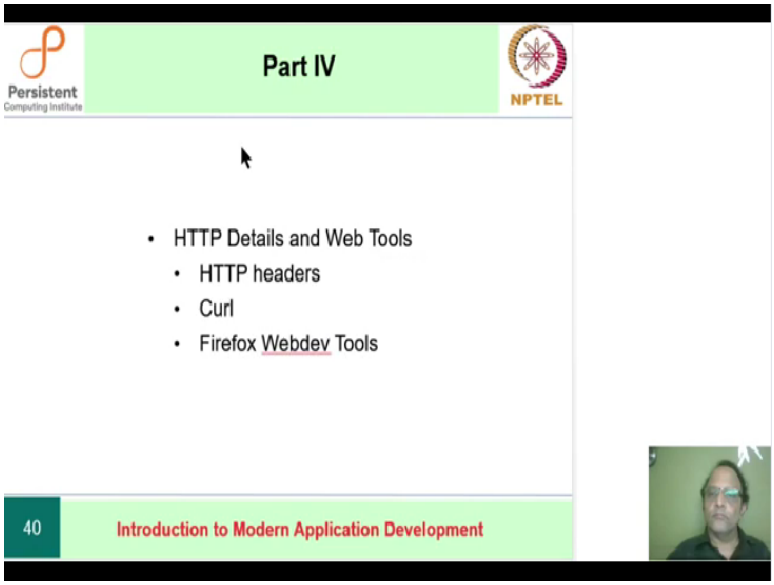
Introduction to Modern Application Development

Prof. Aamod Sane
FLAME University and Persistent Computing Institute
Abhijat Vichare
Persistent Computing Institute
Madhavan Mukund
Chennai Mathematical Institute

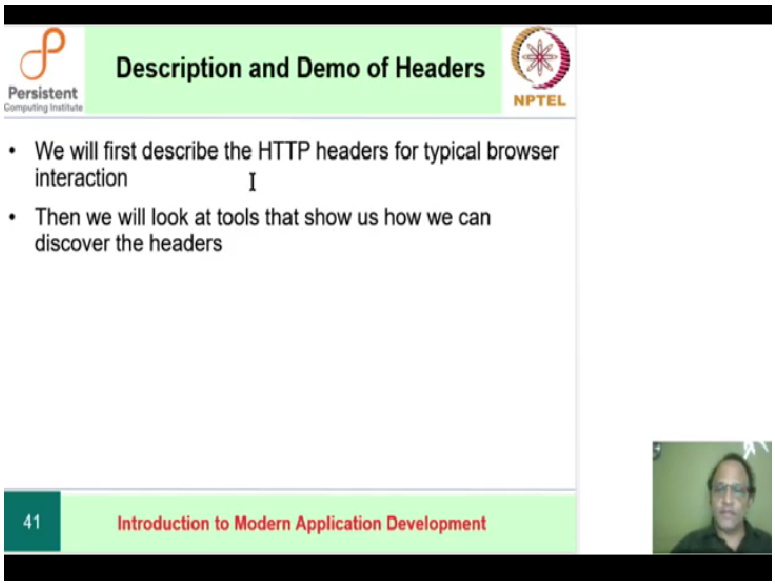
Lecture-16
Session 2-Part 3

(Refer Slide Time: 00:01)

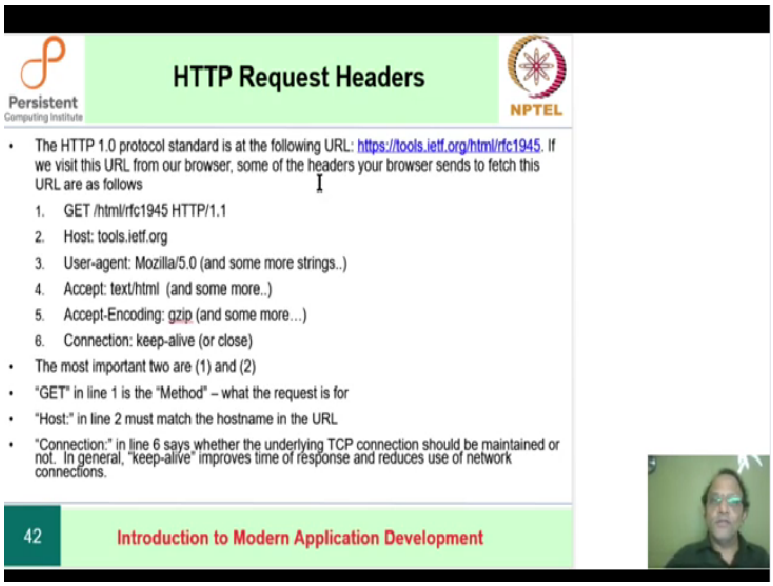


Now that we have seen how HTTP works at its root, we will now study HTTP in greater detail, we will first look at what popular or important headers there are that we should know about. And then we will look at two tools that help us understand or work with web systems much better. One of which is Curl and the second is the Firefox WebDev Tools.

