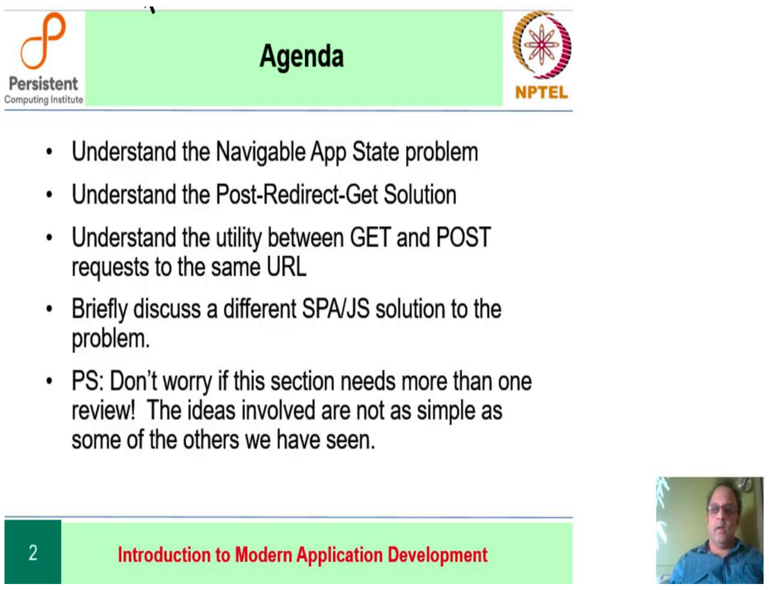
# Introduction to Modern Application Development

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

**Lecture – 33**
**Week 11 – Part 2**

Goal of the session: Navigation of application state problem and a solution.

This problem was introduced in the previous session during the discussion about caches where the issue that view in the browser versus the state on the backend can get out of sync was addressed. **(Refer Slide Time: 00:42)**



## Agenda

- Understand the Navigable App State problem
- Understand the Post-Redirect-Get Solution
- Understand the utility between GET and POST requests to the same URL
- Briefly discuss a different SPA/JS solution to the problem.
- PS: Don't worry if this section needs more than one review! The ideas involved are not as simple as some of the others we have seen.

2 Introduction to Modern Application Development

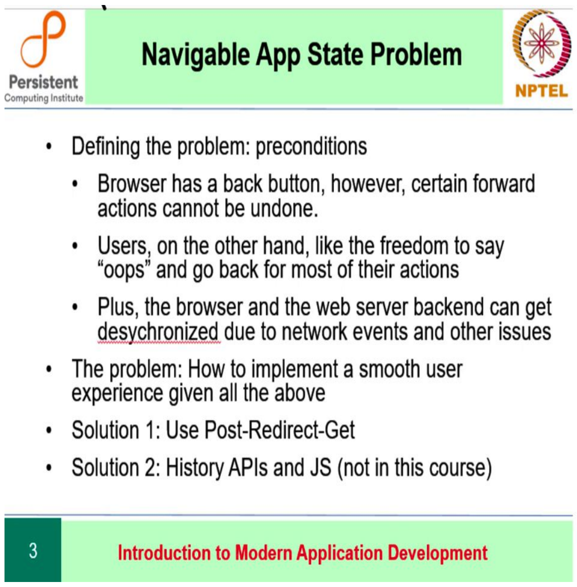
So the agenda is:

- To understand the navigable app state problem: What happens when one wants to make the application state visible via URL.
- Understand how post-redirect-get solves the problem.
- Understand the utility of having get and post requests made to the same URL. (You can do this first of all which is one new thing and second of all that it is useful in solving a particular literal key issue.
- There is a different solution to this application: the navigable app state problem which is used in some applications that relay Javascript. They are called single page applications, will be briefly discussed but they are not part of this course.

The discussion about the navigable app state problem is specific to the case of the web because of the way the browser is a user interface which is separated from the back end.

Many of the things that we will see in the session will be something commonly experienced on one website on another. So, one can relate some of the experiences such as not being able to press the back button of the browser. Also one can understand what happens at the level of a program when the back button is pressed multiple times.

