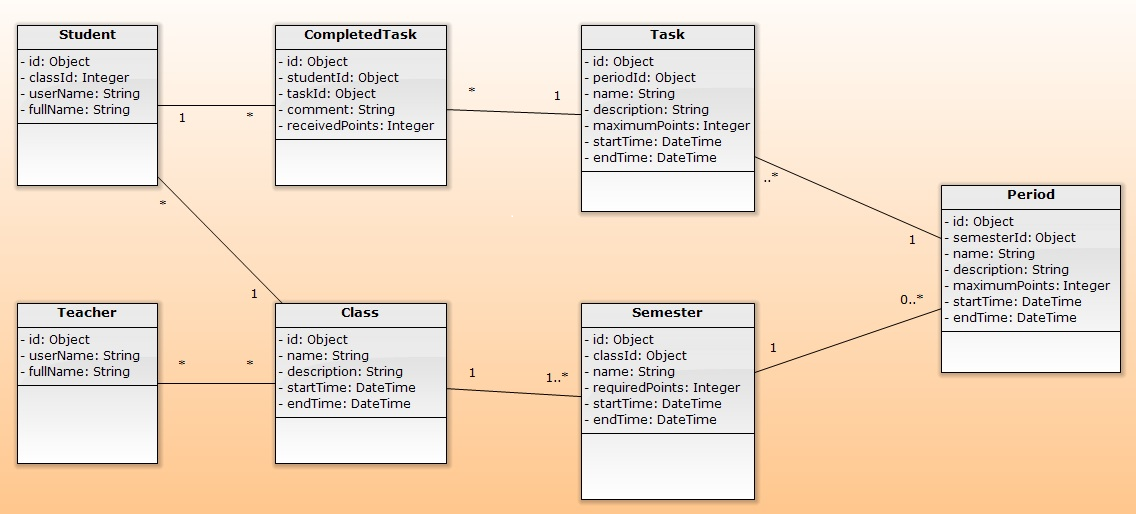
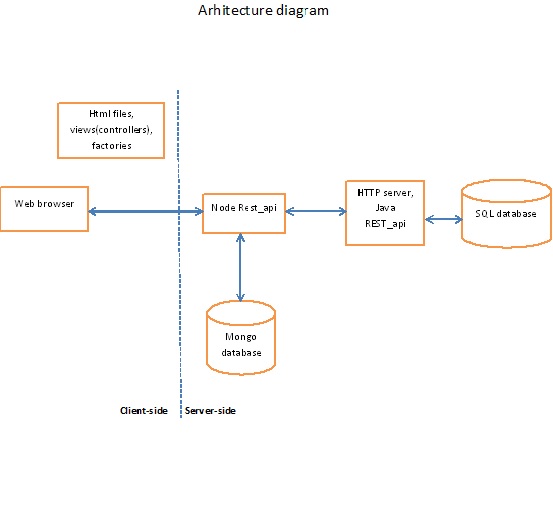
**OVERALL ARCHITECTURE DIAGRAM-----**

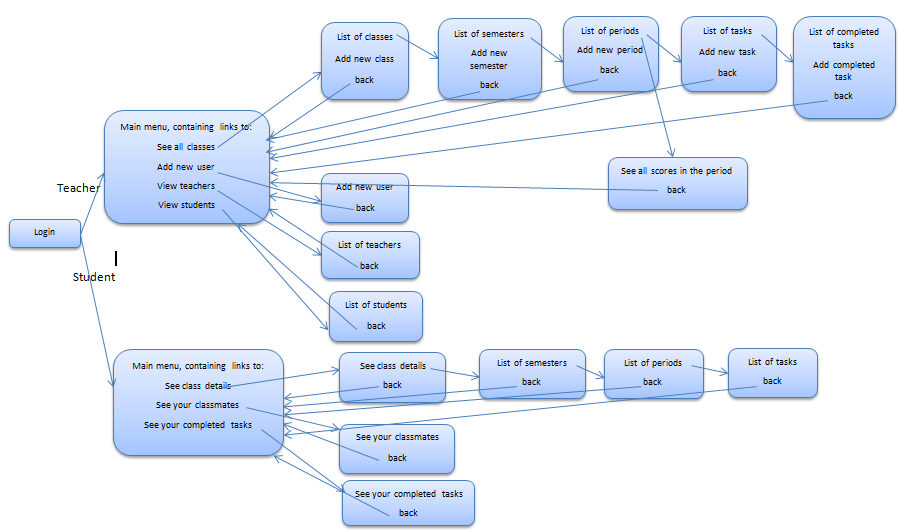
* DATABASE DIAGRAM

****

* ARHITECTURE DIAGRAM

****

**S**ITE MAP:



**DESCRIPTION OF REST SERVICES-----**

**Client to Server:**

The client- side is composed by the templates, the factories and the controllers. The client is connected to the server within the factories (the URI used in the factory has to be the same as in the REST-API JavaScript file). The controller calls the methods from the factories and using $scope they relate to the templates. We use the model-view-controller pattern. In our case, the model is the façade, the view is the template and the controller is the REST-API.