(Refer Slide Time: 00:33)



Let us see what headers we see in a typical browser interaction. And then we will figure out how to actually see the headers in live interactions. **(Refer Slide Time: 00:19)**



Okay. So, the protocol standard for HTTP 1.0 is at the URL that I have shown here. If you visit this URL from our browser, here are some of the headers that the browser sends to fetch this URL. First it says GET html/rfc1945 HTTP/1.1. This is the opening header for the HTTP 1.1 protocol. The next thing is of course, the host which is *tools.ietf.org*. After that we have the user agent, user agent in this case is Mozilla/5.0, which and plus a long set of strings, some of which we have seen before.

And what they are exactly and ‘what they are for’ is not very important, at least not right now. Sometimes it is important, but for now we can just say that there is some way to identify which kind of browser is visiting largely, so that you can customize behavior based on browsers. There are some browsers where it makes sense. As an example, Google crawls you website by saying that the user agent is the Google crawler.

When, because you get a lot of traffic from Google, people will often allow the Google crawler to crawl their website much faster than a normal crawler, which is not owned by Google, let’s say, or even you trying to visit or download website using some tool or the other. So, that is one of the uses of things like User Agent. Another set of headers has to do with describing the capabilities of the browser.

So here, the accept header, what it says is, this browser knows how to show HTML. So, feel free to send HTML data directly, and the browser will be able to render it. So, accept headers tell us something about the client. So, if you see the classes of headers:
1. The things that clients want.
2. Who the client is. And next, what are the capabilities of the plan.
One particularly interesting header is called ‘accept encoding’.

Accept encoding says that you need not send exactly the bytes that are in the resource. Instead, in this particular case, it says that the browser understands gzip. So, you can use *gzip* to zip the contents and send them thereby saving traffic. And then it is the browser's job to get the data and unzip it before showing it to the user. This is one of those things that have become standard since at least 10- 15 years.

Lastly, we have connection and for connection it is either keep alive or it is close. There are other details which you can specify about the connection as well, but we can take a look at it later. Of all these headers the top two are indispensable because if you do not give those headers, then the server will reject your request, of the first line like as one said before, that get is sort of like a ‘Hello’ that we say when we meet a friend and we want to talk to them.

Technically, words like ‘GET’ in line-1 are called the ‘method’. The method is what the request is for; this request is for getting a resource. In the original HTTP 1.0 protocol, this was the only header there was because the browser was purely a documentation rendering tool, and which did not do anything much beyond showing documents, for example: forms were not there in the first version of HTTP that people were able to use.

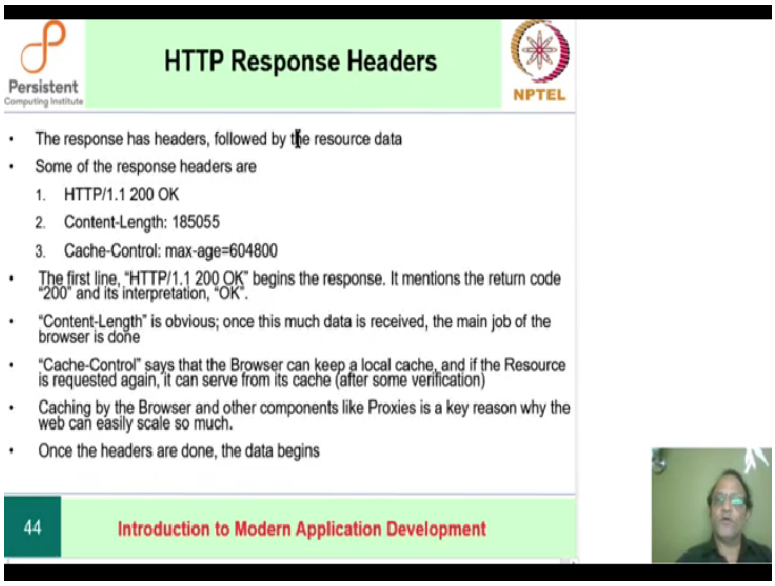
The second restriction, as we have seen before, is that the host line needs to match the hostname that the URL gets. There is a way to actually put the entire URL into the get header. But those kinds of things are used by intermediate components like: ‘proxies’ more than they are used by browsers or either the end line servers. And then as we saw, connection in line 6 says whether the underlying connection should be maintained.

Keep alive or persistent connections will improve the time of response and reduce the use of the number of connections that get made when a page is requested from a browser. The remaining headers tell the server something about the client as we have seen, what is important about this is to realize that, although, we are used to mostly seeing the web on powerful devices where the web can be rendered in nice graphics.

There are situations like some embedded software and so on, where the clients are really limited and so are the servers. And so making adjustments for your particular client is much more important. Another such example is the rare case of tools like Curl, or some of the so-called text mode browsers like w3m, some of which we will see later. Those are meant for limited environments, and so they will put bounds on what it is that they can accept.

So, much for the request headers; as soon as the server decides that it is going to complete the request, it gets the data and then the response. Just like requests are supposed to have certain kinds of headers so is the response.

(Refer Slide Time: 06:59)



The response headers are followed by the resource data. Here are some of the response headers:

- The first one as we have seen before says, HTTP/1.1 200 OK, which means everything is well and here is the resource you asked for.
- The next thing it says is: what is the content length. The idea is that after these many bytes are received, the server's job is done. And the client should feel free, for example, to initiate a connection drop from it is end, if that is what it wants to do.
- The third line is interesting. This is one of the key reasons why the web has scaled so much to literally billions of people over network speeds that have varied from very small to today's tremendous speeds. This is because caching is used everywhere on the web. And the request comes back with something that says how long can you cache. Now in this case, this particular document is considered permanent and unlikely to change again.