**(Refer Slide Time: 03:18)**



Navigable App State Problem

- Defining the problem: preconditions
  - Browser has a back button, however, certain forward actions cannot be undone.
  - Users, on the other hand, like the freedom to say "oops" and go back for most of their actions
  - Plus, the browser and the web server backend can get desychronized due to network events and other issues
- The problem: How to implement a smooth user experience given all the above
- Solution 1: Use Post-Redirect-Get
- Solution 2: History APIs and JS (not in this course)

3 Introduction to Modern Application Development

Problem definition: Navigable App state problem

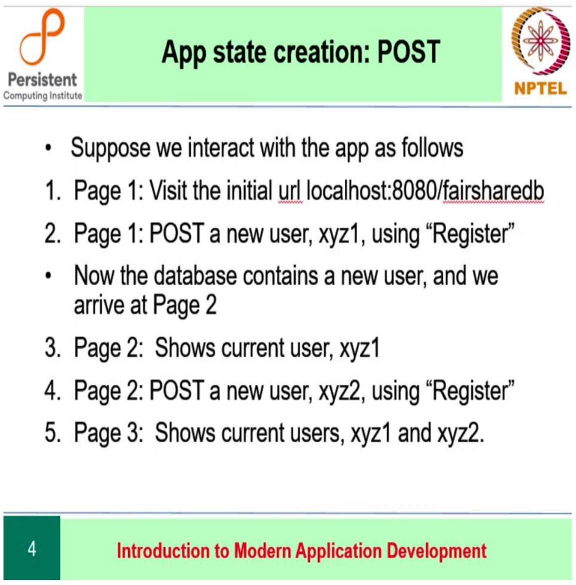
Precondition for this problem : A browser which has a back button in the interface, but the back button cannot undo the actions that we did when going forward (for example: purchases). On the other hand, we as users like the freedom to say oops and go back and maybe rethink and do take an alternative path and so on. The browser and the web server can be de-synchronized because there may be network events or issues on the machine.

Given these preconditions, how does one implement a smooth user experience?

Two solutions:

1. Post redirect get style
2. Using history APIs in JavaScript

**(Refer Slide Time: 04:22)**



## App state creation: POST

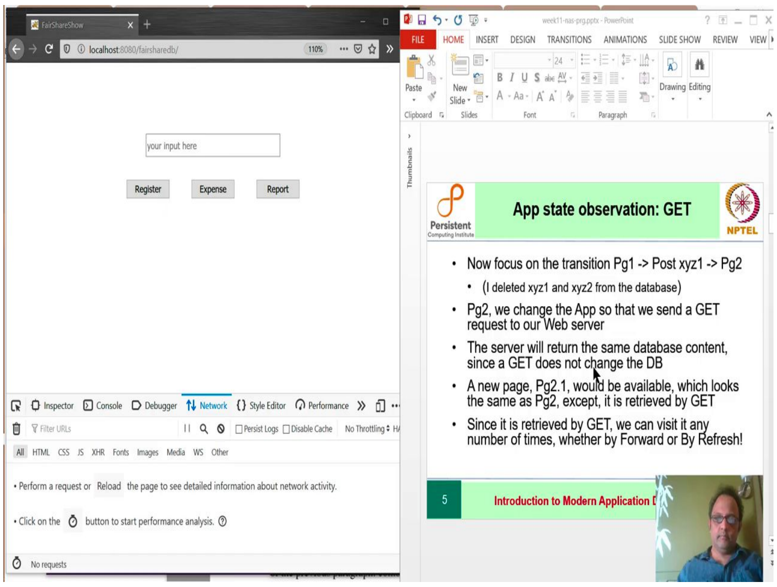
- Suppose we interact with the app as follows
1. Page 1: Visit the initial url localhost:8080/fairsharedb
2. Page 1: POST a new user, xyz1, using "Register"
- Now the database contains a new user, and we arrive at Page 2
3. Page 2: Shows current user, xyz1
4. Page 2: POST a new user, xyz2, using "Register"
5. Page 3: Shows current users, xyz1 and xyz2.

