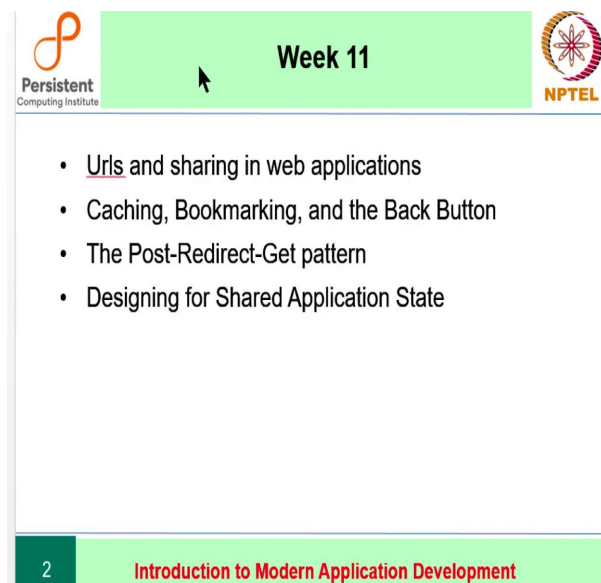


Introduction to Modern Application Development
Prof. Aamod Sane
FLAME University and Persistent Computing Institute
Abhijat Vichare
Persistent Computing Institute
Madhavan Mukund
Chennai Mathematical Institute

Lecture – 32
Week 11 – Part 1

Basic screens of an app and how to make sure that several of the nuances of user interfaces and the database interaction turn out have been discussed.

(Refer Slide Time: 00:33)



Week 11

- Urls and sharing in web applications
- Caching, Bookmarking, and the Back Button
- The Post-Redirect-Get pattern
- Designing for Shared Application State

2 **Introduction to Modern Application Development**

Goal of the lecture:

Study an aspect of web applications of sharing links to web pages. This is a novel idea that exists only in web applications.

1. In the fairshare web app created earlier, upon pressing the report button, the url has appended `/report`. This is done so that further tags such as date in the form `/2020-04-22` can be appended to the url and that url can be shared so the exact state of the report is received and not of older days. This means one can execute an application as it goes stage by stage but reference can actually exist to each of those stages that you can share with other people.
2. We can cache the contents locally so that you can move back and forth. For example,

in one of our standard database applications if we register users, u1, u2, u3 and so on one by one, all of these pages are actually still available in the history of the browser. So upon clicking the back button, we can see the older users and their pages in this app. However, since a single URL is used in the database app, we cannot actually share these independent screens.

Which part of our application should be shareable?

- For instance, credit card data is input in some app, it shouldn't be permitted to see the credit card info even if by mistake the user sends the URL to another user. So, *not all states should be shareable*.
- On the other hand, some states are absolutely shareable. Example : A document website like Wikipedia.

There are two extremes; in an app that involves security details, sharing needs to be an absolute minimum.

The gray line:

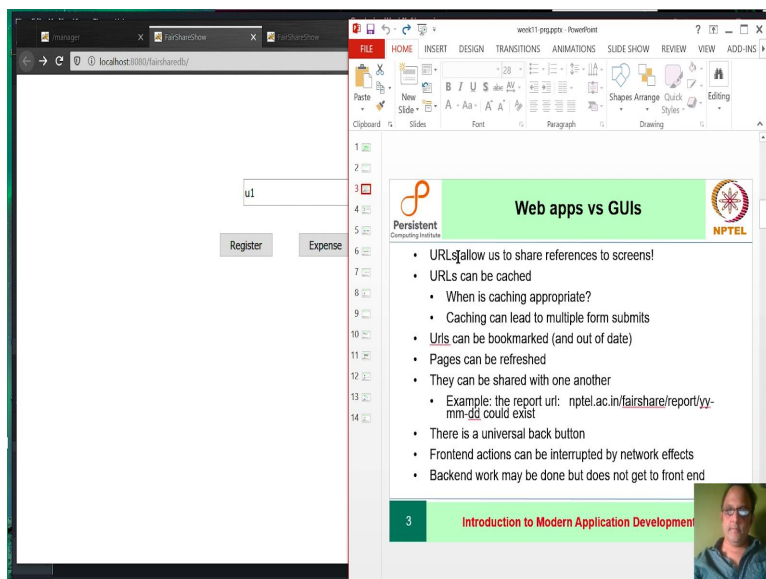
In most apps, something in between is desired. Example: e-commerce site should allow to add items into a cart and then come back to it another day even though the user has logged out of the browser.

