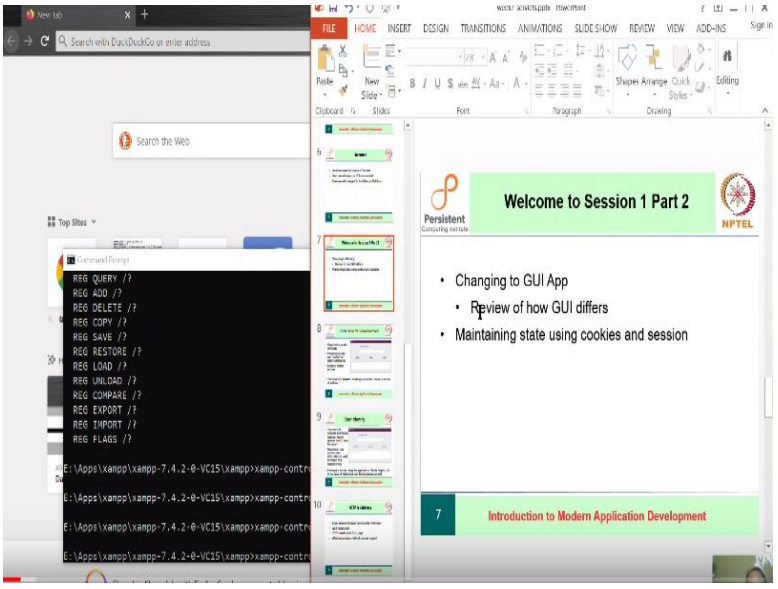
Modern Application Development
Mr. Aamod Sane
Department of FLAME University and Persistent Computing Institute
Indian Institute of Technology- Madras

Lecture - 23
Introduction to Modern Application Development

Welcome to session 1, part 2 of Introduction to Modern Application Development. We have now begun to change our CGI wrapper app into a GUI app, and as a part of that we saw last time that we studied :.

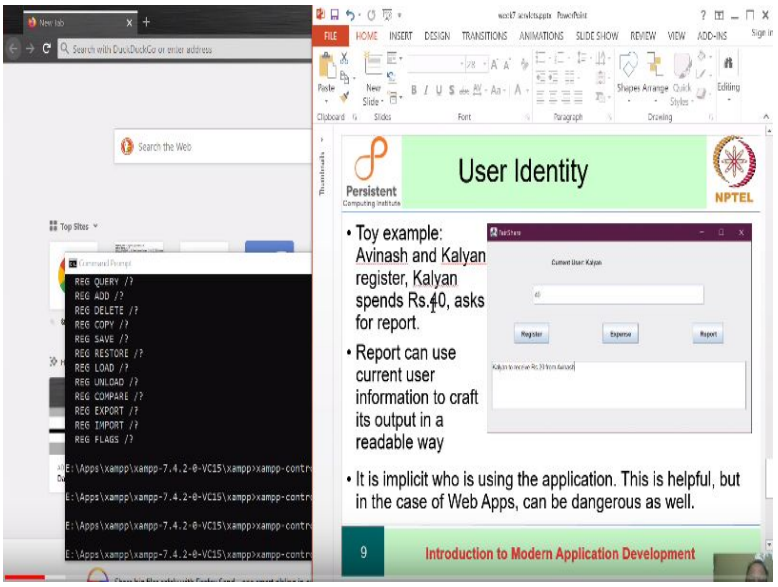
1. How the architecture changed across the three systems ; CLI, Cgi-bin, Servlets.
2. The internals of servlet
3. The demos that Tomcat's built-in servlets give us.

(Refer Slide Time: 00:56)



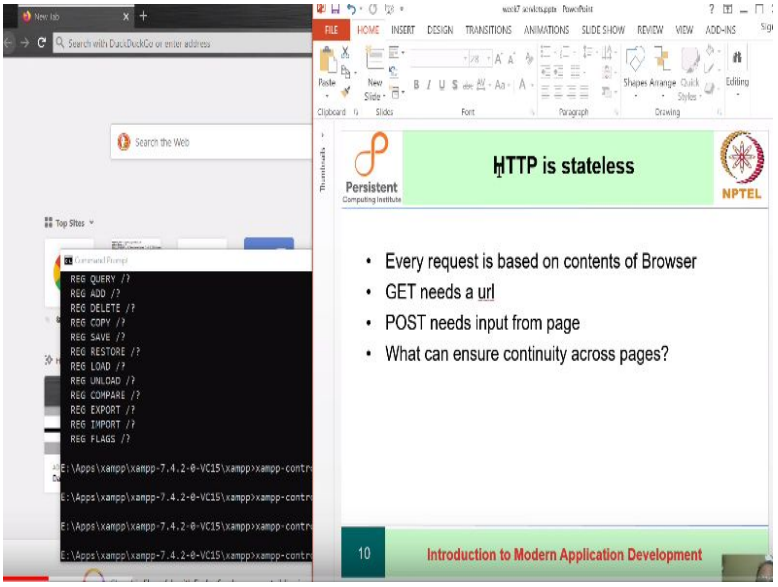
Now, we take a look at the next step of how to make a GUI app out of our CGI app. Our GUI app has a register button which acts somewhat like login and it then displays the name of the current user and it provides identity and context for other commands and enables further actions.