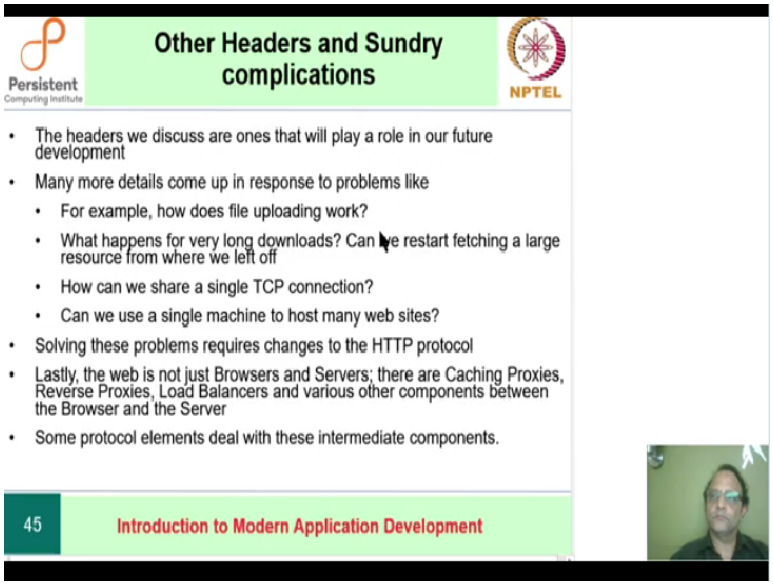
And therefore, the cache control max age is a huge number. This means that both the browser can keep a local cache. And if the resources requested again, and it is available in the cache, then the browser should feel free to show it. One of the things that browsers and cache managers generally have to do is to try to make sure whether their resource did not become stale while it was being cached.

Now it may be that there is a slow changing resource and so there are ways to check for freshness of the resource using methods like ‘head’ which tells you whether a resource has

changed or not. In addition to the browser, which keeps copies on the local machine, there is quite a big chain often of proxies which are held by, say, the institution you work at, your ISP, then there are proxies on the side of the server.

And then there are companies like CloudFlare and Akamai, who do the job of distributing data and keeping caches so that the actual network traffic received by the end data provider is as low as possible in order to save traffic related money. Once the list of headers is done, there is one more blank line. And after that the data begins.

(Refer Slide Time: 09:36)



The headers we have discussed are ones that will play a role in our future development. But, there are many other features of the HTTP protocol, and many different headers that are available that deal with issues like: how does file uploading work? So, a file may be uploaded in chunks and there has to be some structure in the protocol that allows you to upload fragments bit by bit or on the reverse where there is a very large resource, and it must be received one chunk at a time.

This is the question of what happens for very long downloads. Along with that, there are other issues. For example, can you start your download where you left-off. Some further concerns that today's browsers use is to share single TCP connections across multiple resources that may actually be served under different domain names, but by the same server, for example. Some

other things that the web does are to allow a single machine to host many sites and there are headers that have to deal with this kind of thing. So, these problems when they arise, solving them require changes to the HTTP protocol.
Finally, there are headers that deal with other components which are to browse and servers. These components include caching proxies, which cache the data on behalf of the client, reverse proxies which cache the data on behalf of the server, load balancers which are used to ensure that no machine gets overloaded.

And there are many such components that can arise in the giant network that is the web. Some of the protocol elements have to deal with these intermediary components. All in all, there are quite a few headers on the web. Now, that we have seen a list of headers and like I said, there are numerous headers. We will take a look at tools which tell you exactly what headers are being used.

(Video Starts: 11:50)

As we saw, while *socat* is fine for understanding the TCP machinery properly, when you actually want to work just with HTTP, there are better tools than low levels socat. The most important such tool is probably 'Curl'. Curl is a sort of a command line browser, you can give it URLs. And then it just gets the data and dumps it on the screen. So, let us try this. We have curl HTTP localhost, which, as you can see, dumps this kind of thing XAMPP webalizer, etc.

Going back just to verify, this is the list that Curl has dumped on the console, but normally, we do not want it. There are really two things you can do with this: one is, sometimes it is exactly what you want. You want the data from the console, and you want to save it as some file like 'save.txt' and in which case this is a simple way to just get the web page and save it without further ado. But, for our case we will be using curl to get much more information about what happens in interaction with a website.

So, let us take a look. So, we have HTTP slash localhost with a V. And now, this is not just the data anymore. So, for example, this XAMPP is over here, whereas, in the previous one without

the dash v after XAMPP, there was nothing else, after body HTML that was the last thing. Here, there is body HTML. And after that, there is some other material. This is the information that curl has added because we gave it - V.

Here is some more information, all this stuff. Whereas in the earlier case, it is just the document and nothing else. This is information is details about what is happening under the hood when call contacts the systems. Here, for example, we have the things marked with star(*) are interactions that have to do with the TCP level. Here, it says that a connection is being initiated, a connection is being completed.

And once the connection is complete transmission is going to go forward. At this point, we are ready to send our headers. And so, we send these headers out. After that, there is some more interaction having to do with TCP. And after that, you get headers back followed by the data. This is everything in some sense that is happening under the hood. But we do not want all that we really just want pure HTTP headers.