The ubiquitous back button.

- Back button is a user requirement : One of the more surprising studies showed that the back button is probably the most used button on a browser and it is very popular and people do not like it when they cannot use the back button.
- Back button is tricky: Banking sites often either prevent using the back button or notify that pressing the back button will create loss of data. So thinking about the back button is an important part.

Post redirect get pattern which will be discussed in the next lecture deals with the decision of caching a URL, bookmarking and visibility of an application state.

If going back and forth amongst forms can happen, the browser will give a note in the dialog box incomprehensible for a naive user (for a programmer this might make some sense), so if a user hits resend they might get small errors. **(Refer Slide Time: 08:09)**



To summarize:

1. URLs allow sharing references to screens.
2. They can be cached which leads to decision of when it is appropriate.
3. They can be bookmarked and they can get out of date.
4. They can be refreshed.
5. They can be shared with one another under certain conditions.

SHAREABLE LINK : An explicit way of deciding with whom to share the link

Google Docs(for instance) provides a shareable link which, by definition, makes the document shareable. If a person has the link and browses it, Google decides whether the user can access the link.

1. If it is so specially crafted where the permission has been granted by the Creator then surely the user can access.
2. But it can do more than share just among people who have the link: it can share

widely, it can share within an organization, etc.

So, sharing can be controlled at a varying grain.

Varying Grain of Sharing affects the design of the Universal back button:

One is front-end actions can be interrupted by network effects.

Consider the fairshare app: If the register button is clicked with the same data on a mobile phone, it could get interrupted or even silently dropped or there could be a bug in the browser any number of things could have happened. The other thing can also happen which is the front-end actually succeeds in contacting the backend but then the backend effect occurs but the front end is not refreshed. So, quite unusual states about an application can arise because of this kind of sharing as well as because of network effects.

How a URL actually refers to the state of an application?

Variations while visiting fairshare application for the first time from Tomcat manager:

We see a 'GET' in header (if it had login functionality then there would be some context which says it was done by means of a cookie)

Upon registering users and expenses for some users one by one, we create quite a bit of browser history in the back button. All the methods were POST while registering and adding expenses.

Upon clicking the back button at this stage, only GET request. It does not make any other GET requests. The browser is not visiting the database at all when further back and forth happens. There is memory of all the pages seen and then one can go back and forth among them.

But the first time one is entering data, some action is done, so the database is accessed and

some results are returned. When going back and forward we just use the cache URL as we saw. If you revisit after some time, now in our case we are not using cookies or anything but things can happen for example the cookie may have expired. If you bookmark and revisit the cookie may have expired or it may be deleted.

As we know a cookie is used by browsers to remember to identify the requests and to connect them with each other. And if you share it by sending the URL to someone, the incoming person has no cookie. In fact there is no context at all other than what is in the URL. So, in other words the context and what you see depends on when you visit, how you visit, what type of data it is and so forth.

There is no requirement that a browser should have a cache. It could be that the browser always visits the database in order to get the state. And in that case going backwards is a little strange because you cannot undo what the application has done. So, there is a strange sort of disconnect, you have to think a bit about this when going forward we talk to the database we make changes and you show the next screen.

But going backward does not mean you are undoing it, you are merely looking at the picture you had before you did the submission so that is crucial it is just an image.