(Refer Slide Time: 01:47)



This notion of a single application, which shows you a user interface and you work in its current identity, is exactly like how we login to our systems. So on a per application basis, one can have a similar idea, which is known as sessions. So our toy example was that if the current user is known, then report can use that information directly to craft the output in a readable way and that it is implicit as to who is using the application.

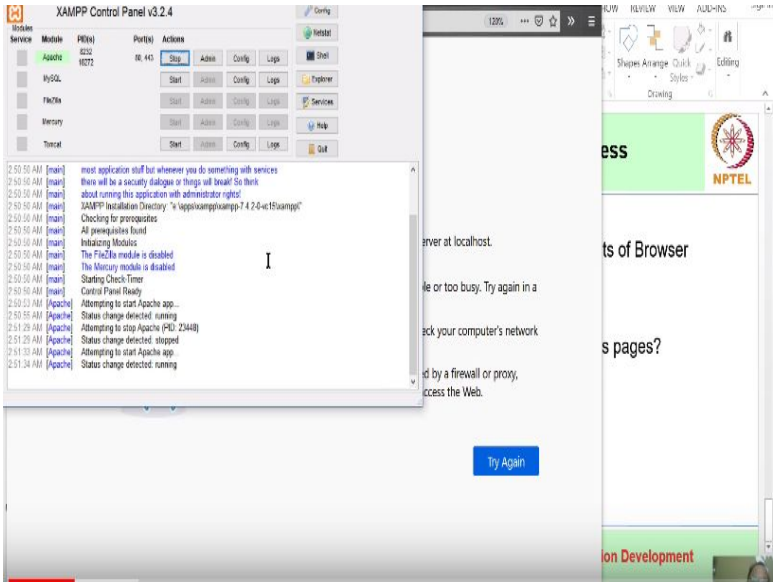
(Refer Slide Time: 02:37)



This nature of GUI apps, is what we intend to replicate this time. The difficulty with that, is **HTTP protocol is stateless**. HTTP is the protocol that connects the browser to the server. To demonstrate what stateless means, do the following:

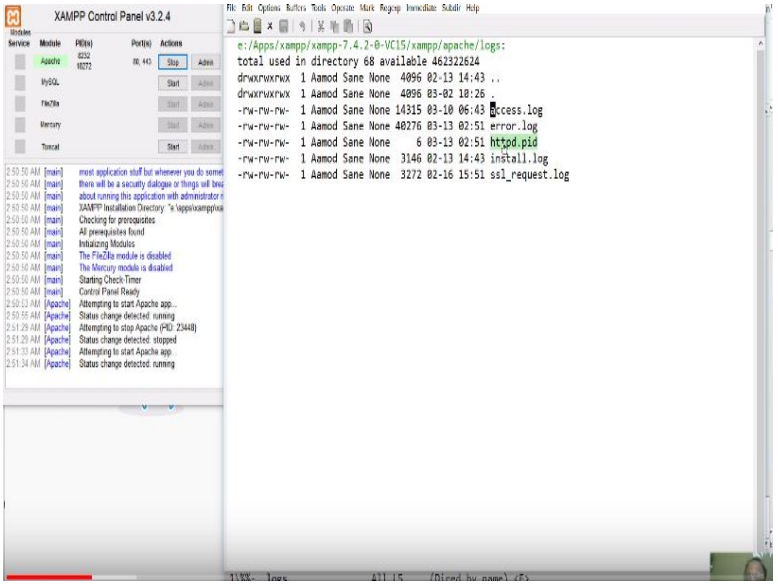
1. Start the apache system

(Refer Slide Time: 03:06)