4 Introduction to Modern Application Development

Look at the above figure. Let us do the following to understand the scenario.

1. Let us interact with the app we have visited fairsharedb.
2. Visit the page first with a get request.
3. Create a new user xyz1 and register. As expected, the method here is post. The user xyz got registered and record his data back so post a new user we have arrived at page 2. The page two is not identified on the URL however, you can see that there are two pages in the history.
4. And now we are going to create another user xyz2 and do a second post. So we got xyz2 and we have a second post.
5. If we go back we will see the old page. Remember the database is not different. It actually has 2 users but this page was never updated to that because the page itself resides only on the browser and so you are just seeing whatever snapshot the browser saw when it requested the page. This is what it means for the reality which is the database to get desynchronized from the view.
6. If you go back still further, of course, there is nothing, this is the first page we have visited.
7. We go forward so the state of the database at this point is the real state but 2 other states are accepted or apparently accessible to us, but they are not really there. And this is why in some sense it makes sense to cache.
8. It makes sense to disallow caching of the old pages because they are showing you things that are not true. So this is our visible evidence that there is a de-synchronization.

**(Refer Slide Time: 07:19)**


## App state observation: GET

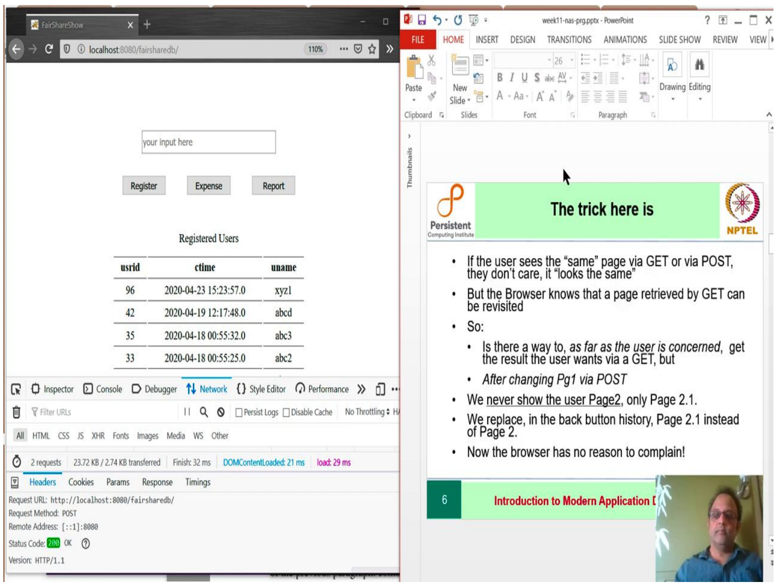
- Now focus on the transition Pg1 -> Post xyz1 -> Pg2
  - (I deleted xyz1 and xyz2 from the database)
- Pg2, we change the App so that we send a GET request to our Web server
- The server will return the same database content, since a GET does not change the DB
- A new page, Pg2.1, would be available, which looks the same as Pg2, except, it is retrieved by GET
- Since it is retrieved by GET, we can visit it any number of times, whether by Forward or By Refresh!

FairShareShow
localhost:8080/fairsharedb/
your input here
Register
Expense
Report
Inspector
Console
Debugger
Network
Style Editor
Performance
• Perform a request or Reload the page to see detailed information about network activity.
• Click on the button to start performance analysis.
No requests
Persistent
NPTEL
5
Introduction to Modern Application

Suppose we had a way to just send a request which gets whatever is in the database at this time. This can be done using ‘get’. The server will return the same database content because gets do not change the database and a new page would be available. Let us call it page 2.1 which should look the same as the page on the screen because we have just retrieved the data as it is. On the other hand because we know that this page was retrieved by get and it does not change the database.

I could repeatedly make this request again and I still see the same thing. So it is not how you reach a particular state that matters that you have reached the state of page 2 using the post. Because the way we have written our get function it does not; it is not supposed to change the dataset.