(Refer Slide Time: 16:10)

The screenshot shows a web browser window on the left and a presentation slide on the right. The browser window displays a form with a text input field containing the value 'd4'. Below the input field are three buttons: 'Register', 'Expense', and 'Report'. Underneath the buttons is a table titled 'Expenses By User' with the following data:

ectime	usrId	amount
1587571128	88	22.00
1587571119	89	10.00
1587553315	32	10.00
1587553160	32	999.00
1587552942	32	0.00
1587552743	32	0.00
1587551838	32	222.00

The presentation slide on the right is titled 'Application State Sharing' and contains the following text:

- Every screen represents a sharable state of an application
- E.g. put items in cart -> add payment -> add address -> finally submit
- Allowed to move back and forth among the screens
- Screens contact various services, e.g. payment gateways
- Important fact: Moving among screens *invokes actions*
- And – moving backward cannot just undo!

Let us say that out with a specific example; every screen represents a shareable state of

application in the case of e-commerce it might be putting items in a cart, adding a payment and addresses then submit. When you move back and forth the screens contact various services and moving among screens can invoke actions. But not all the back-and-forths can be meaningful because you cannot undo say a payment right.

So it is not as if you are rewinding time only that part which is under the control of the browser can be reborn. This as you can imagine is problematic because suppose you have completed a credit card payment then you go back and if the credit card data is still there then going forward will make a second purchase which is definitely not what you want. While going back will not undo the purchase you have already made.

Create a disconnect between the state of the database and what is seen on the screen on the other side.

Moving forward among screens especially invokes actions. And moving backward *cannot* just undo the actions.

(Refer Slide Time: 17:54)

The screenshot shows a web browser window on the left and a presentation slide on the right. The browser window displays a form with a text input field containing the value 'd4'. Below the input field are three buttons: 'Register', 'Expense', and 'Report'. Underneath the buttons is a table titled 'Expenses By User' with the following data:

etime	usrId	amount
1587571128	88	22.00
1587571119	89	10.00
1587553315	32	10.00
1587553160	32	999.00
1587552942	32	0.00
1587552743	32	0.00
1587551838	32	222.00

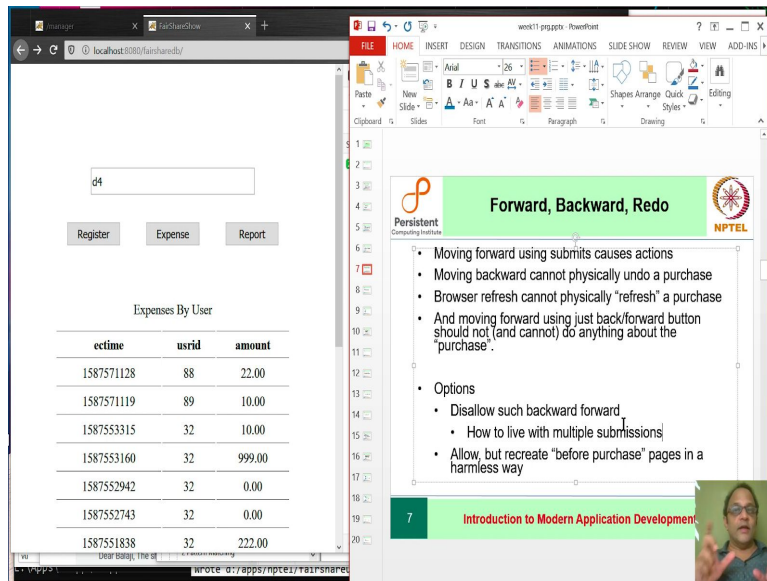
The presentation slide on the right is titled 'Crucial idea: Actions vs Views' and contains the following bullet points:

- Ecommerce carts allow us to move back and forth
- But a ticket cannot be purchased twice
- Some sites limit number of tries on login for security
- While an action can create a screen, moving back and forth among screens should not always repeat the associated actions
- Just exactly how this affects app design is what we will study next

The slide also features the NPTEL logo and a small video inset of a speaker in the bottom right corner.

Limit the number of tries : Cannot always repeat actions and cannot undo them and so need to take that into account when designing the application.

(Refer Slide Time: 18:18)



Scenario: Consider an ecommerce site. Moving forward in the browser by doing submits causes actions. Moving backward cannot physically undo a purchase. Then just a browser refresh cannot physically refresh a purchase and moving forward just the pages, just moving forward using just the back forward button should not and cannot of course do anything about the physical action about the purchase right.

