National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member:______________ **Dated:** _________________

Semester:____________________ **Section:** _________________

EE-357 Computer and Communication Networks

Experiment – 11

Introduction to Wamp Server

		PLO5/ CLO3		PLO5/ CLO3	PLO5/ CLO3	PLO5/ CLO3
Name	**Reg. No**	**Viva / Quiz / Lab Performance** **5 Marks**	**Analysis of data in Lab Report** **5 Marks**	**Modern Tool Usage** **5 Marks**	**Ethics and Safety** **5 Marks**	**Individual and Team Work** **5 Marks**
Noor-ul-Ain Ansar	**284825**					
Myesha Khalil	**305093**					

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

EXPERIMENT NO 11

Introduction to Wamp Server

Objectives

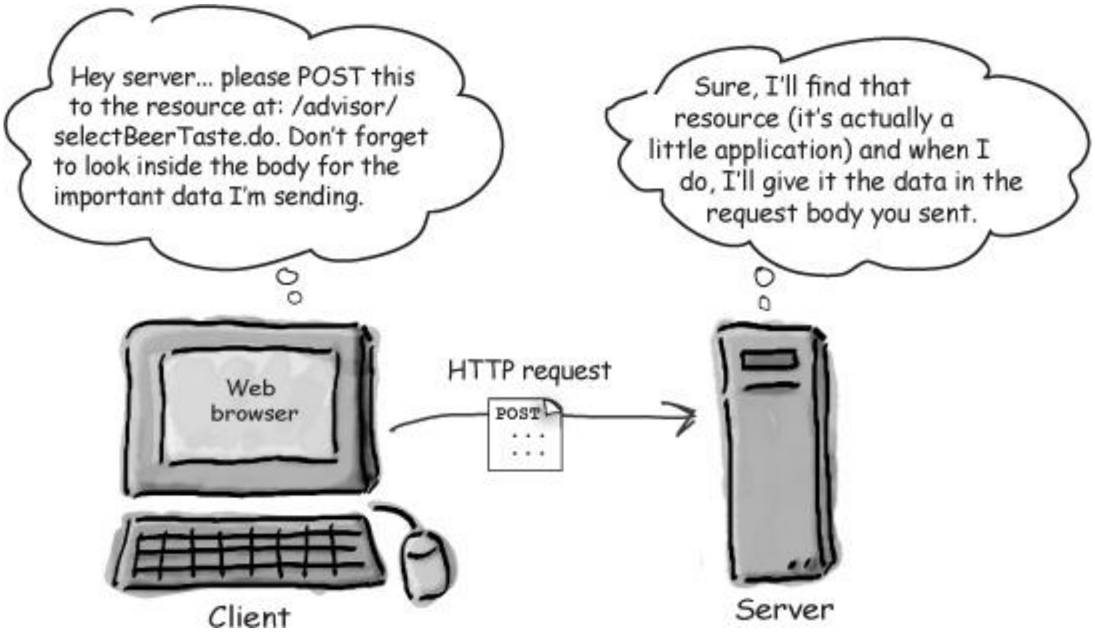
To become acquainted with the Wamp server, ESP node32 wireless device and transport protocols.

Introduction

In this experiment we will learn **how to send post request** to a web page using **NodeMCU or ESP8266**. As we know all webpages are HTTP protocols, GET and POST are methods of communicating between web browser and the server. Also we look towards server side **php** coding. If you are looking for GET method read here.

What is HTTP?

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.


Hey server... please POST this to the resource at: /advisor/ selectBeerTaste.do. Don't forget to look inside the body for the important data I'm sending.
Sure, I'll find that resource (it's actually a little application) and when I do, I'll give it the data in the request body you sent.
Web browser
HTTP request
POST
Client
Server

HTTP works as a request-response protocol between a client and server. Each Hypertext Transfer Protocol (HTTP) message is either a request or a response. A server listens on a connection for a request, parses each message received, interprets the message semantics in relation to the identified request target, and responds to that request with one or more response messages. A client constructs request messages to communicate specific intentions, examines received responses to see if the intentions were carried out, and determines how to interpret the results. A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

Two HTTP Request Methods: GET and POST

Two commonly used methods for a request-response between a client and server are: GET and POST.

