


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 – 34
Week 12 – Part 1

Goal of this week: Complete the application and revise all the important pieces that go into constructing web applications.

(Refer Slide Time: 00:51)




The slide features a green header with the text "Week 12" in the center. On the left side of the header is the logo for Persistent Computing Institute, and on the right is the NPTEL logo. The main body of the slide is white and contains a bulleted list of topics for the week. At the bottom of the slide, there is a green footer with the number "2" on the left and the text "Introduction to Modern Application Development" on the right. A small portrait of a man is visible in the bottom right corner of the slide area.


- In this week, we are going to study
- Cookies
- Complete app logic and screen design
- Local Login
- Externally managed login
- At this point, our app will be done.
- We will briefly look back at what we have learnt
- And discuss aspects of what we have not

Sections covered this week:

1. Study cookies
2. Finish our application logic
3. Finish screen design.
4. Implement local login and some security issues that arise in logins.
5. Look at how externally managed login is done and then the app is essentially done.
6. Discuss some of the features you can add to apps **(Refer Slide Time: 01:33)**




State: consequence of history



- Http is "stateless" I
- That is: every request to the server by the browser is independent; there is nothing to link one request to another
- TCP is "stateful" : both parties maintain connection information
- But state is necessary, because we need to causally relate a sequence of requests, such as a shopping cart.
- One shoppers cart should not be paid for by another shopper
- Carts must not be accidentally exchanged
- **State is the current result of a specific historical sequence of events.**

3
Introduction to Modern Application Development



COOKIES

We talked about maintaining application state, the ability to refer to application screens by a URL and when you visit the URL to be able to see the application where you left it off that is the fundamental idea behind application state.

Statelessness: The protocol itself is said to be stateless what this word means in this context is that every request to the server as far as the protocol design is concerned is independent. There is nothing inherently in HTTP at least as it was originally designed that would link one request to another that is insufficient to build applications as we will see shortly. TCP on the other hand is said to be a stateful protocol because both parties maintain connection information that carries a sort of summary of the historical exchange called a handshake that has happened between the two parts of two ends that create the connection.

Statelessness is good because it means that neither party needs to remember much about each other or at least not remember in a way that is crucial to the functioning of the protocol. So, for example when you maintain a connection right a phone connection let us say or a TCP connection which is more or less the same thing. Both parties have to agree to talk, one has to agree to lift the phone the other has to press the button and create the call and they both need to maintain their respective roles otherwise the connection can get dropped.

In contrast HTTP request is something like a letter let us say you write a letter and then all you know is then the letter has reached the person or in the context of phones or whatsapp , you send a mail and it reaches the person you may know that it has reached them but that is

about it you do not know anything else. You will get some response back from them and the very minimal necessary response is yes I received your letter.

But the subsequent letter may or may not carry any connection to the first one that is up to you and the party you are communicating with. So, in this sense when neither party is obliged to maintain a contact of behavior across multiple messages you can do interesting things like rapidly interleave a communication you are having between say 5 different people and they would not necessarily know and your capacity you can just refuse to talk to somebody in the middle and they cannot do anything about it in a sense right.

Now in a connection based system that is not quite the case once you agree to hold a connection there is an implicit contract that you would not break it until both of you are happy with the breaking. Now this is not hard and fast right, we do break connections and so on and so forth but at least while that discussion is going on both parties hold knowledge about each other.

So in general holding knowledge about each other is necessary for example as I have said here state is necessary because in a shopping cart for instance one person shopping cart should not be presented to another for payment right. They must not accidentally exchange carts the items that one person puts in their cart should not accidentally end up in somebody else's cart because if an addition of an item into a shopping cart is an independent HTTP request then anything could happen right.