**Server to Server:**

**There are two occurrences of this**

* Login: The node server receives the relevant user id and password from the user interface using the ‘/authenticate’ route with a http ‘POST’ method. ‘GET’ is not used so that the user id and password which are sensitive information are never visible as parameters in the address field are thereby losing their secrecy.

The user id obtained is used to contact the JPA server with a http ‘GET’ request using the ‘/find’ route, which is present in the JPA server, whereby the matching user’s user id, password and role name, all stored in the JPA server are returned to the JPA server and then to the node server. If the particular user id is not stored in the JPA database, the server returns a “404” not found error. e

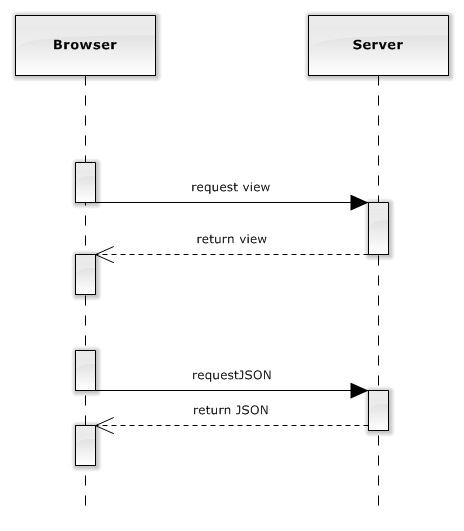
Once the node server has received the details, the password is extracted, compared with the password entered by the user and login is either initiated or disrupted also taking into account which views to display based on the role name obtained. If the request was not successful, the node server returns a “400” bad request message.

* Create new user: Here, from a form in the frontend a new user is created. Some of the users data is stored in the mongo db. However, information like the password is required to be saved in the JPA database. For this purpose a creation is made simultaneously to the mongo db and JPA database. Thus a call is needed between the node and JPA server to add values like the password to the JPA server (which is also encrypted, as mentioned earlier).

The route ‘/oneTeacher’ in the node server using an http ‘POST’ method is called to obtain the values entered in the form. The password is encrypted and the JPA server is contacted where the route ‘/create’ is invoked, which is also a ‘POST’ method, which takes the relevant values from the form, from the node server and creates a user in the JPA database. If unsuccessful the JPA server can handle both bad requests (“400”) and internal server errors (“500”). The node server can handle bad requests (“400”). All errors are returned to the user as error messages including the error code.

[NOTE: All server to server communication can be handled by the http methods]

**SEQUENCE DIAGRAM:**

****

**A STUDY OF THE TCP/IP STACK-----**

DESCRIPTION OF THREE WAY HANDSHAKE:

First the http request goes to a DNS, VIA A SWITCH AND ROUTER which translates a logical address to an IP address.

Then http which is packed into the tcp layer, allows a three way tcp handshake.

NOW Packets can be sent and received. Use the IP layer.

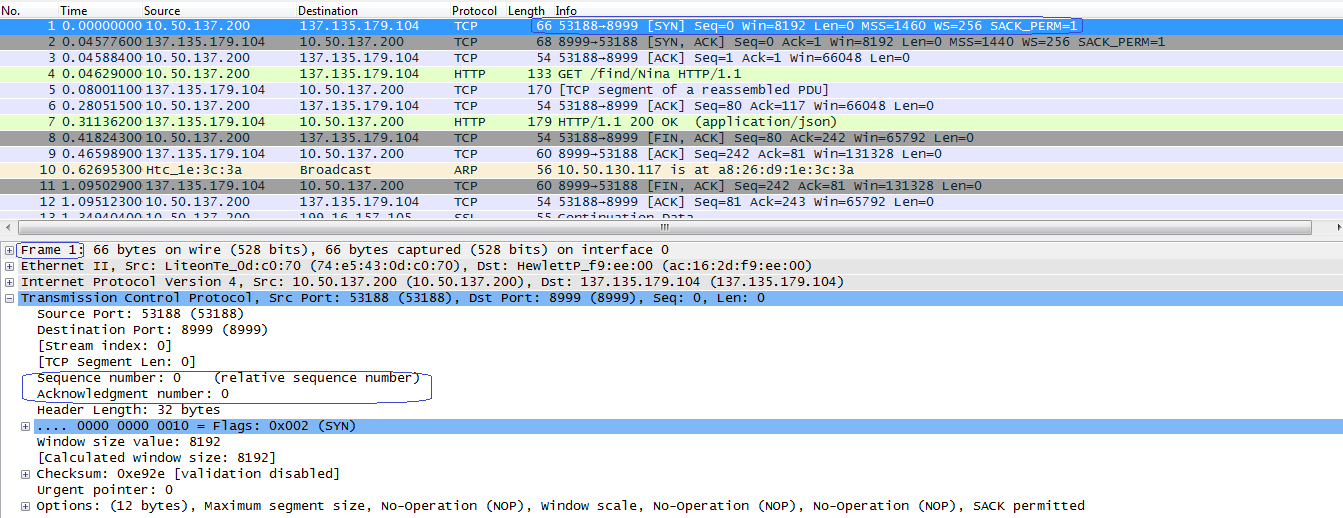
The node application was run on the localhost. However, the Java Server was hosted on Azure and communicating on port **8999**. Login for a user ‘**Nina**’ with password ‘12345’ was initiated.