- **GET** – Requests data from a specified resource
- **POST** – Submits data to be processed to a specified resource

POST

The POST method requests that the target resource process the representation enclosed in the request according to the resource’s own specific semantics. For example, POST is used for the following functions (among others):

1. Providing a block of data, such as the fields entered into an HTML form, to a data-handling process;
2. Posting a message to a bulletin board, newsgroup, mailing list, blog, or similar group of articles;
3. Creating a new resource that has yet to be identified by the origin server; and
4. Appending data to a resource’s existing representation(s).

An origin server indicates response semantics by choosing an appropriate status code depending on the result of processing the POST request; almost all of the status codes defined by this specification might be received in a response to POST (the exceptions being 206 (Partial Content), 304 (Not Modified), and 416 (Range Not Satisfiable)).

The POST Method

Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:

```
POST / HTTP/1.1
Host: foo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 13
say=Hi&to=Mom
```

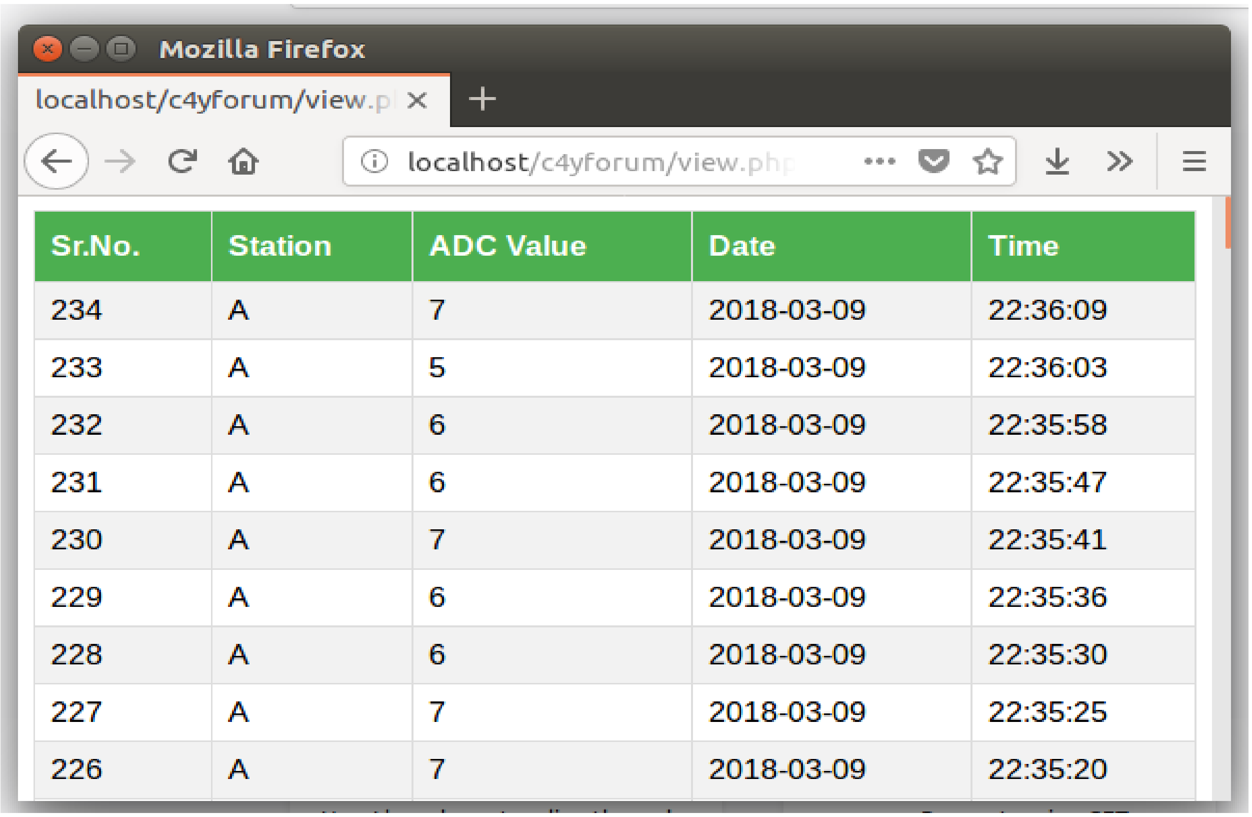
Some other notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