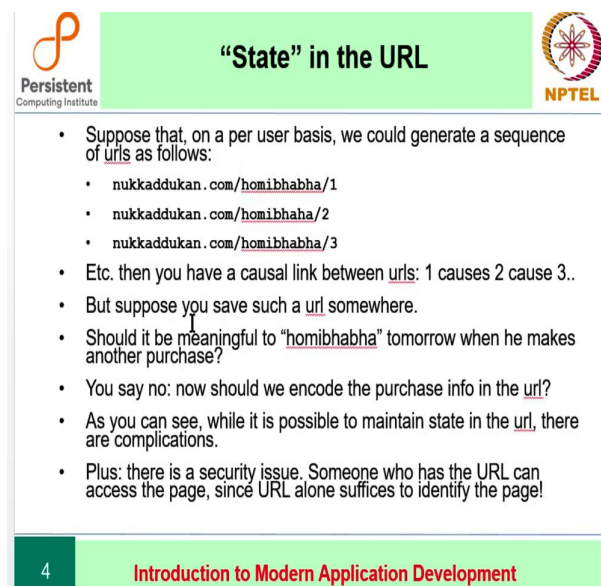
So that we cannot allow and if you want to maintain that sequence then in effect you are maintaining history. So, state is a consequence of history right, it is the current result of a specific historical sequence of events. To the degree that it matters to your application state may or may not be important. A classic site like Wikipedia does not care right, it just shows you the document you come back to, you do not come back does not matter.

On the other hand, a webstore will most certainly care that you complete your payment for instance as much as they would prefer it because that increases their number of customers. So, these kinds of things depend on the application the bottom line is the important part is state is the current result of a specific historical sequence of events and we need a way to represent that if the protocol is unwilling to represent it because it wants to be a stateless

scaleable protocol then it is the data communicated by the protocol that must maintain state that is the distinction.

So you can always get the effect of having state, the question is whether it is built in into the interactions dictated by the protocol or in the data that the protocol carries. In the case of HTTP we have a sort of a light hybrid, for all practical purposes HTTP is still stateless but you can build state on top of HTTP reasonably easily.

(Refer Slide Time: 07:36)



“State” in the URL

- Suppose that, on a per user basis, we could generate a sequence of urls as follows:
 - `nukkaddukan.com/homibhabha/1`
 - `nukkaddukan.com/homibhabha/2`
 - `nukkaddukan.com/homibhabha/3`
- Etc. then you have a causal link between urls: 1 causes 2 cause 3..
- But suppose you save such a url somewhere.
- Should it be meaningful to “homibhabha” tomorrow when he makes another purchase?
- You say no: now should we encode the purchase info in the url?
- As you can see, while it is possible to maintain state in the url, there are complications.
- Plus: there is a security issue. Someone who has the URL can access the page, since URL alone suffices to identify the page!

4 Introduction to Modern Application Development



So, when I say things like *build state on top that means there is a way to represent historical knowledge that is what it means to say build state*. Suppose that on a per user basis right, one way to build state so to speak would be to change the structure of the URLs themselves. So, you have your corner shop and it creates you know URLs like this for a customer called homibhabha and they produce a sequence of URLs 1 2 3 4 etcetera.

Then you can say that, as far as the server of this new per token knows we have created a puzzle thing one possessed 2 causes 3 etcetera. But now suppose you save the URL so the question is should it be meaningful to mr. homibhabha and to paducah when he makes another purchase. The answer is probably not right, they do not transfer over from one purchase to another. And if you bookmark it then is it supposed to be meaningful some days from now etc those kinds of things become issues.

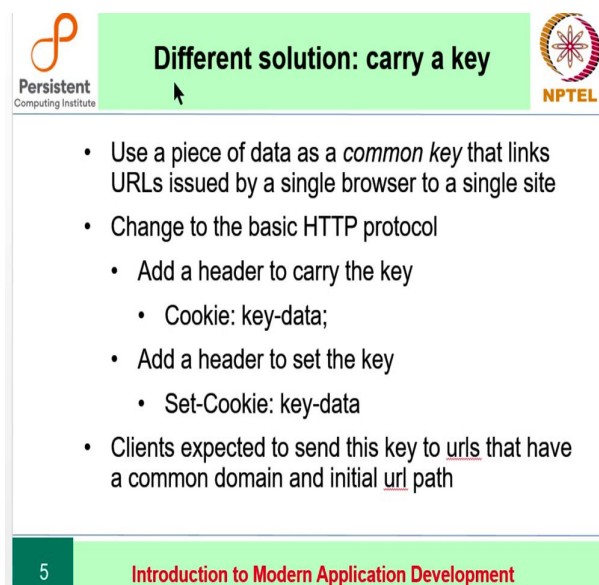
So state and the larger the history you remember the larger this sequence gets, furthermore suppose you branch off you go forward in one direction in one window backward in another

direction in another window then you might start putting numbers like this right 3 dot 2 dot 1 or something like that to indicate, so, if the entire history of exactly what happened. So, it may not be enough to just have this much.