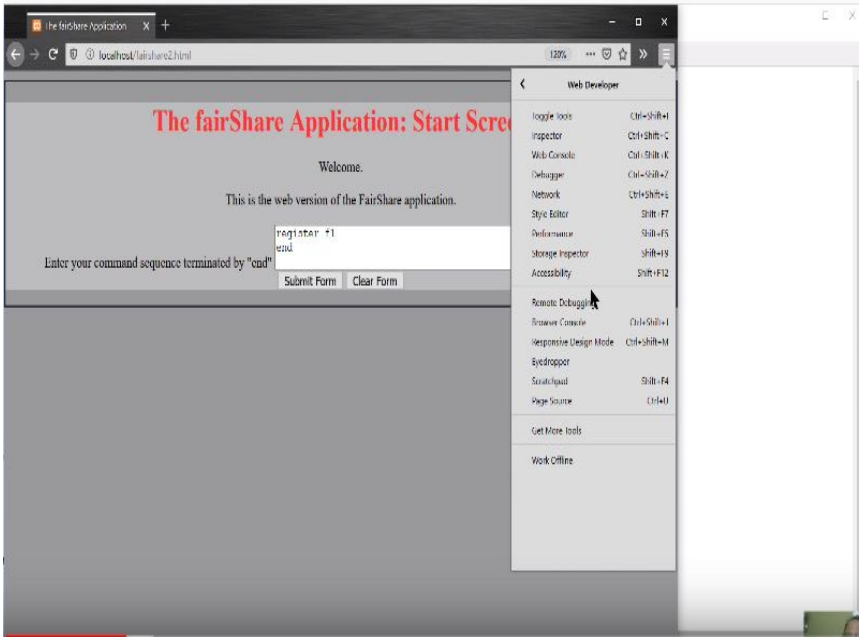
NOTE: Sometimes you have to do start-stop cycles yourself if apache fails to start on the first try. Usually the cause is that the Apache system did not shut down properly. This is very common even in real server systems. Usually, the issue for not shutting down properly is the process ID of the server has not been cleared during shut down. Typically we want only one server to start. So during startup next time, one server is already up as is recorded in a PID file in xampp/logs there is HTTPD.

(Refer Slide Time: 04:39)



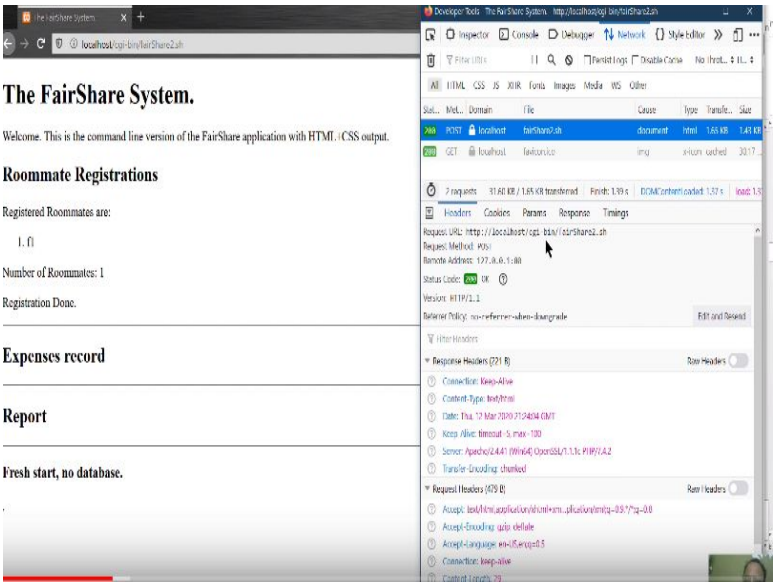
The process ID of the main process of Apache is stored here in a file(in this case, pid is 8232). The actual work is done largely by the second process(here, 18272) while the main process acts as a supervisor.

(Refer Slide Time: 05:26)

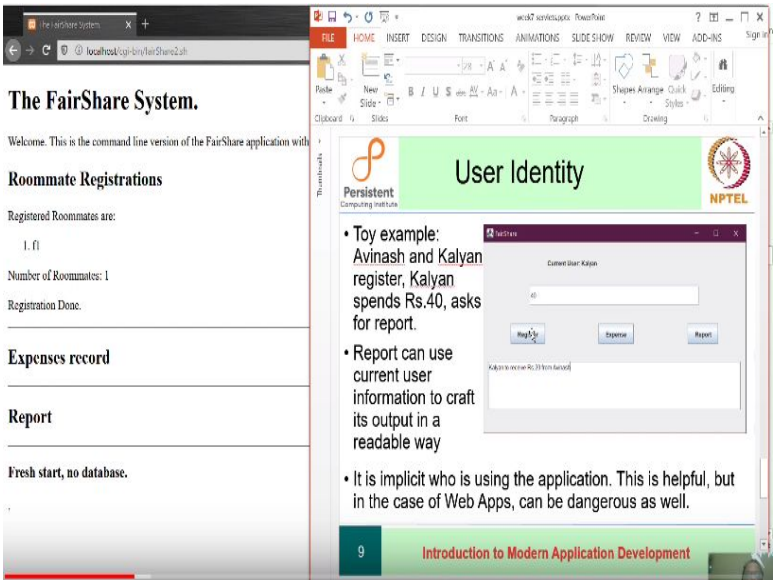


2. Open fairshare Application. In the text box, write: Register f1 and end, Submit the form. Open web developer tools and toggle tools of your browser. Select the POST request in the tool. Observe the contents of the post request. Every piece of information that we are sending across to the server is something that the server can see directly or get directly from the single page that we are working with.