Procedure:

- Download and install Arduino IDE in your system and install esp 32 program libraries from the boards manager. Visit the following URL to get more info about how to install esp node 32 libraries in arduino IDE.
- Once you are done with the libraries, use the code provide alongside with this manual to upload on the ESP32 by serial communication. Don't forget to enter the SSID name and password according to your availability. Also mention the ip address of the Wamp webserver running on your local machine. Select Firebeetle-ESP32 as the board available and choose appropriate com port.
- Wamp sever need to be separately installed on a local PC and it should be running and online. After installation of wamp server, open "httpd.conf" file in the apache setting of wamp and replace "Deny from all" with "allow from all" in whole document.
- Also note that wamp server uses http services and **port 80** is dedicated by default to http. This port should be free and no other service should be using this port. If wamp is not going online, you can either change the port number to some other available ports or just run command prompt in administrator mode and type **"net stop http"** and press **"Y"**. It will stop all other services using http protocol. After restarting all services by wamp, it will go online.
- You can type "localhost" or "localhost: port number" to check if your local webserver is running or not.
- In the installation folder of wamp, place all the codes of ESP webserver provided separately inside the **"www"** directory. It will then be visible on your localhost page.
- Open ESP webserver from localhost, and run each program once sequentially. It will create databases on your server and start listening to data posted by esp32 and display it on localhost/espwebserver/view.php.
- Keep it in mind that this server is created locally and is accessible only within the network to any connected device. So you also view the data sent by ESP through your mobile phones by connecting to the same router.
- You can edit the data fields in the ESP code to send your required data and monitor it on any PC within same network.

- If successfully configured, your local host should look like this.