And in practice what it shows is the more state you try to end in code in the URL it keeps on getting bigger and bigger. And it is kind of clumsy and there are problems with bookmarking and so on and so forth plus there is a fairly serious security issue because if the URL carries its history and you send the URL to somebody by accident then they can continue your history where you left off that is actually probably the most serious issue.

Otherwise you could do something to compress the data etcetera etcetera but the security factor makes it impossible ok. So, what do we do?

(Refer Slide Time: 10:04)



The slide features a green header with the title "Different solution: carry a key" and logos for Persistent Computing Institute and NPTEL. The main content is a list of bullet points explaining the use of cookies as a common key for URLs. A footer bar contains the slide number "5" and the course title "Introduction to Modern Application Development".

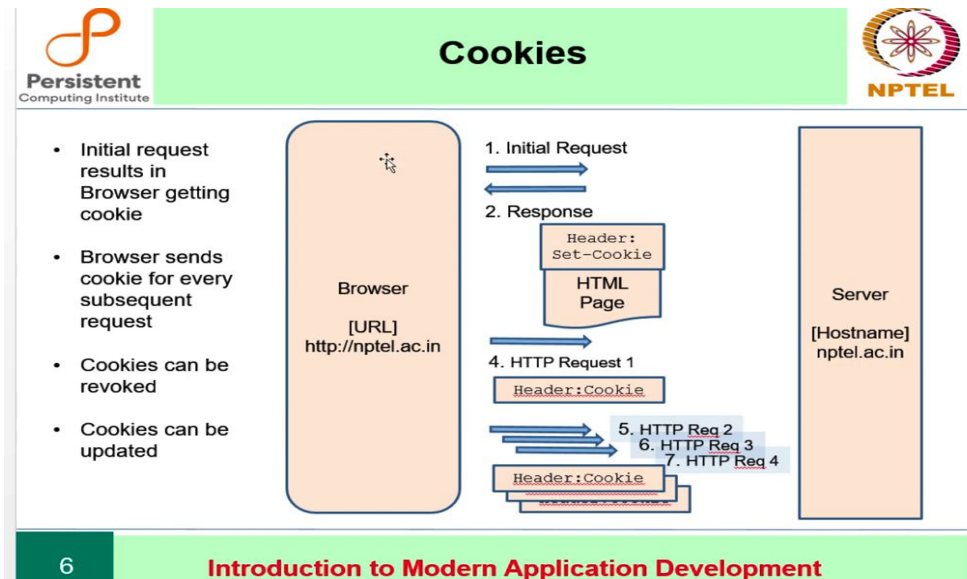
- Use a piece of data as a *common key* that links URLs issued by a single browser to a single site
- Change to the basic HTTP protocol
 - Add a header to carry the key
 - Cookie: key-data;
 - Add a header to set the key
 - Set-Cookie: key-data
- Clients expected to send this key to urls that have a common domain and initial url path

The answer is to have a different solution, you make a slight change to the protocol. We tell the browser that it should carry along a small piece of data which is a key that identifies the URLs in some fashion. We use a piece of data as a common key that links URLs issued by a single browser to a single site. And we make a change to the HTTP protocol we say that will add a header to carry the key called cookie and we add a header to set the key.

So, the server tells the client that it should set a key and the client agrees to carry the key in subsequent exchanges. And so clients are expected to send this key to URLs that have a common domain and initial URL path we will see all of the details of this in our demo very shortly ok.