Packet capture was done.

The following filter was applied to Wireshark: ‘ip.src== 127.0.0.1 or tcp.port == 8999’

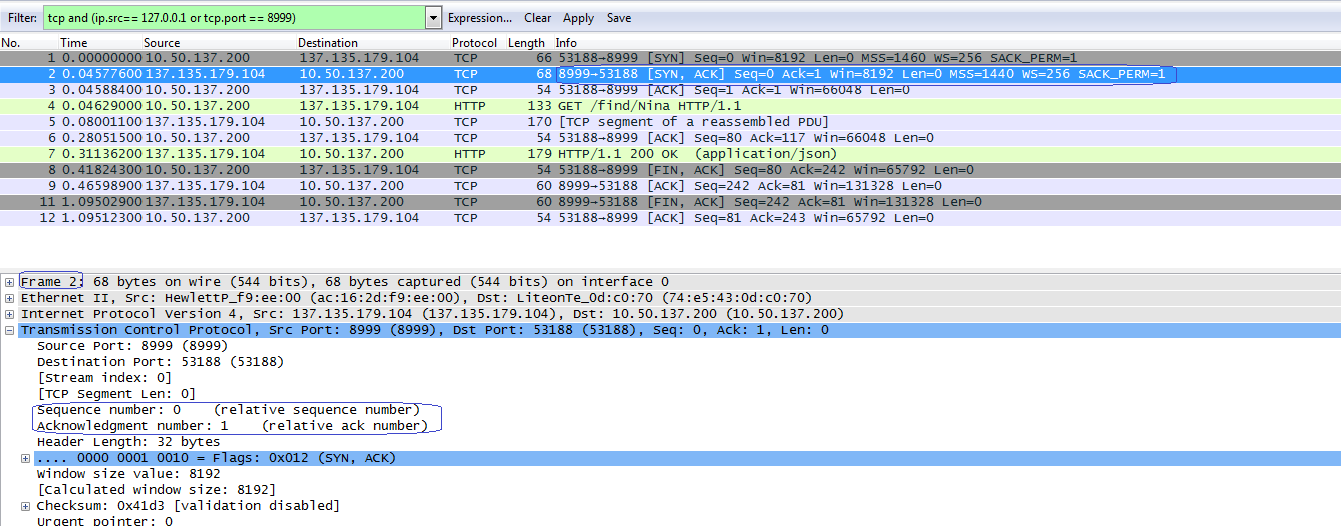
It would be the same if ‘ip.src==137.135.179.104’ were used, as this is the IP for our Java Server.

8999 as mentioned is the port where our Java Server communicates and login is through the Java Server.