**(Refer Slide Time: 10:50)**



Registered Users

| usrid | ctime | uname |
|---|---|---|
| 96 | 2020-04-23 15:23:57.0 | xyz1 |
| 42 | 2020-04-19 12:17:48.0 | abcd |
| 35 | 2020-04-18 00:55:32.0 | abc3 |
| 33 | 2020-04-18 00:55:25.0 | abc2 |

```
2 requests   23.72 KB / 2.74 KB transferred   Finish: 32 ms   DOMContentLoaded: 21 ms   load: 29 ms
Request URL: http://localhost:8080/fairsharedb/
Request Method: POST
Remote Address: [::1]:8080
Status Code: 200 OK
Version: HTTP/1.1
```

### The trick here is

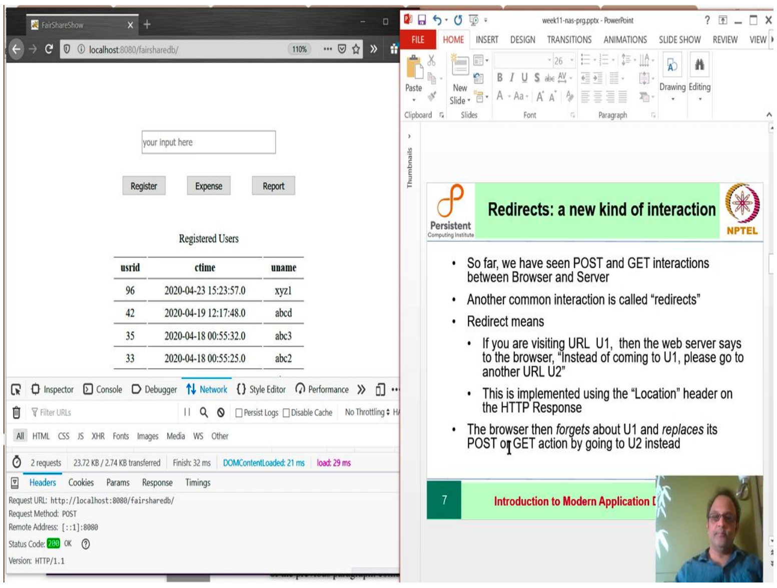
- If the user sees the "same" page via GET or via POST, they don't care, it "looks the same"
- But the Browser knows that a page retrieved by GET can be revisited
- So:
  - Is there a way to, *as far as the user is concerned*, get the result the user wants via a GET, but
  - *After changing Pg1 via POST*
- We never show the user Page2, only Page 2.1.
- We replace, in the back button history, Page 2.1 instead of Page 2.
- Now the browser has no reason to complain!

6

The trick we are after is, if the users sees the same page whether via a get or via post, the user does not care if it looks the same but the browser knows the difference that one was visited by post. And so it will cause a change in the back end if you resubmit it. The other was via get and so it does not matter what you get.

*So the solution will be this way: Find a way as far as the user is concerned, to achieve the request that the user wants via a get method.*

**(Refer Slide Time: 12:50)**



Registered Users

| usrid | ctime | uname |
|---|---|---|
| 96 | 2020-04-23 15:23:57.0 | xyz1 |
| 42 | 2020-04-19 12:17:48.0 | abcd |
| 35 | 2020-04-18 00:55:32.0 | abc3 |
| 33 | 2020-04-18 00:55:25.0 | abc2 |