(Refer Slide Time: 11:05)

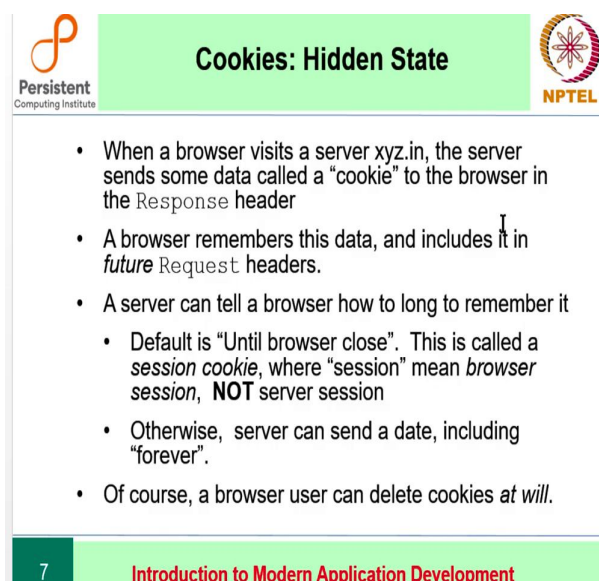
So, here is a picture you have a browser it makes an initial request this initial request does not have any cookie or anything obviously the browser is visiting for the first time. Now the se



Server in its response sends a header called set cookie and gives value together with the HTML page that it returns in response what the browser says is for all subsequent requests 1, 2, 3, 4 etc it will continue sending another header called the cookie with the value.

Such cookies can be revoked. You can have a said cookie that removes a cookie and you can update a cookie because if said another response comes back with the said cookie of the same key as we will see then the current value gets updated. So, that is something that can be done.

(Refer Slide Time: 12:00)



And technically here is what we have, when a browser visits a server xyz dot in the server sends some data called a cookie to the browser in the response header. A browser remembers this data and includes it in the future request headers. A server can tell a browser how long to remember it, this is quite important. So, the default is until the close of the browser as soon as you close the browser and restart it the cookies are gone.

Such cookies are called **session cookies**, now that word session gets used in two different ways in this world. One a browser session is from when you started the browser to when you stopped whereas a server session is when the client first contacted the server and then did not contact it for a sufficient amount of time that the server forgets about it. So, those are the two ideas of the session. *A session cookie refers to a cookie that the browser maintains until the close of the browser.*

In another case a server can send an end it can say that you should expire this cookie by such and such a date and you can give a date arbitrarily in the future. So, that the cookies supposedly is the client is supposed to preserve it forever. Of course this can only be advisory right because you cannot destroy cookies and the client may say oh well I have too many cookies I am just going to get rid of some of them.

You are carrying the state information in the data conveyed by the protocol not the requests of the protocol itself. In contrast, in TCP you may have seen there is something called a **TCP state machine** if you have not seen it you can look it up on Wikipedia where *there are states that tell you where the two parties that are talking to each other are that maintain what is happening with their connection.*

(Refer Slide Time: 14:23)


Persistent Computing Institute

Server Browser Relationship

NPTEL

- Note that a server *cannot force* a browser to do anything!
- This important feature allows many types of servers and browsers to evolve independent on one another.
- *They only need to agree on the **PROTOCOL***
- The magic here is not obvious, and it may not make sense to you why it is not obvious: after all you are used to it.
- It is difficult to overstate just how amazing an idea it is, because it so seemingly obvious in *hindsight*.

8 Introduction to Modern Application Development



So, in general **the server and browser are independent of each other**, a server cannot force the browser to do anything this is crucial. Because this means that server and browser are independent enough to evolve independently as long as they use the same protocol in talking to each other. *This is quite a magical idea although it is difficult to explain just exactly what is magical about it.*

The magic is apparent historically this was not a obvious idea to invent and previous methods of building distributed applications there were systems called corba behind this and that they were quite terrible to the point where they are more or less forgotten that is not to say everything was terrible just lots of things were terrible in comparison the web and the HTTP protocol are wildly successful because they carried this notion of protocols rather than the methods that **Corba** used like language based interfaces.

And it is sort of difficult to overstate just how amazing the idea is but it is not that easy to explain unless you have actually experienced how people were thinking in the absence of the world of protocols. And these ideas are some of those things that seem obvious in hindsight of course not all protocols are the same HTTP is a particularly interesting design. And we will briefly talk about the philosophical ideas in HTTP as we close the discussions in the course.