(Refer Slide Time: 06:01)



(Refer Slide Time: 06:46)



3. However, consider our **GUI** system where upon the press of the register button, another screen is supposed to show up, as in the figure above and so on. This new screen remembers any information provided in all the previous pages!

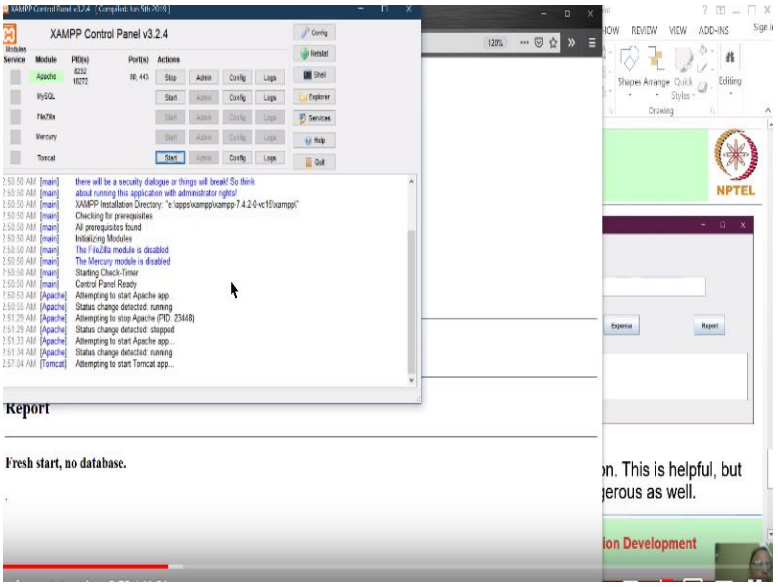
Right now, using **servlets**, every data that went to the server is the ONLY data that is available from the previous page!!

So then there is no way to remember the thing that we are working on.

GUI apps, on the other hand, have sequences of pages that achieve a goal together: for example, in the screenshot of GUI app above, it is implicit who is using the application even when the user came from the previous screen. This idea of r*emembering some information across sessions* is necessary.

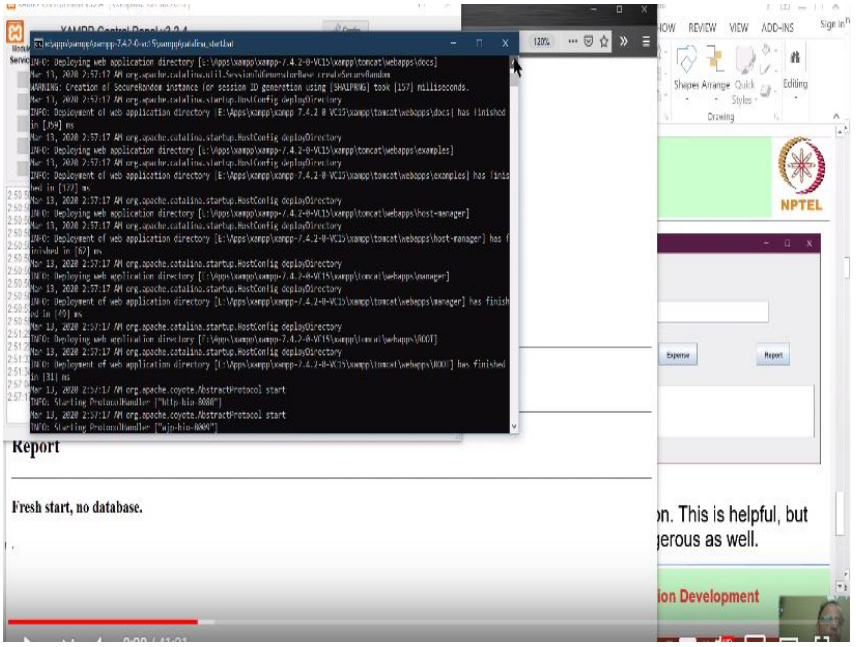
There is hence a need to **create the notion of session in servlets** so that we can remember information over multiple screens. This is achieved using *cookies*, which is a well-known word to all of us who have used the web and a specific kind of cookie is known as a session.