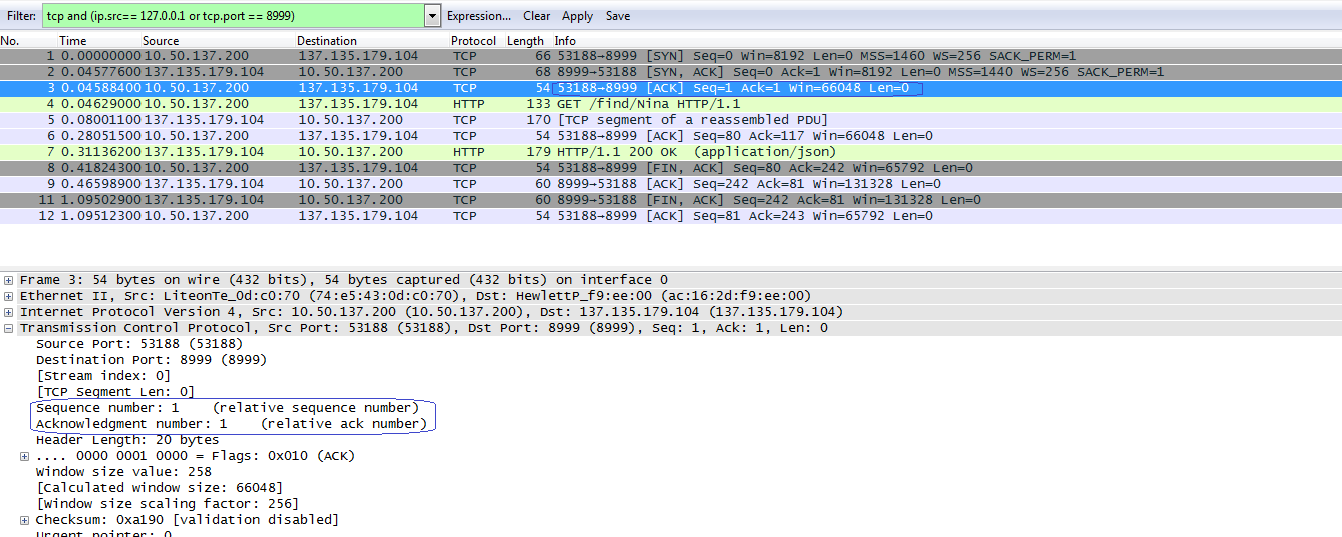
Step one: As seen above a ‘SYN’ or synchronize packet is sent from host A to host B. The sequence number and acknowledgement number are both zero. As the acknowledgement is zero, therefore the sequence number of the next packet from host B to host A must be zero (i.e. the ‘SYN/ACK’ packet). The source IP number is ’10.50.137.200’, source port: 53188. So our node app is using 53188 as the outbound port. Destination IP number ’137.135.179.104’, destination port 8999. The destination IP and port are exactly as for our Java Server.

Therefore it has been contacted.



At this stage, the ‘SYN’ packet from host A, our node app has been received by hostB, our Java Server, who sends its own ‘SYN’ packet to host A, along with an acknowledgement for receiving host A’s ‘SYN’ packet. Therefore, the so called ‘SYN/ACK’ packet.

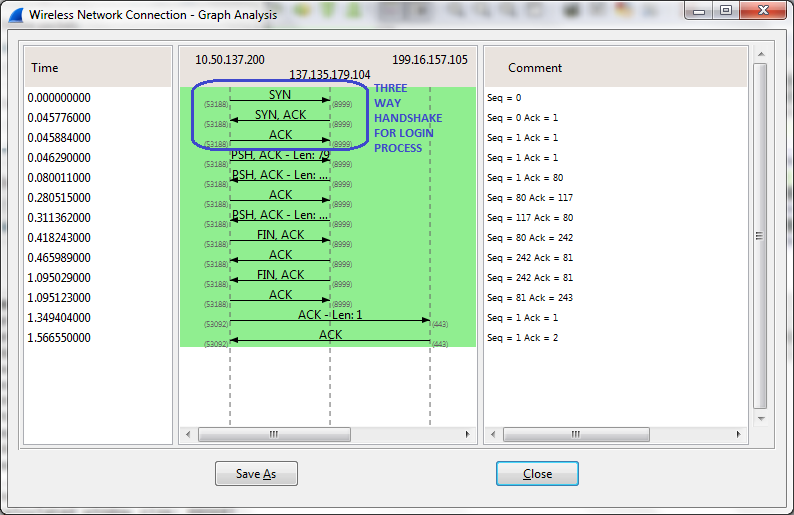
Step two: A ‘SYN/ACK’ packet is sent back from Host B to host A. The sequence number is zero, as expected, and the acknowledgement number is one. As the acknowledgement is one, therefore the sequence number of the next packet from host A to host B must be one (i.e. the ‘ACK’ packet). The source IP number is ’ 137.135.179.104’, source port: 8999. Therefore the packet emanates from our Java Server. Destination IP number ’ 10.50.137.200’, destination port 53188.



At this stage the ‘SYN/ACK’ packet is received by host A, our node app from host B, our Java Server. Finally, to complete the three-way handshake host A acknowledges the ‘SYN’ sent by host B, by sending an ‘ACK’ packet to host B, our Java server. Thereby both hosts have sent a ‘SYN’ packet to each other and both have acknowledged with an ‘ACK’ packet the receipt of their ‘SYN’ packet.