(Refer Slide Time: 16:10)

Persistent Computing Institute NPTEL

Server attaches meaning to cookies

- I
 - A server sends the cookie, and receives it back from the browser
 - *Cookie is meaningless to a browser*
 - At this point, the server can use to cookie to decide what to do about the request
 - Is the cookie valid
 - Is it too old
 - If neither, use they cookie in some fashion as a key to reveal application data to the user

9 Introduction to Modern Application Development



So that is one part, now the second is the meaning of the cookie resides in the server it is meaningless to the browser, browser just blindly sends the cookie and whatever the server does it does once a server receives a cookie it can decide what to do about the request it can check whether a cookie is valid it can check whether it is too old. If it is neither then you can use the cookie in some fashion to respond to the user.

(Refer Slide Time: 16:44)

Persistent Computing Institute NPTEL

Demo

- Quick Cookie Firefox addon
- How cookies work: session and persistent
- Code for cookies
- What we are doing in our demo is weird:
 - We are letting the client specify a cookie
 - Server then sets it
- Usually: the server decides whether or not to set the cookie
- Tomcat, in particular, always sets JSessionId

10 Introduction to Modern Application Development



DEMO

Pre requisite:

Install browser add-on called **cookie quick manager**.

So when you press this icon it gives you these options and then it says manage all cookies. So, when you start looking at all cookies because we have recently visited this domain addons dot Mozilla dot org you can see that both dot Mozilla dot org and addons dot Mozilla dot org have set a bunch of cookies. We are not going to spend time trying to understand these cookies because I do not know much about them first of all and anyway we are going to run our own cookie demo so that you can understand what is exactly going on.

But let us just go over what this shows which says that cookies are associated with a domain like I said the HTTP protocol was changed to agree that the client will supply a cookie to the domain where it came from but not just the domain there is something else called the path which also plays a role and we will see an example of that shortly. So, it says that this cookie for instance has the name underscore ga and its value is this.

Now whatever it is, it carries some meaning to mozilla, they are using it to track something or the other. And it is supposed to expire at this time, i.e. in 2022.

Start one of the servlet managers, start tomcat. When you log into the manager you can see that we got a new cookie here for localhost. And the localhost cookie shows that it is a J session ID cookie so this J session ID about which we will learn shortly is a cookie that is set by the servlet container which is Tomcat and it has some value here. It says that this cookie is valid for the path slash manager so the URL has to be localhost slash manager.

If the URL you are contacting has this part localhost slash manager as it is starting part of the path then the browser will set the cookie to the manager application itself. Another thing it note says this is a HTTP only cookie so if we had a HTTPS connection this will not work and it is set to be a session cookie. Now if you recollect a session cookie is supposed to be here.

So until browser close so this is called a session cookie as opposed to this cookie from you know Mozilla which had an expiry date and it is not a session cookie. It is not HTTP only so this will get sent to both HTTP and HTTPS and the path is slash here the path is slash so which means everything every single path on Mozilla dot org will receive this cookie. There is some other thing here about strict and lax and whatnot.

There are lots of details like this on the web: if you do it professionally you have to know that

but for basic concepts this is what we are doing this good enough ok. So, here we are, we have created a new application called cookie hello and we are going to demo it. So, let us open until all right what cookie hello does is it takes the same form that we have been working with just changes it to do these 3 activities: *set cookie*, *delete cookie* and *persistent cookie*.

And then it shows you which cookie you have, what is the name and what is the value all right. Now what does our let me get rid of this add-on tab and what does our system say? So, if you notice this number changed we now have two cookies and we have to J sessionid cookies one is the first one which is slash manager and the second is from cookie hello. So, because Tomcat is a servlet container it as soon as a browser visits it for the first time on a unique path it issues a new J session ID cookie.

This is standard built-in functionality for session maintenance whose utility we will see in a short while. So, already we have had our first result now see this page is supposed to show you the cookies so there is a problem. What happened, why did the system not show the J session ID cookie. The answer lies in something fairly interesting. So, the cookie has been set and the cookie has been received right.

So which means our browser currently has this cookie but consider how this page was created? When the page was created there was no cookie right. So, because there was no cookie the page was created on the server side the server is showing you the page as it saw at the moment you made the request, at that time there was no cookie. Now let us see what happens if we make the same get request again watch, the session ID cookie we have is this our cookie hello application is here when we went to the server for the first time the server made the webpage sent it to the browser there was no cookie visible to the server at that time.