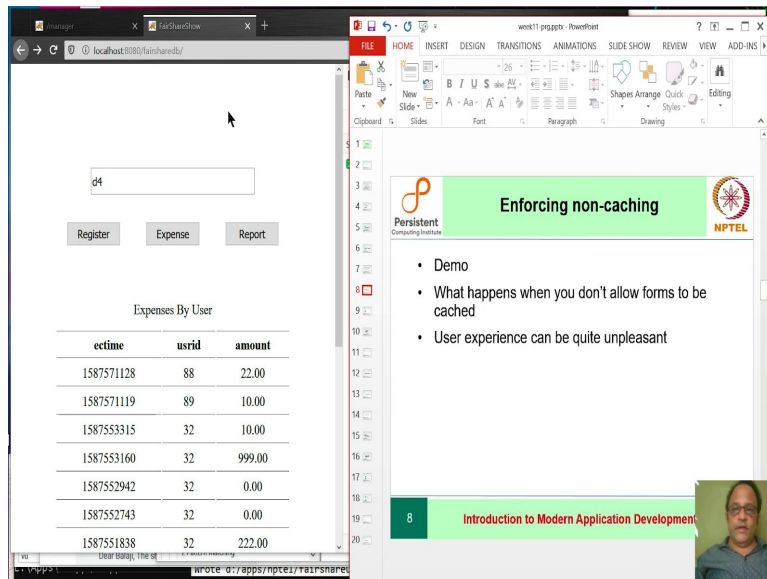
So, if that is the case and if there is a disconnect between the browser and real life unreviewable activities then what should we do?

1. Completely disallow such a thing.
2. A cleverer design allows it but recreates the before purchased pages in a way that makes it clear that whatever else is done on the page the best that will happen is in effect going forward.

The second option is much harder to do .

So there are many scenarios and many decisions to be taken care of as a developer. So before the app is designed, the developer needs to have a mental model where real-life actions have happened, the browser is showing something and coordinating between the two is an important design issue.

(Refer Slide Time: 22:09)



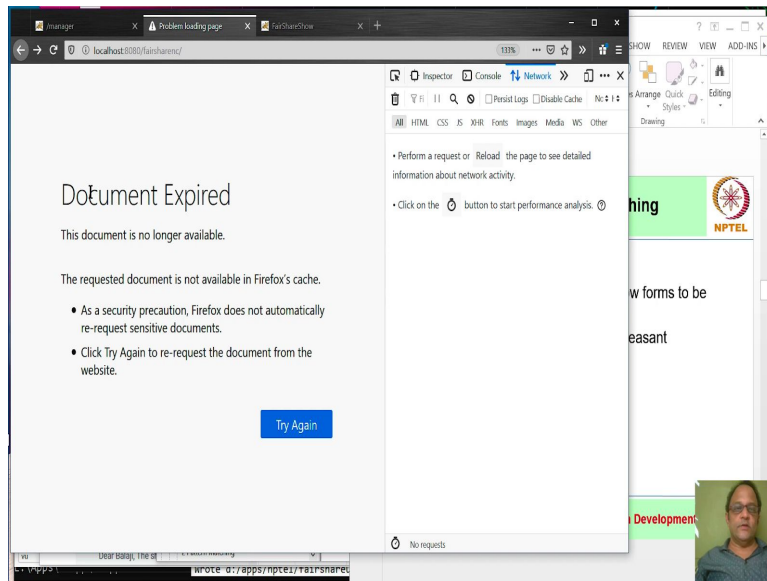
What does the user experience when we have a non caching version of this system?

Consider a new version called fairsharenc for this scenario and take a look at what the network requests.

Register and add expenses one by one and ask for a report. We get different headers, one is pragma which is a header that says no cache. Another header is 'expired and finally cache control (The response header in the cached version had content length, content type, date and server).

These are the three new headers we added to tell the browser that it should not cache previous pages.

(Refer Slide Time: 25:04)



Upon clicking the back button now, it says the document is expired or it is no longer available. And then it will say that as a security precaution Firefox will not automatically re-request sensitive documents, click try again. If the user tries again, other mechanisms are needed to stop them from going forward and pressing the button again.