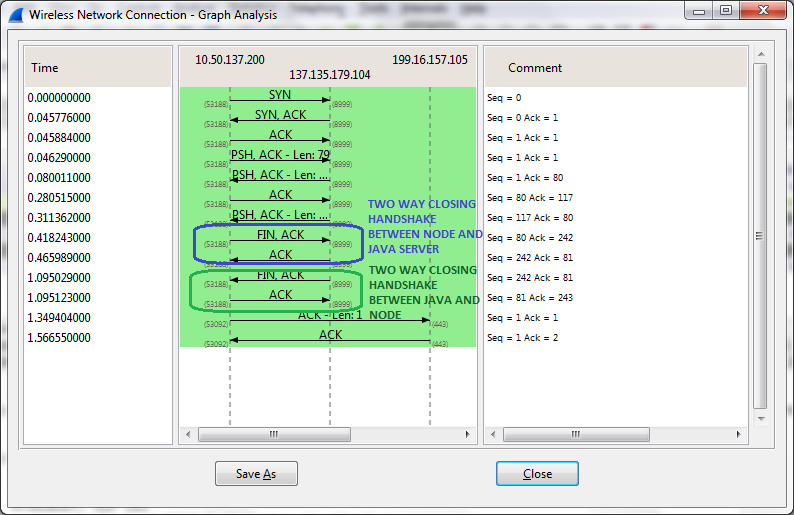
Step three: As seen in the screenshot above, an ‘ACK’ is received by host A. The sequence number as expected is one. Thereby the three-way handshake is complete and the IP layer can start the fully duplex transfer of packets, which is its payload from the TCP layer and above.

SUMMARY OF THE THREE WAY HANDSHAKE, HIGHLIGHTED IN BLUE:



**BIDIRECTIONAL TWO WAY CLOSING HANDSHAKES:**

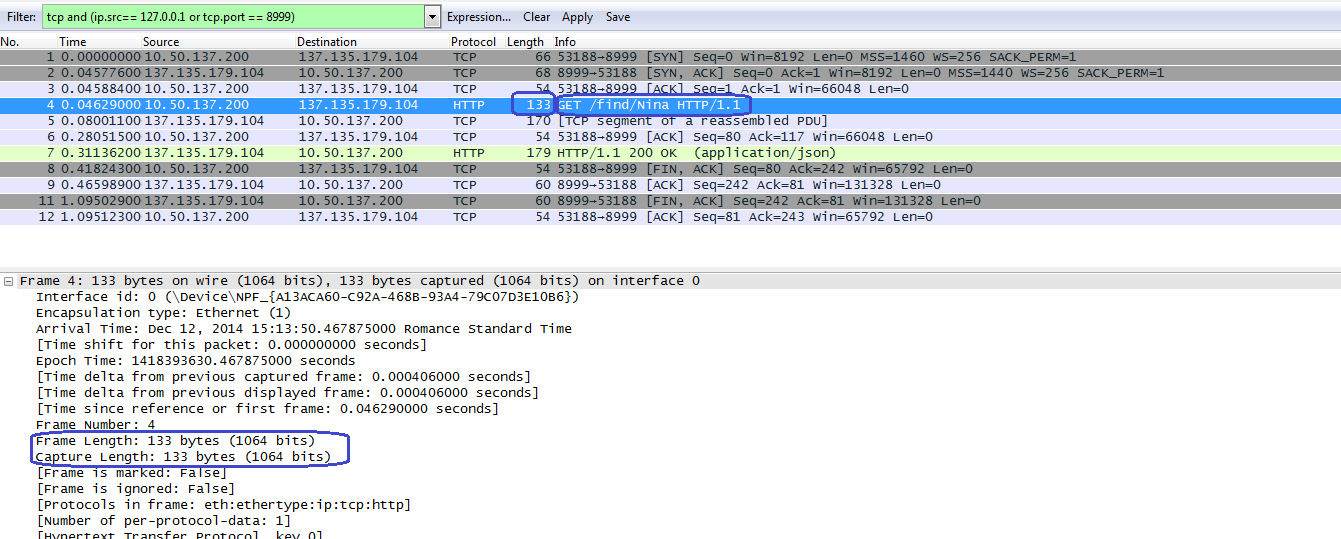
PLEASE NOTE: AFTER LOGIN IS SUCCESSFUL, THE FULLY DUPLEX CONNECTION IS CLOSED. EACH HOST SENDS A ‘FIN/ACK’ PACKET TO ITS OTHER SIDE. EACH SIDE RECEIVES AN ‘ACK’ PACKET FROM THE OTHER SIDE.



QUANTITY OF DATA TRANSFERRED:

Here we note that the total data transferred is not the same as what the browser receives.