But we know there is a cookie because the browser add-on is telling us there is, so let us see what happens. As soon as we made the second request the cookie is now visible because the server had access to the cookie even though we made the same request again. So, let us see what the server saw. So, the request headers look like this: the browser is sending this cookie 1b5 etcetera and this is the same cookie that we see in this cookie manager right.

Let me maybe move this to a new window so that we can see both at the same time. So, here we have it here is our J session ID cookie 1b5 etc, this 1b5 so on. So, the request had sent it

which is why the server could see, how did the server see it. Let us look at some code ok. So, when we made a get request the server ran this thing from the cookie hello class. It got the hello JSP and ran this function show cookies which looks like this.

You can read about this part on the web for you know how to see cookies in a in a servlet the basic idea is the HTTP request object has a function called get cookies which returns an array of cookies and we do exactly what we did for the database we loop through the cookies and turn the name value pairs into a list of maps right. In this case there is only one map in this list because all cookies are stuffed into a single map.

But by doing this we can use the same JSP code that we had used previously which is this one so hello and here we are so we have a for each with name value bears this for each loop which creates which loops over this thing called cookie data and ck name and ck value is generated here. So, we said that tribute cookie data and we create this list so what happened is the browser sent the request the server extracted the cookies from the request itself between go to the database on just the request and then generated the page and sent it back to us.

Now if we repeat this there is no new cookie getting set so nothing is happening even though we are making the request again and again. And here is what we have: this is the local cookie. So, that is as far as setting the J session ID cookie is concerned but in this application we are going to do something kind of interesting. We will actually send a key value pair and you can put anything in here.

A key value pair and on the server side we are going to take this key value pair and turn it into a cookie and set it. So, let us do a set cookie call, watch this, we are going to make a post call. So, we made the call and this is what the headers look like. So, the request header right is a post header, it still carries this cookie but we are telling the server what took it to set via the field. So, the field is a + 3734...

And this encoding of spaces plus is a standard thing that happens called URL encoding I do not I think we have talked about it before but if not you can look it up the name is URL encoding. In any event this is what we have sent and then look at what happens the server sends back a response which actually contains a set cookie. And if we look at our if we refresh this thing here localhost now shows us three cookies, so this is our cookie.

So in fact the browser has received the cookie but as before right because the server generated the page for us it is still not showing the cooking and if I do refresh we will see the cookie, so that is the basic idea. Now deleting the cookie is the same kind of thing you just say which cookie you want to delete. We will hit it and here is what we have. So, our request was like this our request was the body of the request is here.

We have said the field is a and the button is deleted then what happened was the browser process this it is still sending the cookie because we have not deleted it yet the request goes through. But then the browser comes back and says set the cookie to empty right and set its expiry date in the past if you look carefully in your own browser this is a little too faint in your own browser then you will see that the expiry date has been set to 1 Jan 1970 which is basically in the past.

And so the cookie should be deleted but because the page was generated when the cookie was still there we still see it in the page. If you wanted to you could write some JavaScript code to make this happen directly on the client-side but that is ok this way you understand the cycle better. And now if you refresh it right I am just hitting refresh the cookie is gone, the cookie for localhost is gone.

So the browser has already deleted it and now if we refresh again the cookie is gone so that is set and delete. So, delete is basically setting the cookie right here if we use our standard style which is to get the parameter and say set delete and the persistent cookie then what we see is if you want to set a cookie you get the parameter split it on space and to the response object you run the add cookie command and add a new cookie object.

For delete you get the cookie name from the field and generate a new empty cookie set max-age to 0 so 0 because of the unix heritage is interpreted as 1 Jan 1970 and when you add the cookie the browser goes and removes the cookie next time around that is. That is next time there will not be any such cookie in the requested sense ok. Now one thing about so I am going to set a few cookies to show you what persistence and not persistence means.

I have set cookie again the update has not happened and then I have something again set cookie and now I see the previous cookie if I want to see the current one I can do resend and I

will see of course the it doesn't have to be a resend of the of the form it can just be a get request that part does not matter at all so even if it is a get you will still see the cookie back because of the way we have written the code.

That is we have not bothered about get or post just show cookies happens in either case. So, here we have to actually take the action such as setting and clearing and then show the cookie here we do not take the action and just show the cookie all right. So, that is the basic idea and so I have a bunch of cookies here if I refresh this refresh our add-on then the two cookies are visible.