Headers, which begin with less than less than(<<) and headers which begin with greater than greater than(>>). So, in the usual UNIX style, we will filter out everything else, we will say, let's take this data, and join error output. So, how this data is coming in, by the way, is that the actual data that Curl is retrieving, the web page itself, It is printing on standard output, all these other things with stars and less than signs and whatnot.

Those are getting printed out on the standard error. So, what we want to do is to throw out standard output and get standard error in its place, which is done in unix with this rather peculiar incantation, that is not obvious but it is very useful and it is very common to use it. After this, we have *egrip* which is a way to grip out the data with some regular expression. And it says that you should throw away everything except or rather select everything except these guys.

So, now only the header lines are left. So, here we have GET HTTP 1.1, which we sent, the host is localhost, the user agent, as you can see is Curl with its version, and curl is willing to accept anything at all. So, there is no particular type of data that we need here. This part is the headers

that are sent by the server. So, here we have HHTP/1.1 200 OK, which says everything is fine. As far as the request is concerned, there is a date, just as the agent says what kind of agent I am.

The server says what kind of server it is. It then tells you the content length so that you can close the connection afterwards and it tells you the content type. Although in this case, we do not care because we are willing to accept anything and everything. Now, let's try to do something similar with you can try the *tools.ietf.org* site yourself. I will try a different site, which is *httpd.apache.org*.

Apache website itself, apache.org. Everything else remains the same. And so, let us see how we interact with httpd.apache.org. If you noticed, we sent our requests then there was a slight delay, probably some sort of network hiccup, and then we got these many headers. Our headers have not changed, they have remained the same headers that we sent to localhost, but the incoming headers, there are 2 new things there is LastModified and there is ETag, a bunch of other things like accept ranges, the one another thing called vary: accept encoding and so on.

So, there are, like I said before many different headers with many different purposes. And here is an example of some of them. Here are some more interesting things we can show with Curl. We know that our Apache is in fact running on two ports:

1. The usual HTTP port, and
2. The HTTPS port.

Let’s try and connect to our local HTTPS port. Now it complains, it says that, although this is an SSL server, it doesn’t have a certificate or rather it is a self-signed certificate, and therefore it is not going to accept it.

This has to do with security on the web, which we will discuss later. For now, with curl there is a way around it. You can tell curl with a dash tip to ignore this, and now it starts behaving exactly like our normal HTTPS. Let’s see what happens if we do this with a browser, if you try to go to HTTPS with a browser.

Then what we see is that the browser says that this connection is not secure. But this browser is probably being clever. It understands that localhost is the local machine. And so it does not really matter. It says that there is an implicit exception to this. If we remove this exception, we get this kind of message. It says potential security risk. But in this case, it does not really matter. So, we can say, accept the risk and that is precisely the behavior of minus(-) k as far as curl is concerned.

Here's something else interesting that we can find out from curl, let us try the following. We will say curl *http tools.ietf.org html rfc1945* and let us see what happens. So, this time we got something else. It says that this document has moved, and where has it moved, it has moved to HTTPS. So, what has happened is that *ietf* is only serving these documents from HTTPS server. So, these are the kinds of things that are easy to find out with curl.

Besides curl, the next most important set of tools that are very useful in web development or the browser dev tools. On Firefox, the keystroke control+shift+I shows you these tools. The tools are also available here as web developer tools. Okay, now let's see what all is available in the tools. We have the inspector, which is useful for looking at HTML, the console and a debugger, which have to do with JavaScript.

The style editor, which is for CSS, and a number of other things about involving various browser facilities that we will study later. For now, our main interest is to look at the network tab. For this, let us do something interesting. We will start our old socat server and first visit that server to see what it is that we get from the browser. Here we go. I have already visited it once. So, what I am going to do is to shift reload this page.

When you shift reload, the browser ignores all caching and behaves as if you are visiting the page for the first time, so let us do it and see what happens. Okay, here is what we see. We see as expected the request to get slash, which is to be expected because what we have here is that the URL does not have any request URL part. Here is something unusual, where the browser requests for a fav icon.

A fav icon is this icon like the W for Wikipedia, the welcome here for the apache project, and various other places where the fav icon shows up. In our case, since we are visiting the socat server, we do not return any fav icon, but the browser makes a request anyway, so as to distinguish the tab. In case such a fav icon is available. First let us see what has happened on the server side. As expected, on the server side, we have received two GET requests.

One for slash and one for fav icon, and as we saw the last time we ran the server, we have all these details available from the server, where it tells us about things like the caching parameters, and the user agent and so on and so forth. This is printed by our server since that is all it does. Let us take a look at what the browser tools show us. So, if you click here, you will start seeing all the details of the browser.

So, here what we have is request headers. And there are no return headers or response headers in this case, that is because our fake server does not actually return any headers except the acknowledgement that everything was okay. Let us see what else is there. We do not have any cookies, we do not have any parameters. And the response is just the string, which is shown here as we might expect.

Quite a few interesting facilities are available in the browser tools. For example, here is a way to filter out headers for I just typed in cache. So, what we see is that: two cache control related headers, another useful facility is to see what the raw headers look like. So that, all the spelling and all these issues become clear. In case there were some problems that at some point, we also have these kinds of things.