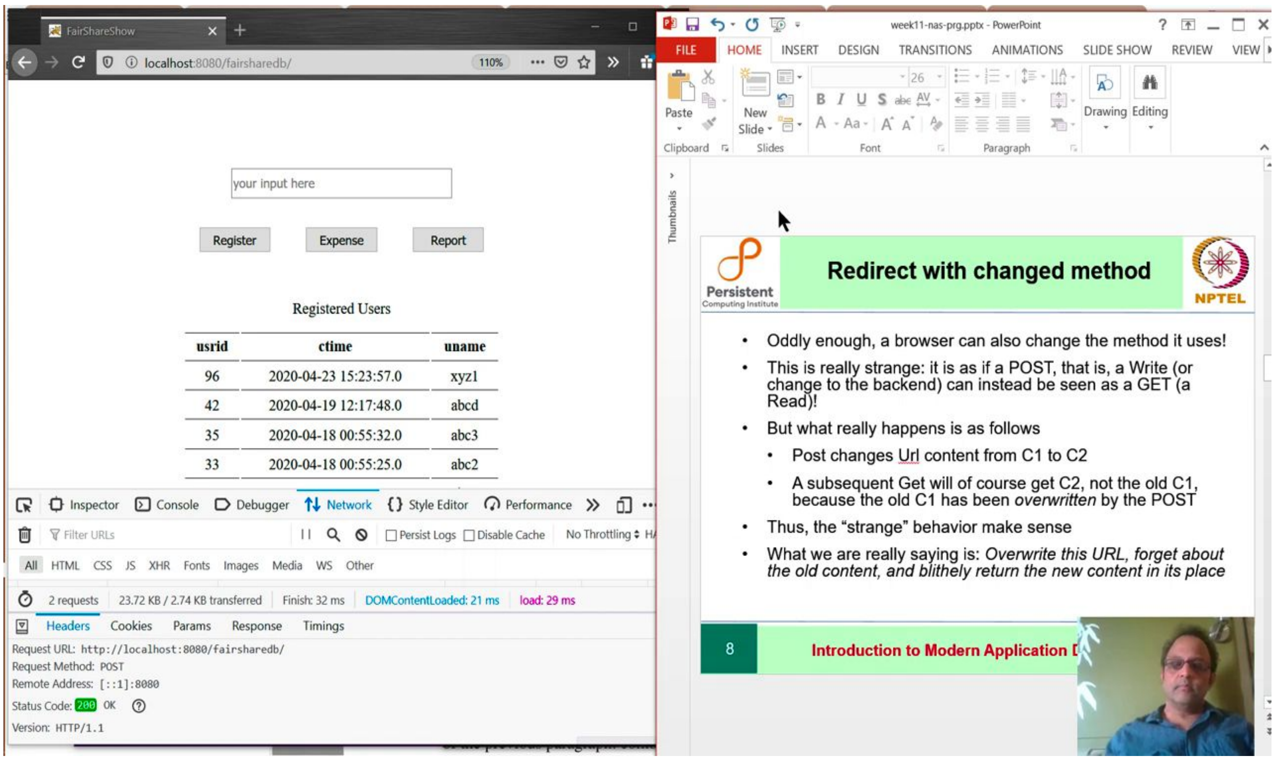
```
2 requests   23.72 KB / 2.74 KB transferred   Finish: 32 ms   DOMContentLoaded: 21 ms   load: 29 ms
Request URL: http://localhost:8080/fairsharedb/
Request Method: POST
Remote Address: [::1]:8080
Status Code: 200 OK
Version: HTTP/1.1
```

### Redirects: a new kind of interaction

- So far, we have seen POST and GET interactions between Browser and Server
- Another common interaction is called "redirects"
- Redirect means
  - If you are visiting URL U1, then the web server says to the browser, "Instead of coming to U1, please go to another URL U2"
  - This is implemented using the "Location" header on the HTTP Response
- The browser then *forgets* about U1 and *replaces* its POST or GET action by going to U2 instead

7

**Redirect** is a new kind of interaction and the meaning of redirect is, if you are visiting URL U1, then the server tells the browser instead of coming to location U1, please go to another location U2. This is implemented as a header called location in the https response. In that case, the browsers forget about U1 and replaces its post or get action by going to U2 instead.**(Refer Slide Time: 13:41)**


FairShareShow
localhost:8080/fairsharedb/
your input here
Register
Expense
Report
Inspector
Console
Debugger
Network
Style Editor
Performance
Filter URLs
Persist Logs
Disable Cache
No Throttling
All HTML CSS JS XHR Fonts Images Media WS Other
Headers Cookies Params Response Timings
week11-nas-prg.pptx - PowerPoint
FILE HOME INSERT DESIGN TRANSITIONS ANIMATIONS SLIDE SHOW REVIEW VIEW
Persistent
NPTEL