What you will notice is that we have the key value, we have a key value, we have a context which is the cookie hello and these are session cookies. So, which means what I expect to happen is when I close the browser and restart it the cookies should not be there. On the other hand I am going to set another cookie bbb12345 set it as a persistent cookie, here let us take a look at the request what it says is set a cookie with an expiry date.

So I am setting it to expire at 3:40 GMT whatever it is and about an hour or so however I did it. The code is here. We set the age to so the general idea is you know seconds minutes hours and days and so this is basically set to one hour. So, in another hour if we close the browser and get back the cookie will be gone but they are not going to wait that long. I just want to show you the difference between persistence and non persistence.

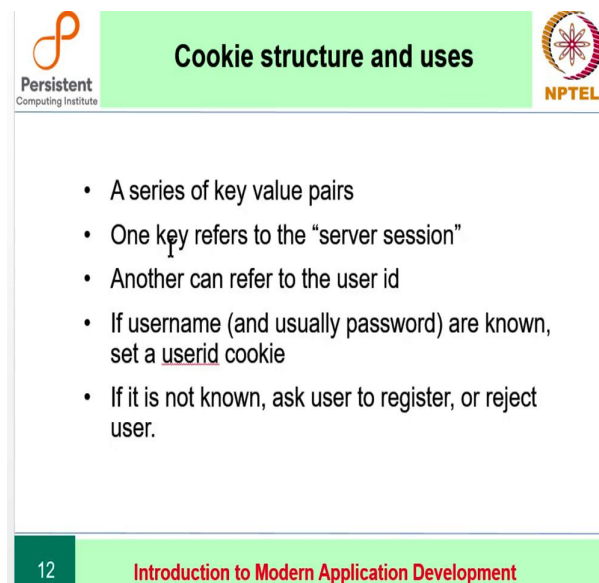
So the persistent cookie has not shown up. I am going to issue a get request and I can see my bbbb which was a persistent cookie. If you go to our add-on and you hit refresh what we see is a cookie called bbbb which is not a session cookie and it is going to expire in an hour all right so now what I am going to do is just close the browser down and restart it. So, we exist and I have this Dave Fox thing so let us open a new window.

In that window I will start the cookie manager as you can see localhost still has one cookie bbbb123 etcetera so that is persistent. Reasonably straightforward kind of elegant and once you have this add-on cookies are pretty easy to understand. They are also visible in the standard inspector and so on but just to show you technically you do not need the add-ons and whatnot. But the add-ons make the interpretation much easier.

So we have our toggle tools and if you go look at this stuff I think it is somewhere in memory storage are there are cookies that are surprisingly ok. I will have to look at this to see why it is not showing the persistent ok, maybe it keeps it somewhere else at all ok. So, if I find this out I will show you next time alright. So, that is our discussion of how cookies work right and what we are doing in our demo is not usual we are letting the client specify a cookie normally the server will just do it on its own.

And so what we have is we are sending this form field right and then there is a set cookie call coming back from the server telling us to set the cookie. You can again in JavaScript setup cookie completely locally so you can do it all in browser if you want. And so the effect of set cookie can be seen after one refresh all right.

(Refer Slide Time: 40:50)



The slide features a green header with the title "Cookie structure and uses". On the left is the logo for Persistent Computing Institute, and on the right is the NPTEL logo. The main content area contains a bulleted list. At the bottom left, a green footer displays the number "12" and the text "Introduction to Modern Application Development". A small video inset of a speaker is visible in the bottom right corner of the slide frame.

- A series of key value pairs
- One key refers to the "server session"
- Another can refer to the user id
- If username (and usually password) are known, set a userid cookie
- If it is not known, ask user to register, or reject user.

So, in summary cookie is a series of key value pairs one key refers to the server session another can refer to the user ID username and password and all can be carried there are problems doing that you know it is not a good idea to do it as simply as it seems we will discuss that in the next session. And if it is if the cookie is not known then you can you know take action based on that. So that is the basic idea.

(Refer Slide Time: 41:22)


Persistent
Computing Institute

Next session

NPTEL

- Implementing the login page
- Cleaning up app logic and screens

13 Introduction to Modern Application Development



And we will now use cookies to implement the login page, understand sessions and so on
alright see you next.