For example, the request URL is this and the URI is empty, because at the end of the URL is all we have. Then there is the request method GET. And finally, the address which was fetched which in this case happens to be the localhost. Then, we have favicon.ico which lets us turn the raw headers of so we can see what it has asked, we already seen it on the server side here the result of this is an empty icon, which is not returned by us, but it is synthesized by the browser.

So, that it does not have to create a special case and then include no icon, as if, without creating the special case, thinking about whether or not an icon was available. But this is entirely synthesized by the browser in this case. What else is there? We have in these tools, the inspector which is to look at HTML, the console and the debugger for JavaScript, the style editor for CSS, and a bunch of other things that we will study as we develop web applications further.

Next, hide request details and see what else was there. So, in this case, we are told that the type which came back was plaintext and 80 bytes were transferred. The response itself was 61 bytes. And this is how much time it took. Notice that fav icon which we did not return, took some more time while the browser decided that there was no icon and that it should actually synthesize. Here is a somewhat of a summary:
There are two requests how much data, what is the total time. These things like DOM content loaded and all we will study in time. Okay, so that is about one use of the browser tools. Now let us try something else. This time we will simply reload the current page rather than doing shift reload and let us see what happens. So, this time, it is known to the browser that there is no fav icon. So, no fav icon request gets made.

However, we do ask again for the main request In that case, it is this date, which will be different. Other than that, we would expect everything else to be the same. Here is a couple other things. This tab is showing us absolutely everything. But, you can ask for requests for individual pieces. And there is not anything being sent that is in these categories. Because the only thing we had sent was plain text for which there is no special category except other, which is a catch all for everything else.

So, that's what we have from the browser devtools for this simple site. Let us study another case. This is where we were running the apache from our xampp package. And this is what we expect to see from just going to localhost. As before, what I will do is I will start the web developer tools here and then shift reload, okay. Now, we have something more interesting. Besides slash there are all these other things blank gif, text gif, etc, which are the icons that are used to get the appearance of a directory from the localhost.

This is done by Apache as part of it is of standard approach of going and synthesizing this kind of response when there is no particular index or index.php or any such file present. And this time the type is HTML, these are gifs, this is an icon. So, accordingly we will see things over here. Now, the only HTML request is for slash, we have a bunch of image related requests.

And there is no particular CSS JS or any other kind of request here, all we have is every single request that we are able to see. Let's look at what the request and response headers are. This time both the request and response headers are available. The request headers are below. This is the entire set of requests just like last time, for our simple server. But this time, we have response headers.

In particular, we have connection keep alive, we have the length of the content. We have the content type, we have the date, we have keep-alive and we are told the identity of the server. Let's see if there is something else, there is no cookies, of course, no parameters. And the response data is this thing where the payload is shown to be this piece of HTML. So, a table is created. And all these kinds of things are put together, so that we have some response to see.

Here is a list of timings, what happened during this request, was it blocked was there were there any issues talking to the DNS, and there was some wait while the server had taken time to respond. Stacktrace here tells us in what order these things happen, these kinds of details, they depend on the browser architecture as well, so we would not look too much into that.

Now, let us shut off request details again. And we will move on to some other sites. Alright. Our next site will be the site for the apache server project. Again, start these tools and do shift reload. So, let me hit it again and get fresh reserves alright. So, this time, as you can see, there are quite a few more requests, most of which are images. There are stylesheets and see what this guy shows.

So, here we have two stylesheet requests. Is there JavaScript? Yes, there is our JavaScript. Then were there any other kinds of things, was there media looks like no media. And finally, when we go to Network Tab in the devtools, we shall see all the requests. Next, look at what the headers

look like using the button in the Network Tab. We have request headers since they are originated by us, they do not change, but response headers this time, have a lot more detail. In particular, notice these new things the ETag, LastModified and so on and so forth, which we have seen with curl. And we see this again.

In the cookies section, we find that there are no cookies. We know what the response is like. So, we do not want to look at that again alright, let us close the request details and see what else we get. So, there were a total of 15 requests. This much data, it took about 4 seconds for everything to come in. And the basic page was rendered in after about one and a half second. So, this is an easy way to understand how well your site is performing.

Now, let's go on to another major site, which is Wikipedia. On loading the website, we observe that it takes quite a bit of time to load everything in this page. And to see the effects of caching, we will reload without using shift. What do we see here? Looks like not much difference. Not much got cached. So, it is showing everything but it is showing caches and that is interesting.

I wonder why they are cluttering to stuff up with caches. Okay, I need to spend a lot more time to understand this in detail. So, for now, I will just let this go. And we will just know that this thing has much of a difference. I am going to hit it again and see what happens. This order, it should have done much better. Okay, let us see if Wikipedia does any better if we try this thing out.

So, we have this page here, we will hit shift reload, so as to get everything. What do we see Wikipedia is actually quite a bit faster. Not surprising because Wikipedia is a very popular site. And they are heavily cached everywhere. And they must be spending some effort on being easily accessible from all over the world. Now this time, we have quite a few more interesting looking URLs. So, let us see what all we get.

Here we have the basic site and request headers we have seen but check these response headers out. So, they have age, they have something called back-end timing. They have various things about caching. There is this thing, which I have forgotten what p3p is for, it is a very large list of

headers and unless you happen to deal with one of them, it is a long time after you need to notice all the details.