(Refer Slide Time: 08:58)



So, to get an idea of what those things are, we will look at Tomcat.

1. Start Tomcat.



NOTE: Sometimes "status change detected" might be reported while starting Tomcat. **(Refer Slide Time: 09:28).** The above window might pop up. When you start Tomcat, the information about the start is provided in windows like this. (The above figure says it looks for some kind of JRE, JDK etc. It finds one and when it finds one, it starts executing and it is giving us some information). It is good practice to look up these kinds of displays for warnings. Here, for example, the warning is creation of a secure random instance took xxx

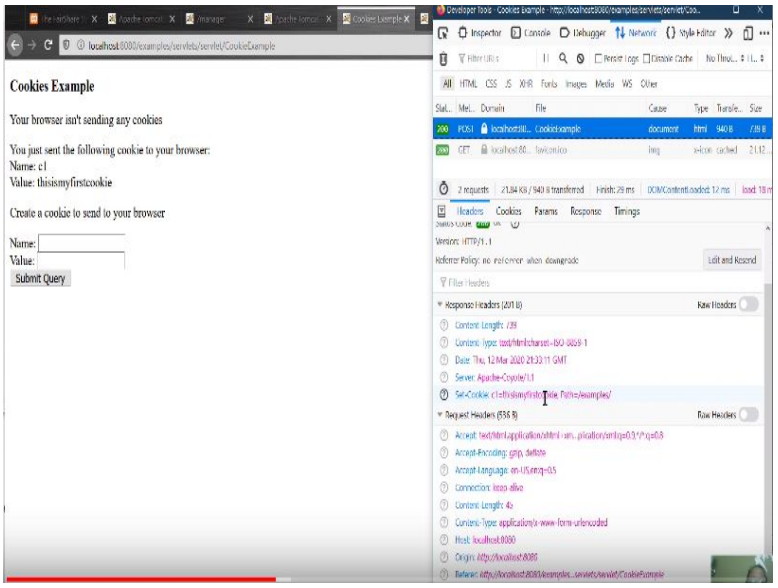
milliseconds, which might probably have been reported as the system expects it to start in a shorter time.

2. In your browser, enter the URL : http://localhost:8080. Login to the manager app. Enter the username and password set in the last lecture. (This information is available in xampp/Tomcat/conf/tomcat-users.xml).

Since HTTP is stateless, if there are a sequence of pages and each time you go from one page to another based on solely the information present in the page before, then there is no simple way to generate some sort of information, which persists across that entire series of pages, but when we have an application, we might need to remember pieces like this.

3. Go to *servlet examples* . Go to *cookies* example and let us see what this example looks like when we perform a request. The GET request elaborated in web developer tools depicts how all the information came directly from the page before and this time, there is no request body or anything.

(Refer Slide Time: 13:43)



4. Run the app, create a cookie called *c1* whose value is *thisismyfirstcookie*. The response this time has a header called **Set-Cookie.** This is where the server tells the browser to set the above cookie.

NOTE: Conventionally, cookies will normally get set automatically by the server, however the *Set-Cookie* header exists in this version too.

Understanding The Cookie information

The raw response header has the following Set- Cookie information:

```
Set-Cookie : c1= thisismyfirstcookie; Path= /examples/
```

The *Path* says that out of the parts of the URL, the *c1* cookie applies to anything that has *examples* in it.

To understand what the above statement means:

1. Set a cookie (we have already done, i.e. *c1*) in cookies example.
2. Start a watcher at this point. (we have already done, i.e. Toggle tool)
3. Execute servlet examples (In the hierarchy, we have *servlets*, which is under *examples)*

This time, the browser is sending a new header, which contains the cookie (*c1*) that was set!!

So, from now on, all interactions with this browser that happens under examples will automatically get this cookie in the request.

Takeaway:

Browser effectively says to the server that; "I interacted with you once, you sent me something which says Set-Cookie. I remembered this information for future requests. I am attaching the cookie whenever the URL to be loaded in under examples."

Where is the information about cookies stored by the browser?

To see where the cookie got set,

1. Go to "Options" in the browser.

2. Go to “Privacy and Security”-> “manage data”.
3. You will see the cookies that are set by localhost.

We can remove cookies that have been set by clearing them here.

Can multiple cookies be set for the same path?

Set another cookie *c2*, with value *secondcookie* and submit the query in cookies example (similarly enter cookie c3 with value thirdcookie).

While submitting the query, we see that the request header has information about c2 while the response header has c3. This is because the browser has c2 information and is giving to the server, while c3 has been asked to be set to the browser(by server) and hence part of the response header.

So, let us do a fourth cookie and so now the browser is sending two cookies (c2,c3), and getting back yet another one (c4).