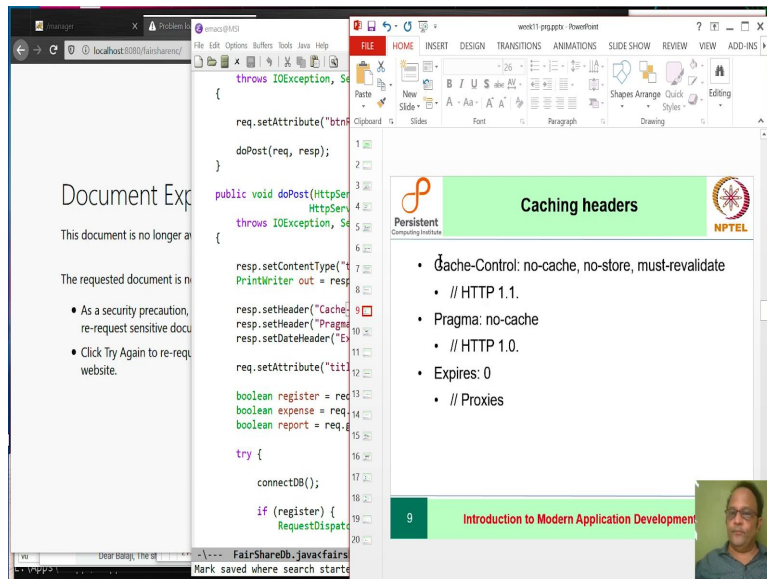
But if the user reloads the original page and goes ahead, even the forward document is now expired. So, the user is stuck: going backwards and forwards gives the document expired. Although it is correct in terms of the working, it is quite a horrible experience. This is handled by adding control to the back button using JavaScript to ask to login. So, although avoiding caching is certainly possible, it is also the case that the users' experience is not that great.

Preventing caching can be necessary but it can lead to bad experience. So in general we should try our best to minimize this kind of experience on part of the user.

Implementation of cache control:

Consider the non caching version of the system.

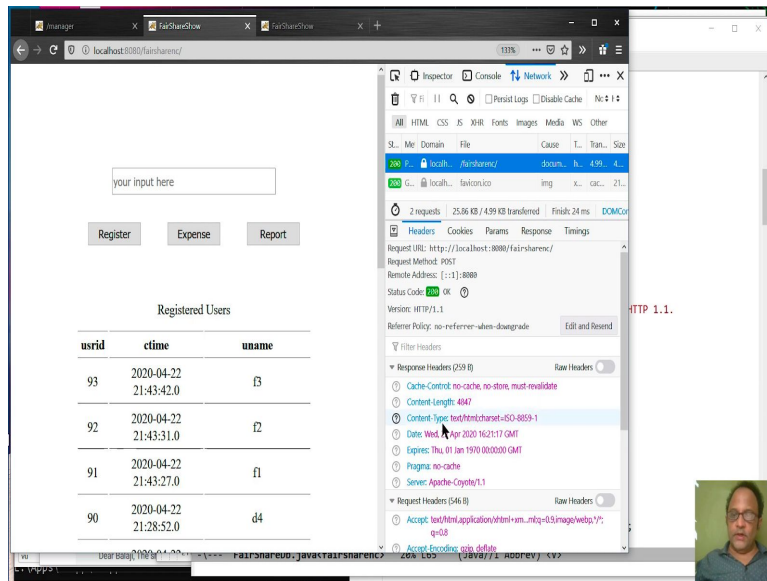
(Refer Slide Time: 27:34)



Here is what you have to do: the headers we saw are these 3 headers, cache control which says no-cache, no-store, must-revalidate. The reason for many headers largely has to do with history. The header which says that the expiration date is 0 that is the expiration date is 1970 remember time in this world begins in 1970. This is needed for intermediate components called proxies. We are not going to say too much about proxies in this course just enough to know that they exist.