But here is something interesting. In addition, to the normal headers, there are a lot of x dash headers, these x dash(x-) headers are not standard headers, they are done by Wikipedia for its own use. So, for example, x varnish has to do with a cache called varnish, which probably Wikipedia uses. And therefore, it is telling something useful to the varnish cache, which once you study varnish, you will know what that is about. Okay, what else do we have here.

So, here we have a stylesheet, but the stylesheet URL, this time has parameters. So, this parameter step tells us that there is a language, there are a number of modules, that there are styles, there is something called '**skin**'.

What else? So, as you can see, these tools tell us a lot of interesting things about what is going on behind our sites. And once you are familiar with the tool, you can go to popular sites and see what it is that the popular sites actually load.

Most of you if you have not seen these tools before, you will be shocked because there are a lot more requests that are going on, then you might imagine, and nonetheless, the performance of the overall system is pretty good. Okay, now let us see what happens if we simply reload, hoping for caching to have some effect. Aha, this time DOM content loaded went down to 325 milliseconds as compared to one second.

Because almost every single thing is cached, including the page itself. Now that is really cool. Okay, so this is a great demonstration of what caching does. And indeed, we see here that 0 bytes have been transferred. Good. So, the only time spent was in doing some amount of rendering on the page. Alright. Let's take a look at yet another page, which is this thing. And this time we are going to do a page with a query string.

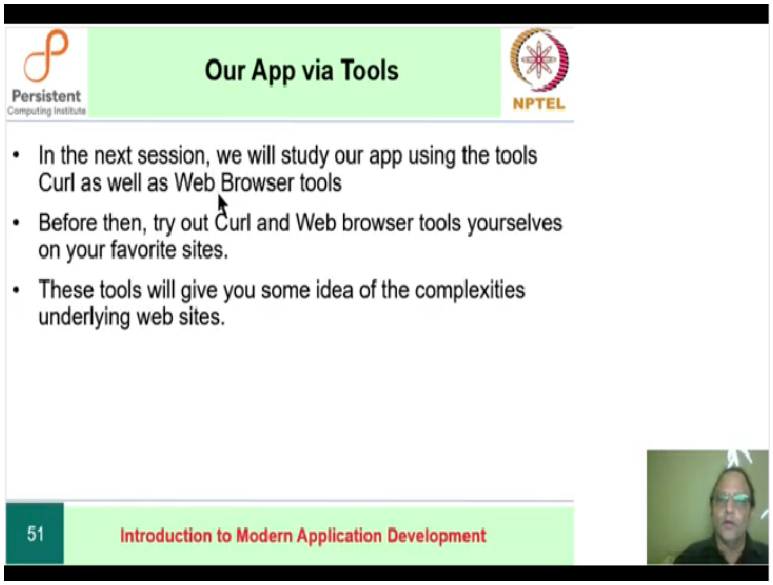
So, we give this search query. And let us see so Wikipedia gave us this page "query string url". Alright. All the other elements have been cached from the page that we had just loaded. And

here is what we have from the search query itself. More parameters, we have headers. Is there anything new? Looks like something called server timing. I did not notice it before, maybe it is there as well. okay.

I guess it was basically waiting for the search alerts. So, as you can see, this is a rather powerful tool, which tells us all sorts of neat things about what a web page is doing.

(Video Ends: 38:53)

(Refer Slide Time: 38:54)



We are done with most of this session. So, next time, we will study our app using the tools that we have seen right now. And before, then you should try our Curl and web browser tools yourselves on whatever programs you write, plus any of the favorite sites that you normally use. And this will give you some idea of the complexities that are behind most websites. Now let us summarize what we have done in this session.

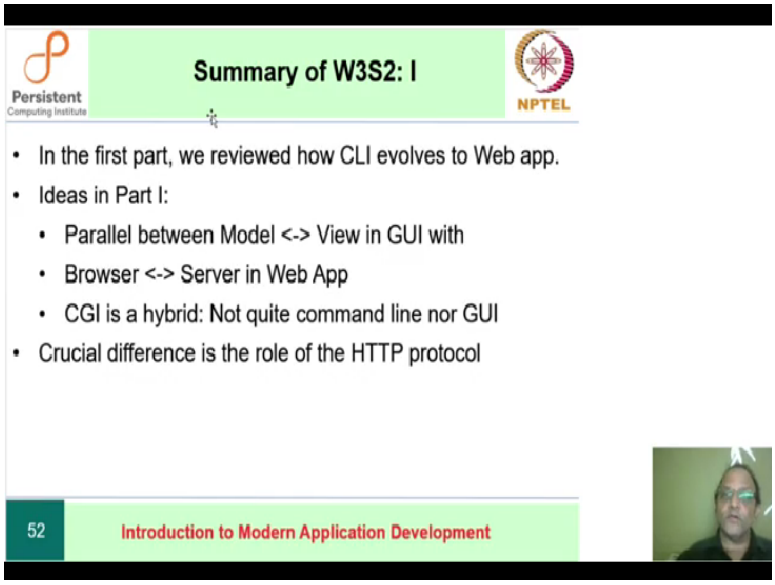
In the first part, we reviewed how a CLI evolves to a web app. The ideas in part 1 were the following

- First, there is a parallel between model and view in GUI with browser and server in web app.

