers a test html page located in /var/www/index.html. Adjust this page a little bit.

Start the *lynx* browser from your *Laptop* node and connect to your *WebServer* You should now be able to see the default page from the *WebServer*.

Provide a screenshot of what you see in lynx after successful connection to the *WebServer*.

Text

Description automatically generated

Task 4: REST on ESP32 board

In this task we are going to set up a simple REST server on our ESP32 board.

To set up your ESP32 board e.g. on Arduino IDE you can use this link:

<https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

You can use e.g. “ESP32 dev module“ as a board in the Arduino IDE.

Once your ESP32 is up and running you can set up your own REST server.

You can use the framework code provided in t-sem3-cb-code/com/nt\_ass3.

This is ESP32-adjusted code from <https://www.mischianti.org/2020/05/16/how-to-create-a-rest-server-on-esp8266-and-esp32-startup-part-1/>

Study the code and try to understand it. If you filled-in the credentials of your WIFI network correctly, you should be able to access your server on: <http://esp32.local>. The MDNS protocol should take care of it (<https://en.wikipedia.org/wiki/Multicast_DNS>).

Hall Sensor:

Provided ESP32 has a Hall Sensor inside.

Study the working of it (<https://randomnerdtutorials.com/esp32-hall-effect-sensor/>).

Add a HTTP GET Request to your code and find out the value of your Hall sensor. Try to change this value (by using a magnet) and see the difference on your web interface. Provide a screenshot.

Graphical user interface, text, application

Description automatically generated

Without magnet

Text

Description automatically generated

With magnet

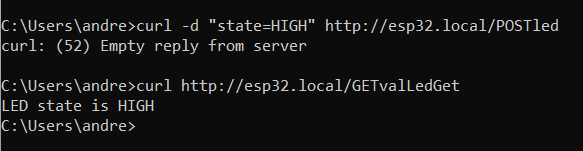
Add a HTTP GET Request to your code to get the value of your built in LED. Provide a screenshot.

Graphical user interface, text

Description automatically generated

Add a HTTP Post Request to your code to change the value of your built in LED.

Provide a screenshot of the changed value.



*Text

Description automatically generated*

*Note: You can’t do easy POST through URL. You can use a CURL command to do that. Consult Internet on how to do it.*

*Note: In the provided framework file there is a commented function setBuiltInLed() that shows you how to get value from the POST request and how to send HTTP OK back to the client.*

Provide the CURL command you have used to switch the built-in LED on.

curl -d “state=LOW” <http://esp32.local/POSTled>

curl -d “state=HIGH” <http://esp32.local/POSTled>

Submit your final ESP32 code (without WIFI credentials -) in Canvas.

1. HTTP: <http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol> [↑](#footnote-ref-1)