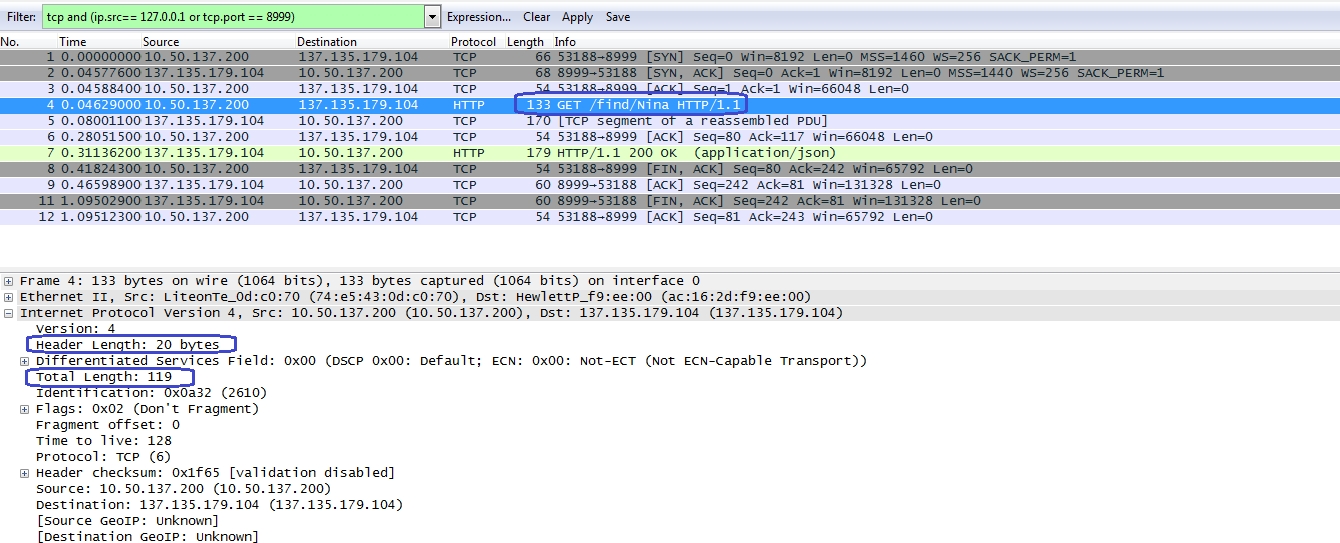
We calculate data received is length of packet minus length of header.



Here we look at **frame 04,** this is a http ‘GET’ to the Java server, using the Java Servers ‘/find/’ route, which takes the user name as a parameter. It can be seen in wireshark as ‘/find/**Nina’**

Length of packet is seen as: 133 bytes.

Furthermore, please see below:



The length of the header for the same packet (please note it is from the IP layer), is 20 bytes.

Therefore actual data received is 133-20=113 bytes. (!However, total length as seen in the image above is 119 bytes)

LAYERS OBSERVED:

The TCP/IP stack summarily consists of four layers.

**Application layer**: Uppermost layer.

Addresses-alphanumeric entries. Payload-html pages etc.

**TCP layer**: cannot transfer packets. It establishes the pathway for the transport of packets (three-way handshakes). Sees which server has to be used, eg chat, web etc, by identifying port numbers. Has both connection based(TCP) and connection less(UDP) protocols.

Addresses-Port numbers. Payload: Http, ftp, dns, dhcp etc.

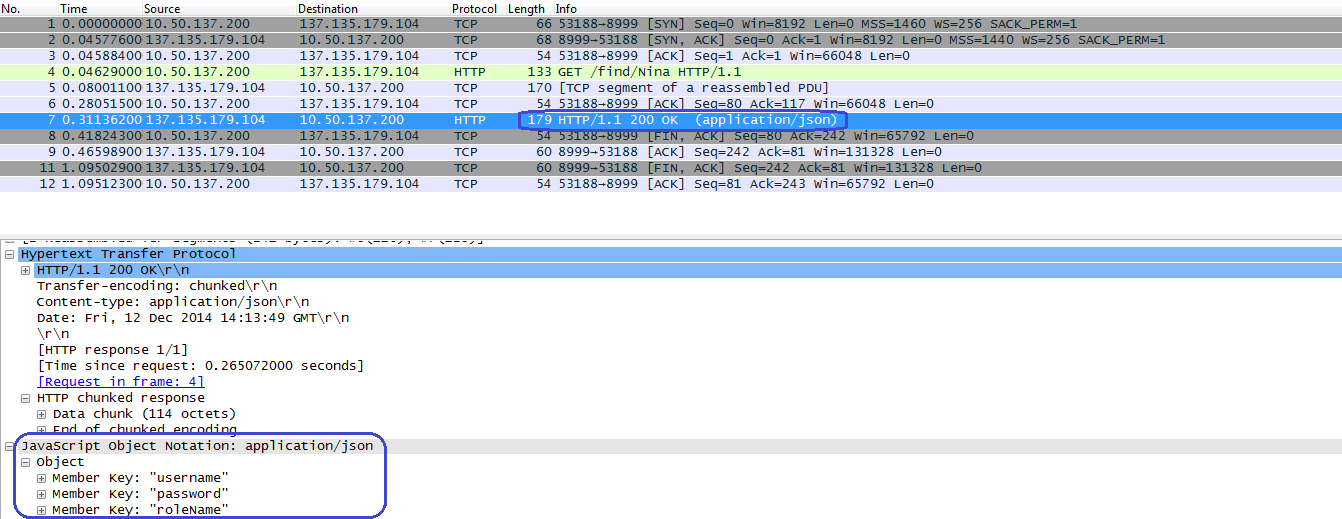
**IP layer**: now starts the actual transfer of packets through the ‘tube’ established by the TCP layer, using the IP addresses of the packets.