Now, let us just submit nothing (i.e. click “ Submit” with empty text boxes). The application gets confused, returns **“name may be null or 0 length”**, giving us the **IllegalArgumentException**, but the old cookies that we had already set are properly getting sent as usual.

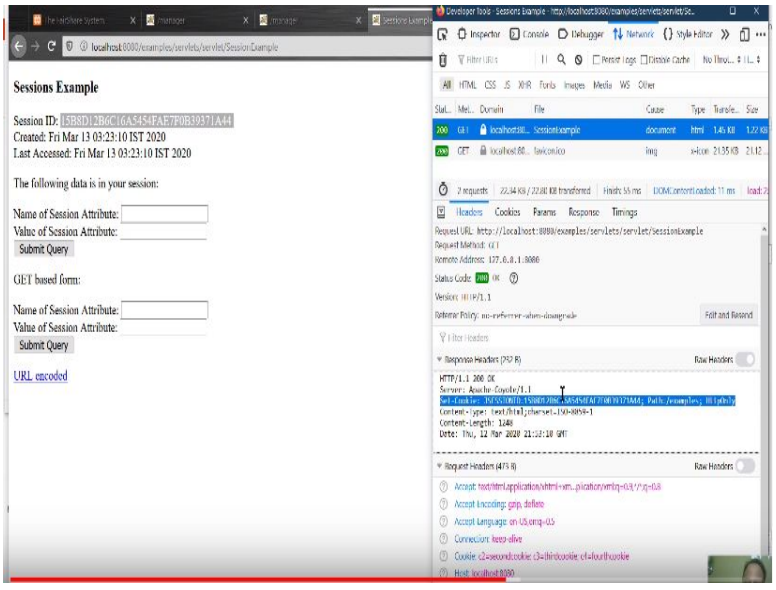
So, this is how a browser manages to send so much information. When you are experimenting with things like cookies, one useful thing you can try is to create a private window. Because in private windows, you do not store the information that goes with them and if we look at something basic like Google site, what you will see is first of all, of course, that it makes a very large number of requests that you would not necessarily expect, and these things will usually contain some cookie or the other. From the names of the cookies, sometimes you can figure out what is going on. These cookies in requests are used by web companies.

Cookies: Boon or Ban?

Amazon and other companies use cookies to understand your activity, to provide better suggestions. But this information can be misused. Tracking your activity, say throughout a day or throughout a month might be not so secure, which is one reason why from time to time, your cookies need to be cleaned. One can get extensions, which do cookie cleaning for you, on part of your browser, so that you do not get tapped for years together and some of these cookies will stay that way and get refreshed and so on far more times than you might imagine.

SESSIONS

The next interesting thing that we have to see is the idea called a *session* .**(Refer Slide Time: 26:48)**



1. Execute Session example.
2. The header above says, the moment we execute this example, a unique id is generated.
3. To see where this id is: this session id information actually i s obtained from a cookie, which is set by the server. So, what this cookie is telling our browser is that it should set a cookie whose name is JSESSIONID and the value is this particular

unique id. The path is “examples” and it is a http only cookie, so it will not get sent to a https site, if it is there. So, ultimately a session is just a cookie.

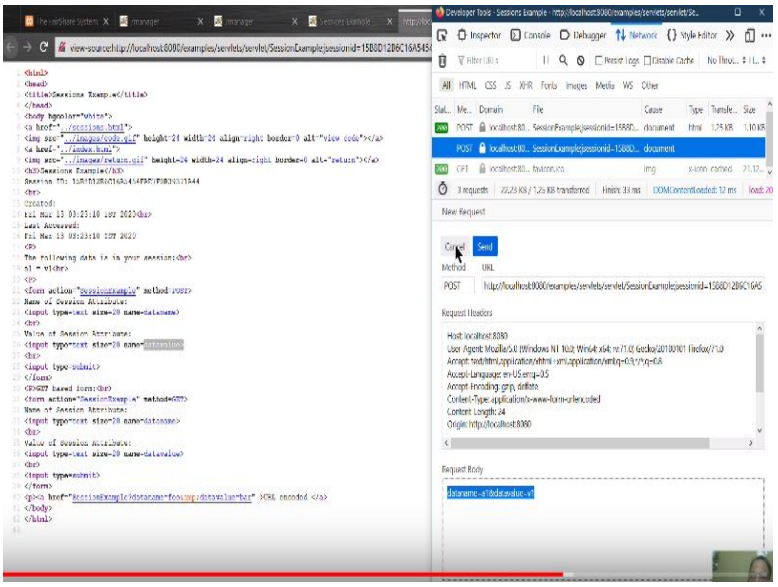
What is the distinction between cookies and sessions?