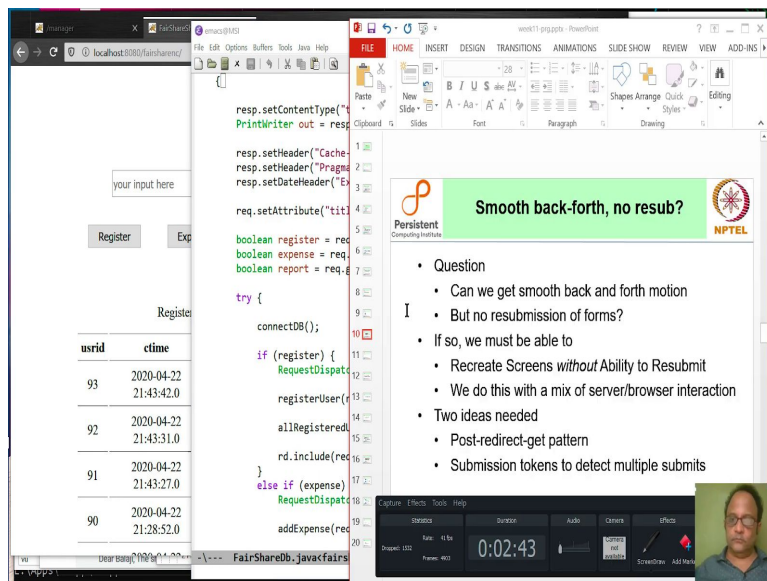
And as far as Java is concerned the code that sets these things looks like this: so you set a header to cache control and pragma saved where and there is something else called day-trader which has an expire score and when that is received you get this sort of user experience. Just to see this thing again let us hit register.

(Refer Slide Time: 29:08)



And we will see that cache control expires and pragma these headers have been added to the these headers have been added to the response. So, that is as far as the programming is concerned and the rest of the code remains the same.

(Refer Slide Time: 29:29)



The question that remains for us is can we actually get smooth back and forth motion but with no resubmission of forms. And if so we must be able to recreate screens without ability to resubmit or we must be with resubmission that happens because you are detecting duplicates or something like that. And you can achieve this effect with a mix of server and browser interaction. Two ideas are needed here post redirect get pattern, submission tokens to detect multiple segments.

Once these two ideas are in place we will be able to get the flow to be as smooth as we want does not mean you always want to allow arbitrary backtracking but if you do the things you learn will show you how to actually do it.

(Refer Slide Time: 30:33)

The screenshot shows a web browser on the left displaying a form with 'your input here', 'Register', and 'Exp' buttons, and a table of registered users. The table has columns 'userid' and 'etime' with the following data:

userid	etime
93	2020-04-22 21:43:42.0
92	2020-04-22 21:43:31.0
91	2020-04-22 21:43:27.0
90	2020-04-22 21:28:52.0

The middle part shows an IDE with Java code for a REST API:

```
resp.setContentType("text/html");
PrintWriter out = resp.getWriter();
resp.setHeader("Cache-Control", "no-cache");
resp.setHeader("Pragma", "no-cache");
resp.setDateHeader("Expires", 0);
req.setAttribute("title", "Register");
boolean register = req.getParameter("register") != null;
boolean expense = req.getParameter("expense") != null;
boolean report = req.getParameter("report") != null;
try {
    connectDB();
    if (register) {
        RequestDispatcher rd = request.getRequestDispatcher("register.jsp");
        rd.include(request);
    } else if (expense) {
        RequestDispatcher rd = request.getRequestDispatcher("expense.jsp");
        rd.include(request);
    }
} catch (Exception e) {
    out.println(e.getMessage());
}
```

The right part shows a presentation slide titled 'POST-REDIRECT-GET' with the following bullet points:

- Let us first study post/redirect/get,
- This motivates the need for "addressable URLs"
- The next step is to examine resubmission tokens

The slide also features the NPTEL logo and a small video inset of a speaker in the bottom right corner.

Let us first study post redirect get which among other things will tell us the need for addressable URLs addressable as in URLs that you can share with people that have adequate context so that if you just type in the URL, the application knows what to do even if you do not have a cookie for instance. And then the next step after that is to examine resubmission tokens and resubmission tokens is something we will talk about after we are done with cookies because they are part of the same set of ideas. So, for now we will study addressable URLs and the post redirect get method.