Payload-packets of data. Addresses-IP addresses.

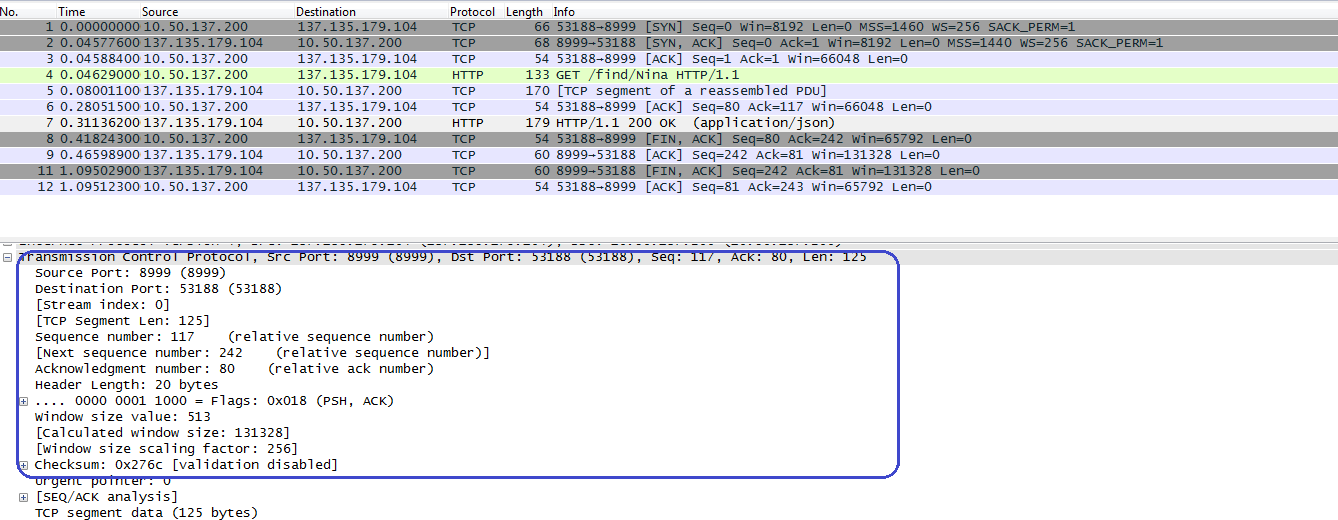
**Network/Link layer**: In order to reach their destination packets must flow through several networks. The routers along the route unpack the network packets and get the IP numbers. They use mac addresses to tell which network the packet has to go to next.

Payload-network packets. Addresses-Mac addresses.

ALL LAYERS ARE ANALYSED WITH THE OBSERVATION OF **FRAME 07,** (EXCEPT UDP)



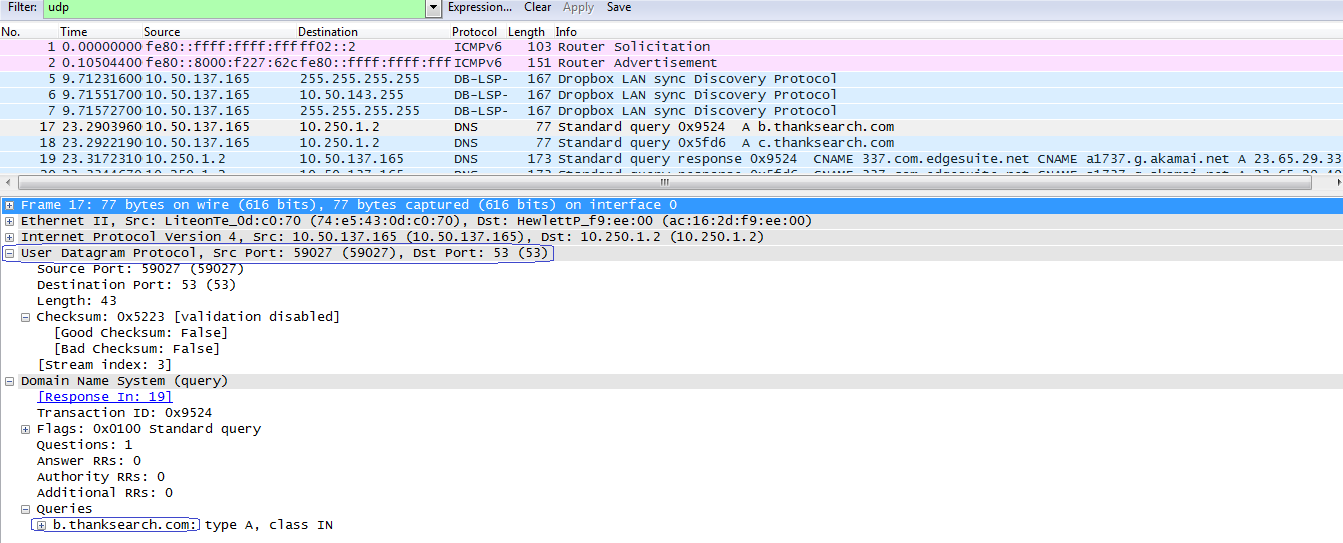
At the Application Layer level, we can see that the header is returned. The status code is ‘200’, OK. The Version of HTTP used is 1.1, Content type is ‘application/json’ and content is with the keys “username”, “password” and “roleName”, just as stored in our Oracle database hosted by the Java Server. The data was received in chunks of **114 octets.**