]While cookies which are used for purposes like tracking, which stays around and helps you know that it is the same person whereas a session might be temporary, for example, banks will often log you out as they say after a certain amount of time has elapsed for security’s sake. So, the things that determines how long you are logged in and whether you are logged out is what is the job of a session, but **a session is just a specific kind of cookie**, it merely managed differently from other cookies, which might be for instance, login cookies, which says that this person logged in.

So let us see what happens, so we have attribute one, and value is v1, and if we submit this guy, then what we see is, first of all the session id is put here as well, but in addition to that, it is also getting sent.

Moreover, the earlier cookies c1, c2, and c3 are different and the session id is just these things, plus we need information about where this particular data is being kept and that it can choose to keep. Store that information in a way that we will see when we look at the code of that application. So, here we are as far as this is concerned, here the request method was POST. If you look at what was being sent, you sent these two guys, so data name and data value and those are basically the names of the form.

(Refer Slide Time: 30: 12)



Just to confirm that, we can look at it like this, and you will see data name and data value in both the forms. This is what we sent. That got sent to the server as part of the request and then, as part of the response it sent back these headers and this information it baked into the page itself. So, for example, this string was generated by the server and this time, session data, is something that is not remembered by the browser, but it is remembered by the server.

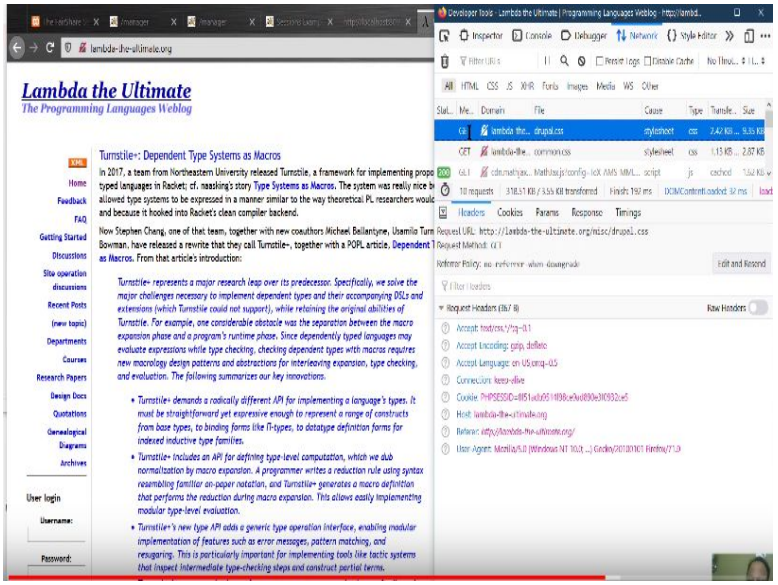
There is a cookie. Cookie is for a general purpose mechanism to store data across pages. One particular kind of cookie is called a session id and you can store data in a session, which means that the server will try to remember some data for the browser where it may put it in a file or it may store it in a database, or it may store it in any other persistent container and that data will be accessible when you give it this particular session key.

So session data acts like a key value pair on the server side and cookies act as key value pairs on the client side and the client does the job of remembering the client-side cookies whereas the server will do the job of remembering the server side information. For all browsers, cookies are a standard mechanism, whereas a session, technically speaking, is nothing that is specified by anything in the http standard.

It is something decent enough practice that was invented by people who wanted to program applications with sessions. Let us take a look at a different example of this. Visit the site “

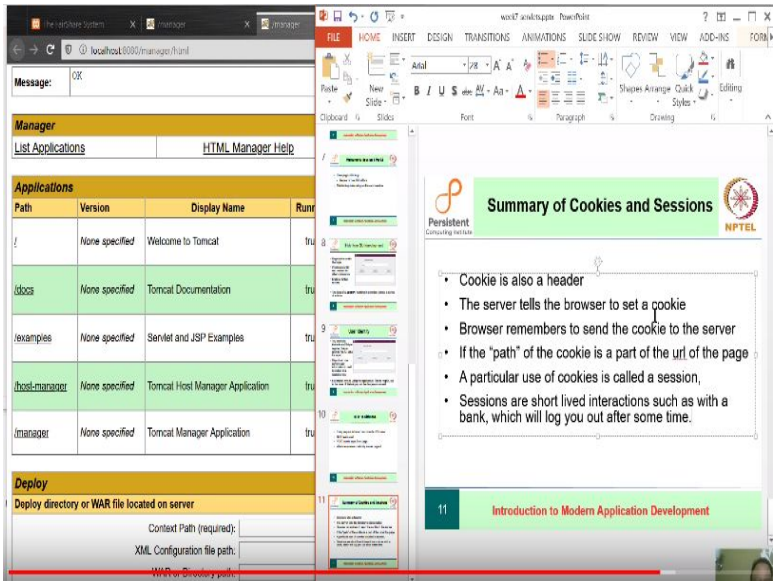
Lambda the ultimate". It is a programming language discussion site. Start a watcher here, and here if you look, there is a cookie that gets set called the PHP session id.