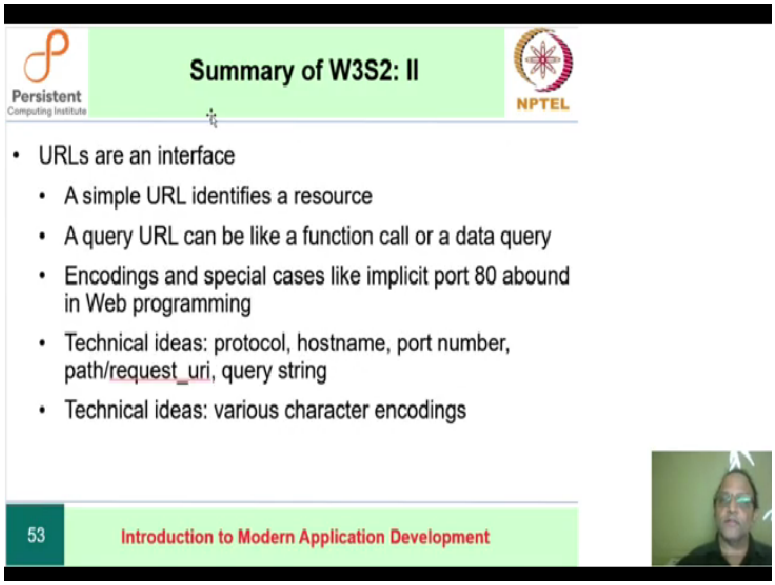
- Then, we have the idea that CGI is a hybrid. It is not quite command line nor GUI, but has elements of both. The crucial difference between a GUI and a web app is the role of the HTTP protocol.

I notice that I wrote model view and browser server here, I guess model view ought to be reversed. Next okay, let us move on to the next alright.

(Refer Slide Time: 40:08)



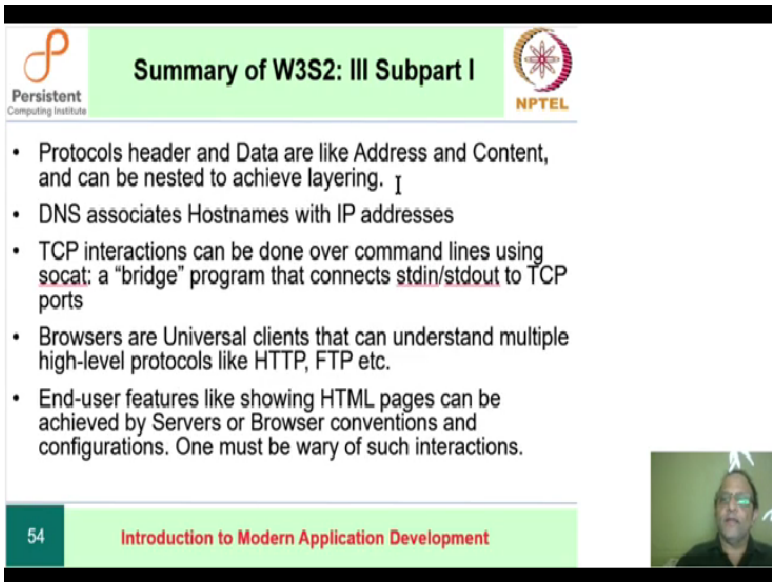
(Refer Slide Time: 40:09)



Here is a summary of part 2. Part 2 is main lesson is that URLs are kind of interface. A simple URL identifies a resource. A query URL can be like a function call or a data query. There are some odd things in URLs such as encoding, plus special cases like implicit port 0, and so on, which were done for one or another sort of convenience might have become part of the ecosystem.

A few technical ideas are that there are explicit parts of a URL with the names protocol, hostname port number, path or request URI and the query string. And another such technical idea is that various character encodings exist for historical as well as currently valid reasons for describing what a URL really is.

(Refer Slide Time: 41:11)



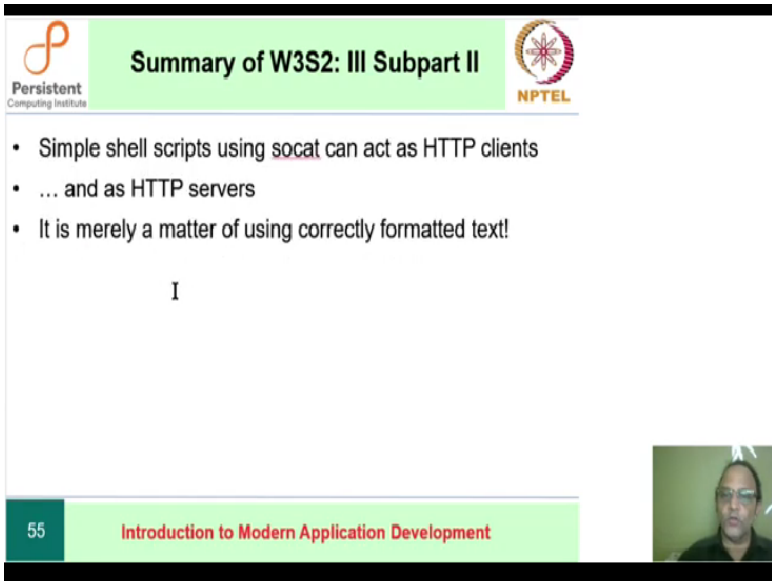
Next one in part 3 so part 1, we saw that protocol, header and data are like address and content, and can be nested to achieve the layering of protocols. We also saw that DNS associates host names with IP addresses. That's its main job. TCP now can be bridged with command lines using socat. It connects the standard in standard out to TCP ports and makes it easy to write TCP programs using standard facilities like shell scripts.