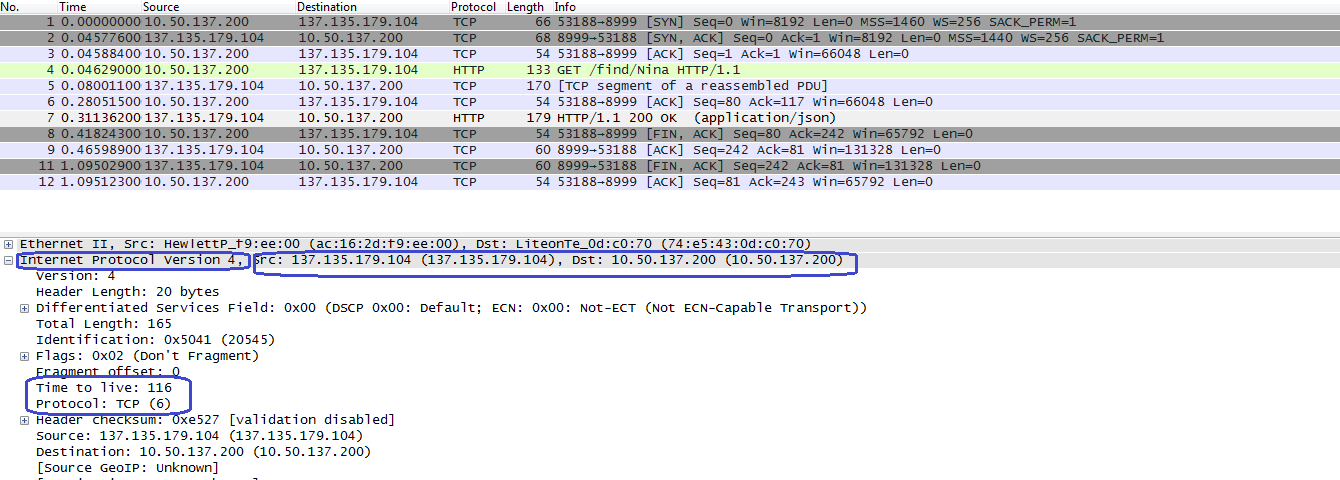
At the TCP level, the header contains the source and destination ports. It also contains the window size, which implements flow control by regulating the amount of data that can be transferred by the packet. The checksum is also mentioned. Checksums are tallied at sending and receiving side to ensure safety. This is all for the TCP packets using the TCP protocol which is connection based and starts with a link being established between both hosts using a three-way handshake.

However, at this layer communication can also occur with a connection-less protocol, which compromises safety but enhances speed, namely **User Datagram Protocol (UDP).**

DNS or domain name server calls for example are over UDP. This is a recursive search between servers to find the server which can provide the ip number for the alphanumeric web address typed in the address field of a browser by a user. Please see below (using packet capture from another hook up):



The UDP header is usually in 16 bit hexadecimal.it contains the source and destination port and the length of the packet.



The next layer is the IP layer. Here the actual transfer of packets occurs, using IP addresses. The packet capture screenshot above shows which version of IP numbers is being used. Here it is version 4. The protocol used in the layer above is mentioned. Here it is TCP. There is also information about the length of the header and checksums for safety among other things. There is mention of **time to live,**  for the packet, 116 milli seconds, to prevent denial of service attacks, all packets have a finite life time.

At the bottom is the network layer. Here the packets are passed through different networks using routers which identify the next router by finding it from a table of mac addresses.

SUMMARY OF THE PERSISTENCE STRATEGIES:

The entity class creates a blueprint for tables in a database which are created by the IDE using Oject Relational Mapping(ORM). In order to add tuples to the database, and entity manager factory is initiated. This in turn initiates an entity manager. The transactions are managed by the entity manager. A transaction is begun. Then it is committed. Then we persist an entry to the entity manager. Now the entry is ‘managed’. Finally when all the persisting is done, the entity manager is closed.