(Refer Slide Time: 32:41)



PHP programs that are written to create this site and they generate these kinds of session id cookies. The header is cookie, the name of the cookie is session id, and this system will remember something about our session on the server side. So, that is the overall structure.

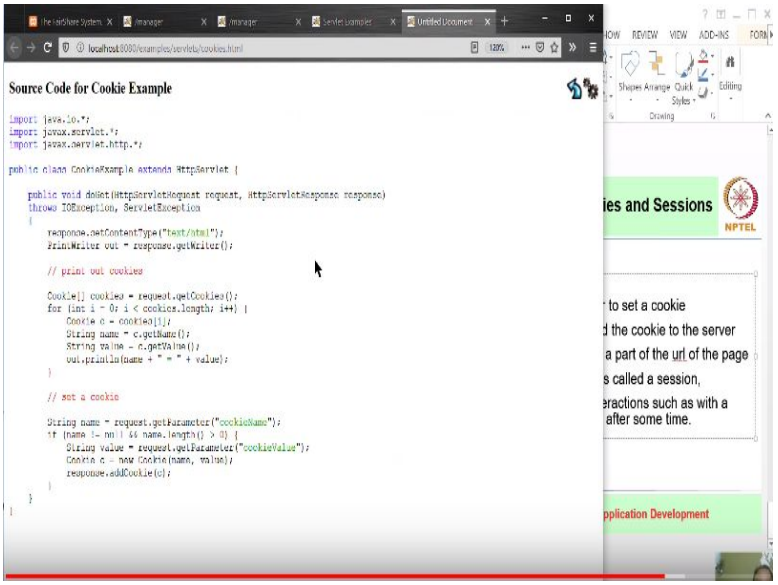
(Refer Slide Time: 33:28)



So, what we learnt is that a cookie is also a header. It is the server that tells the browser to set a cookie, and the browser remembers to send the cookie to the server if the path of the cookie is a part of the URL of the page.

A particular use of cookies is called a session and sessions are short lived interactions such as with a bank, which will log you out after sometime.

(Refer Slide Time: 35:47)



So, that is our summary of cookies and sessions. Now, let us take a look at the code for those things.

Go to servlet examples and here we have the code for cookies example.

1. The core part of the code is as shown in the Tomcat.
2. We have already seen the doGet and receiving the servlet objects.
3. It gets access to the body of the system as a PrintWriter object and from the PrintWriter object, it can get an array by using the function call, called request.getCookies and it prints them. Later if you ask to set a cookie, it gets one by saying request.getParameter(name) and once it gets that parameter name, it will get the second parameter, which is the cookie value, thus you get to set the cookie.

Let us just look at what the code was talking about.

So, here we have our cookie and if you look at the page source,we have cookie name and cookie value parameters, which are form fields, and in this code, this cookie value and cookie name is exactly what we are getting, but this time if you want to set a new cookie, you call

this function, response.addCookie, which actually creates that header we saw called Set - Cookie.

How did this Set - Cookie comes about?
It came about by means of addCookie. We use these headers to set a cookie, when requested by the server, and the browser will remember all the cookies that have been set so far for this particular path, where path is “examples”. So all the pages under examples, which includes all the servlets, which all get to see all of these cookies.

In a similar way, let us see what the source code for sessions looks. Similar to cookies, we get getSession information. With every session, there is a session object that is created on the server side, which contains this information such as HttpSession etc. It is just going to remain in memory and when you create one, it automatically gets fields like: id, created, last accessed etc.

To complete this discussion, besides the simplified code, let us take a look at the real code. So, in a real code, the difference is that while the naive code focuses purely on the setting of the session, this code also has all information necessary to generate this page i.e. current session etc. is contained in this guide and the actual work of getting the data of the session is done in this.

So, getAttributenames() and hasMoreElements() etc. is where the information is getting stored. Given a session, it is also possible to make it persist past a server removed, by storing it in a file or a database etc. and much of this machinery is actually standard in Tomcat.

Now that we have seen the one important core idea which is cookies and how a specific way of treating cookies yields sessions, we now have machinery to try to implement this multipage memory in our application, what is remaining is to understand how to get the forms to work especially with the multiple buttons and with that we will have transitioned into the world of servlets and GUI like applications.