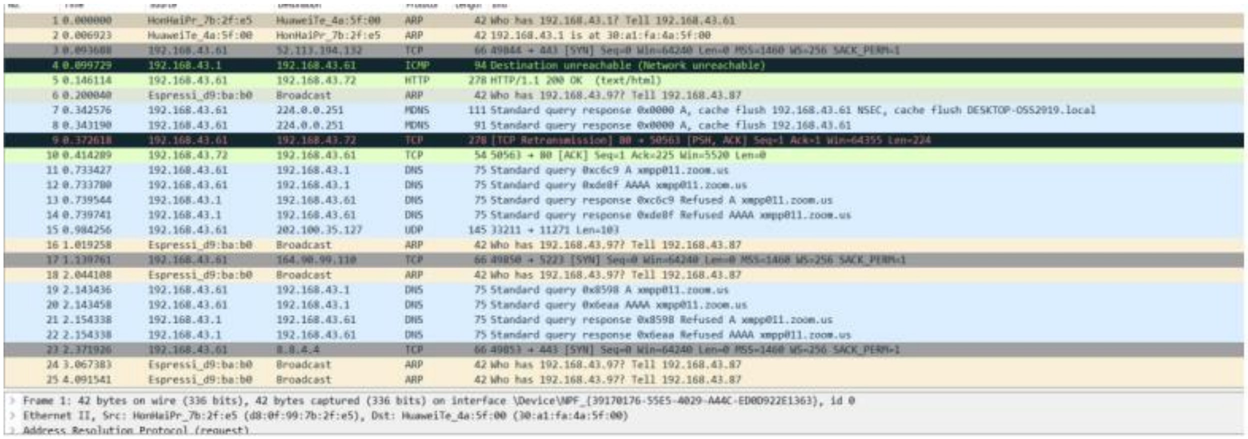


Sr.No.	Station	ADC Value	Date	Time
234	A	7	2018-03-09	22:36:09
233	A	5	2018-03-09	22:36:03
232	A	6	2018-03-09	22:35:58
231	A	6	2018-03-09	22:35:47
230	A	7	2018-03-09	22:35:41
229	A	6	2018-03-09	22:35:36
228	A	6	2018-03-09	22:35:30
227	A	7	2018-03-09	22:35:25
226	A	7	2018-03-09	22:35:20

Student Activity

Install and configure wamp server and send packets through esp nodemcu to the server. You have to identify the different types of protocols used in the transmission using Wireshark and pointout the data sent. Paste screenshots of your work below.