Another lesson of part 1 is that browsers are universal clients that can understand multiple high level protocols like HTTP, FTP, etc. Finally, end user features like showing HTML pages can be

achieved in many different ways. By using server or browser conventions and configurations, you have to be vary about such interactions. And you should know why it is that your program behaves the way it does.

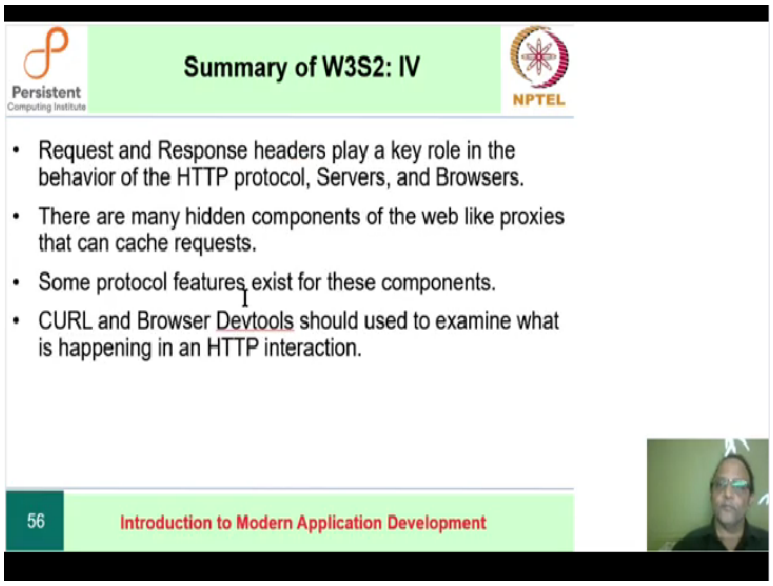
Since there is so much variety in end user behavior, we find that you must spend a great deal of time on occasional problems that arise for one particular set of users but not for others. Nonetheless, things used to be much worse when there was a lot more diversity. But by now, most people use common and browsers which are updated from time to time and so web programming has become quite a bit easier nowadays.

(Refer Slide Time: 42:58)



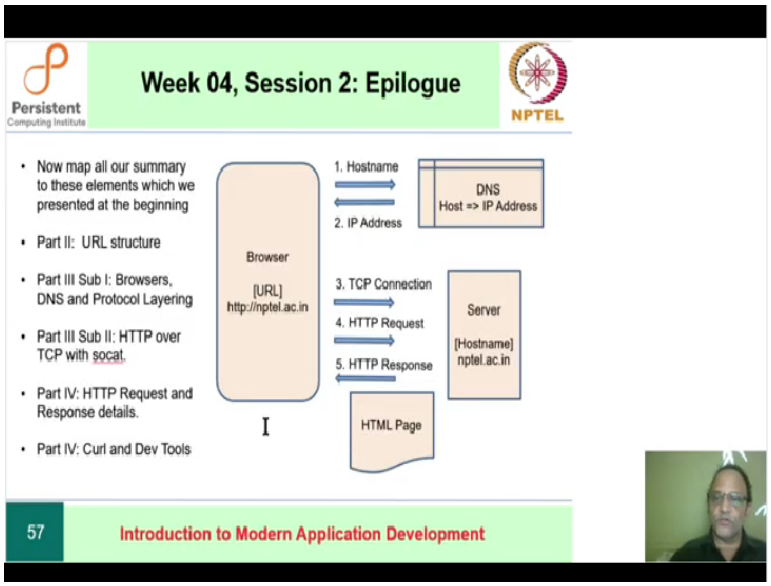
In part 2, what we saw is that simple shell scripts using *Socat* can act as HTTP clients and servers. And the beauty of HTTP is that it is merely a matter of using correctly formatted text.

(Refer Slide Time: 43:12)



In part 4, we saw that request and response headers play a key role in the behavior of the HTTP protocol in the behavior of servers and browsers. There are besides servers and browsers, many hidden components of the web like proxies that can cache requests. Some of the features of protocols such as particular headers exist for these components only. And the last lesson is that curl and browser dev tools should be used to examine what is happening in an HTTP interaction. Okay, we began this discussion with a diagram that summarizes the situation.

(Refer Slide Time: 43:53)



So, let us end with the same diagram and see how the ideas that we studied map into this world. So, part 2 is about URL structure, which is shown over here. Part 3 is about DNS and about protocol layering, the connection between line 3 and line 4, which we saw how nested packets

achieve this sort of layer protocols. Then in the sub part 2 of part 3, we saw how to get HTTP to work over TCP and indeed the details of how line 3 and 4 are related.

Lastly, part 4 is about the details of HTTP requests and response and also about curl and other developer tools.

(Refer Slide Time: 44:48)

Okay, this has been a long session with many intricate new details. In our assignments, we will go over these to help reinforce them for you. You will find that understanding issues in web app development we will come down to having a clear picture of the ideas in this session. And gradually the ideas will sink in with practice. See you next time.