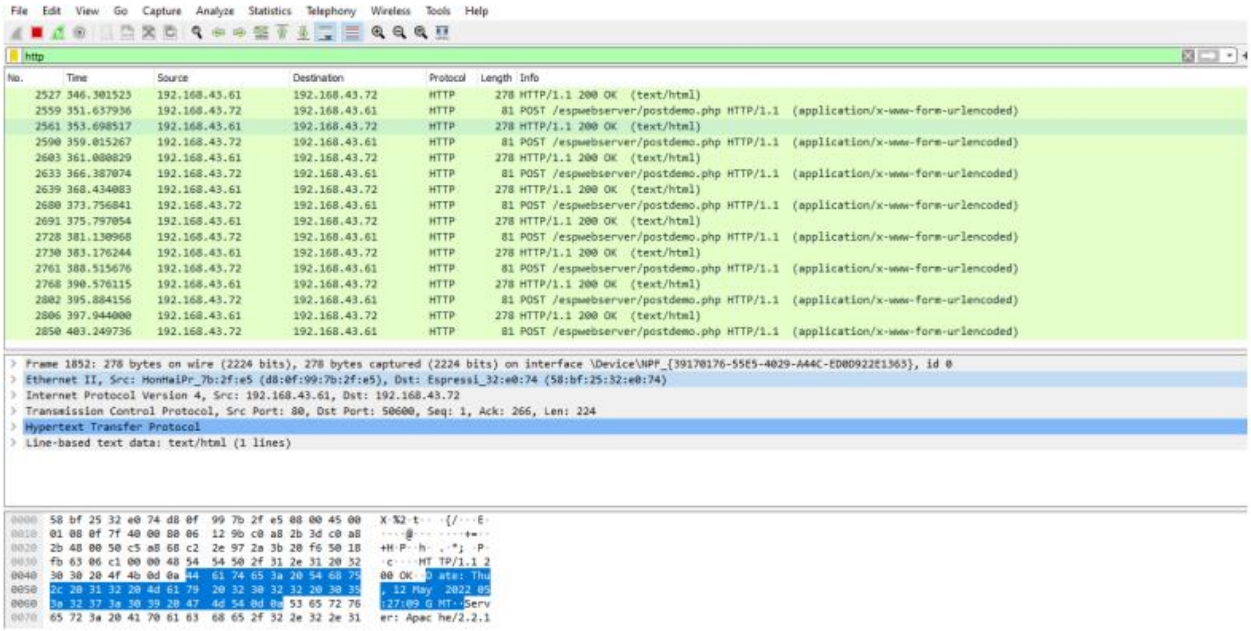
Wireshark SC:



```
 1 0.000000  HonHaiPr_7b:2f:e5   HuaweiTe_4a:5f:00   ARP    42 Who has 192.168.43.1? Tell 192.168.43.61
 2 0.006923  HuaweiTe_4a:5f:00   HonHaiPr_7b:2f:e5   ARP    42 192.168.43.1 is at 30:a1:fa:4a:5f:00
 3 0.093688  192.168.43.61       52.113.194.132      TCP    66 49844 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
 4 0.099729  192.168.43.1        192.168.43.61       ICMP   94 Destination unreachable (Network unreachable)
 5 0.146114  192.168.43.61       192.168.43.72       HTTP  278 HTTP/1.1 200 OK  (text/html)
 6 0.200040  Espressi_d9:ba:b0   Broadcast           ARP    42 Who has 192.168.43.97? Tell 192.168.43.87
 7 0.342576  192.168.43.61       224.0.0.251         MDNS  111 Standard query response 0x0000 A, cache flush 192.168.43.61 NSEC, cache flush DESKTOP-0SS2919.local
 8 0.343190  192.168.43.61       224.0.0.251         MDNS   91 Standard query response 0x0000 A, cache flush 192.168.43.61
 9 0.372618  192.168.43.61       192.168.43.72       TCP   278 [TCP Retransmission] 80 → 50563 [PSH, ACK] Seq=1 Ack=1 Win=64355 Len=224
10 0.414289  192.168.43.72       192.168.43.61       TCP    54 50563 → 80 [ACK] Seq=1 Ack=225 Win=5520 Len=0
11 0.733427  192.168.43.61       192.168.43.1        DNS    75 Standard query 0xc6c9 A xmpp011.zoom.us
12 0.733780  192.168.43.61       192.168.43.1        DNS    75 Standard query 0xde8f AAAA xmpp011.zoom.us
13 0.739544  192.168.43.1        192.168.43.61       DNS    75 Standard query response 0xc6c9 Refused A xmpp011.zoom.us
14 0.739741  192.168.43.1        192.168.43.61       DNS    75 Standard query response 0xde8f Refused AAAA xmpp011.zoom.us
15 0.984256  192.168.43.61       202.100.35.127      UDP   145 33211 → 11271 Len=103
16 1.019258  Espressi_d9:ba:b0   Broadcast           ARP    42 Who has 192.168.43.97? Tell 192.168.43.87
17 1.139761  192.168.43.61       164.90.99.110       TCP    66 49850 → 5223 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
18 2.044108  Espressi_d9:ba:b0   Broadcast           ARP    42 Who has 192.168.43.97? Tell 192.168.43.87
19 2.143436  192.168.43.61       192.168.43.1        DNS    75 Standard query 0x8598 A xmpp011.zoom.us
20 2.143458  192.168.43.61       192.168.43.1        DNS    75 Standard query 0x6eaa AAAA xmpp011.zoom.us
21 2.154338  192.168.43.1        192.168.43.61       DNS    75 Standard query response 0x8598 Refused A xmpp011.zoom.us
22 2.154338  192.168.43.1        192.168.43.61       DNS    75 Standard query response 0x6eaa Refused AAAA xmpp011.zoom.us
23 2.371926  192.168.43.61       8.8.4.4             TCP    66 49853 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
24 3.067383  Espressi_d9:ba:b0   Broadcast           ARP    42 Who has 192.168.43.97? Tell 192.168.43.87
25 4.091541  Espressi_d9:ba:b0   Broadcast           ARP    42 Who has 192.168.43.97? Tell 192.168.43.87

> Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{39170176-55E5-4029-A44C-ED0D922E1363}, id 0
> Ethernet II, Src: HonHaiPr_7b:2f:e5 (d8:0f:99:7b:2f:e5), Dst: HuaweiTe_4a:5f:00 (30:a1:fa:4a:5f:00)
> Address Resolution Protocol (request)
```

HTTP Packets SC:



```
No.     Time        Source          Destination     Protocol Length Info
   2527 346.301523  192.168.43.61   192.168.43.72   HTTP     278 HTTP/1.1 200 OK  (text/html)
   2559 351.637936  192.168.43.72   192.168.43.61   HTTP      81 POST /espwebserver/postdemo.php HTTP/1.1  (application/x-www-form-urlencoded)
   2561 353.698517  192.168.43.61   192.168.43.72   HTTP     278 HTTP/1.1 200 OK  (text/html)
   2590 359.015267  192.168.43.72   192.168.43.61   HTTP      81 POST /espwebserver/postdemo.php HTTP/1.1  (application/x-www-form-urlencoded)
   2603 361.080829  192.168.43.61   192.168.43.72   HTTP     278 HTTP/1.1 200 OK  (text/html)
   2633 366.387074  192.168.43.72   192.168.43.61   HTTP      81 POST /espwebserver/postdemo.php HTTP/1.1  (application/x-www-form-urlencoded)
   2639 368.434083  192.168.43.61   192.168.43.72   HTTP     278 HTTP/1.1 200 OK  (text/html)
   2680 373.756841  192.168.43.72   192.168.43.61   HTTP      81 POST /espwebserver/postdemo.php HTTP/1.1  (application/x-www-form-urlencoded)
   2691 375.797054  192.168.43.61   192.168.43.72   HTTP     278 HTTP/1.1 200 OK  (text/html)
   2728 381.130968  192.168.43.72   192.168.43.61   HTTP      81 POST /espwebserver/postdemo.php HTTP/1.1  (application/x-www-form-urlencoded)
   2730 383.176244  192.168.43.61   192.168.43.72   HTTP     278 HTTP/1.1 200 OK  (text/html)
   2761 388.515676  192.168.43.72   192.168.43.61   HTTP      81 POST /espwebserver/postdemo.php HTTP/1.1  (application/x-www-form-urlencoded)
   2768 390.576115  192.168.43.61   192.168.43.72   HTTP     278 HTTP/1.1 200 OK  (text/html)
   2802 395.884156  192.168.43.72   192.168.43.61   HTTP      81 POST /espwebserver/postdemo.php HTTP/1.1  (application/x-www-form-urlencoded)
   2806 397.944000  192.168.43.61   192.168.43.72   HTTP     278 HTTP/1.1 200 OK  (text/html)
   2850 403.249736  192.168.43.72   192.168.43.61   HTTP      81 POST /espwebserver/postdemo.php HTTP/1.1  (application/x-www-form-urlencoded)
```

```
> Frame 1852: 278 bytes on wire (2224 bits), 278 bytes captured (2224 bits) on interface \Device\NPF_{39170176-55E5-4029-A44C-ED0D922E1363}, id 0
> Ethernet II, Src: HonHaiPr_7b:2f:e5 (d8:0f:99:7b:2f:e5), Dst: Espressi_32:e0:74 (58:bf:25:32:e0:74)
> Internet Protocol Version 4, Src: 192.168.43.61, Dst: 192.168.43.72
> Transmission Control Protocol, Src Port: 80, Dst Port: 50600, Seq: 1, Ack: 266, Len: 224
> Hypertext Transfer Protocol
> Line-based text data: text/html (1 lines)
```

```
0000  58 bf 25 32 e0 74 d8 0f  99 7b 2f e5 08 00 45 00   X·%2·t·· ·{/···E·
0010  01 08 0f 7f 40 00 80 06  12 9b c0 a8 2b 3d c0 a8   ····@··· ····+=··
0020  2b 48 00 50 c5 a8 68 c2  2e 97 2a 3b 20 f6 50 18   +H·P··h· .·*; ·P·
0030  fb 63 06 c1 00 00 48 54  54 50 2f 31 2e 31 20 32   ·c····HT TP/1.1 2
0040  30 30 20 4f 4b 0d 0a 44  61 74 65 3a 20 54 68 75   00 OK··D ate: Thu
0050  2c 20 31 32 20 4d 61 79  20 32 30 32 32 20 30 35   , 12 May  2022 05
0060  3a 32 37 3a 30 39 20 47  4d 54 0d 0a 53 65 72 76   :27:09 G MT··Serv
0070  65 72 3a 20 41 70 61 63  68 65 2f 32 2e 32 2e 31   er: Apac he/2.2.1
```

It is highly evident from the above screen shots that the packets are being transmitted towards the esp web server.