Registered Users

| usrid | ctime | uname |
|---|---|---|
| 96 | 2020-04-23 15:23:57.0 | xyz1 |
| 42 | 2020-04-19 12:17:48.0 | abcd |
| 35 | 2020-04-18 00:55:32.0 | abc3 |
| 33 | 2020-04-18 00:55:25.0 | abc2 |

```
2 requests    23.72 KB / 2.74 KB transferred    Finish: 32 ms    DOMContentLoaded: 21 ms    load: 29 ms
Request URL: http://localhost:8080/fairsharedb/
Request Method: POST
Remote Address: [::1]:8080
Status Code: 200 OK
Version: HTTP/1.1
```

**Redirect with changed method**

- Oddly enough, a browser can also change the method it uses!
- This is really strange: it is as if a POST, that is, a Write (or change to the backend) can instead be seen as a GET (a Read)!
- But what really happens is as follows
  - Post changes Url content from C1 to C2
  - A subsequent Get will of course get C2, not the old C1, because the old C1 has been *overwritten* by the POST
- Thus, the "strange" behavior make sense
- What we are really saying is: *Overwrite this URL, forget about the old content, and blithely return the new content in its place*

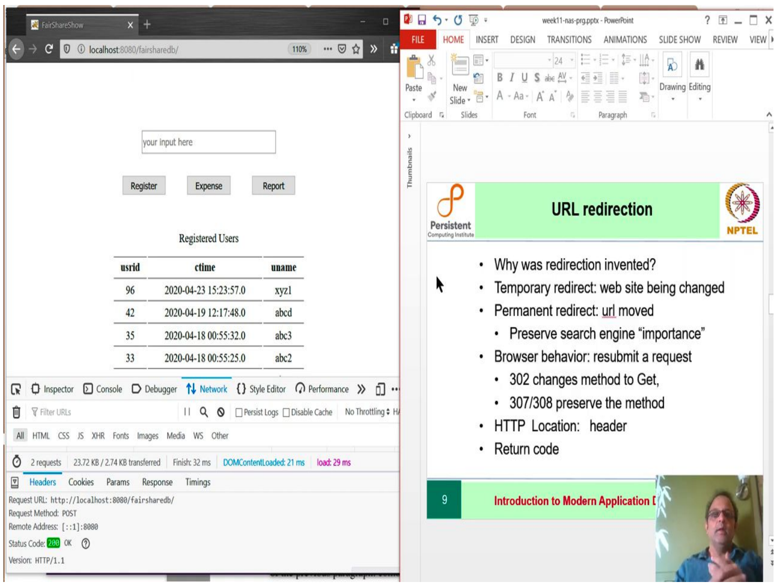
8 Introduction to Modern Application D

**Key observations about post-redirect:**

1. It might sound like redirect should merely change the URL but a browser can also change the method it uses to visit U2.
2. Post changes the content of a URL from content 1 to content 2.
3. Whether content 2 is available via a different URL or whether it is available via the original URL does not matter much in this case. You can in fact send a subsequent get and the only content you could receive is C2 because, after all this is a database and if you said you change a particular record and then it goes ahead and changes.

So once you are inserted xyz1 that is it, whether it is retrieved by get or by some other means like a post etc should not matter and so the strange behavior sort of make sense because what you are saying in a redirect from post to get is : *Override this URL with the post body that I sent, forget about the old contents and just peacefully return new contents in place of the old.*

**(Refer Slide Time: 16:41)**



Registered Users

| usrid | ctime | uname |
|---|---|---|
| 96 | 2020-04-23 15:23:57.0 | xyz1 |
| 42 | 2020-04-19 12:17:48.0 | abcd |
| 35 | 2020-04-18 00:55:32.0 | abc3 |
| 33 | 2020-04-18 00:55:25.0 | abc2 |

```
Request URL: http://localhost:8080/fairsharedb/
Request Method: POST
Remote Address: [::1]:8080
Status Code: 200 OK
Version: HTTP/1.1
```

URL redirection

- Why was redirection invented?
- Temporary redirect: web site being changed
- Permanent redirect: url moved
  - Preserve search engine "importance"
- Browser behavior: resubmit a request
  - 302 changes method to Get,
  - 307/308 preserve the method
- HTTP Location: header
- Return code

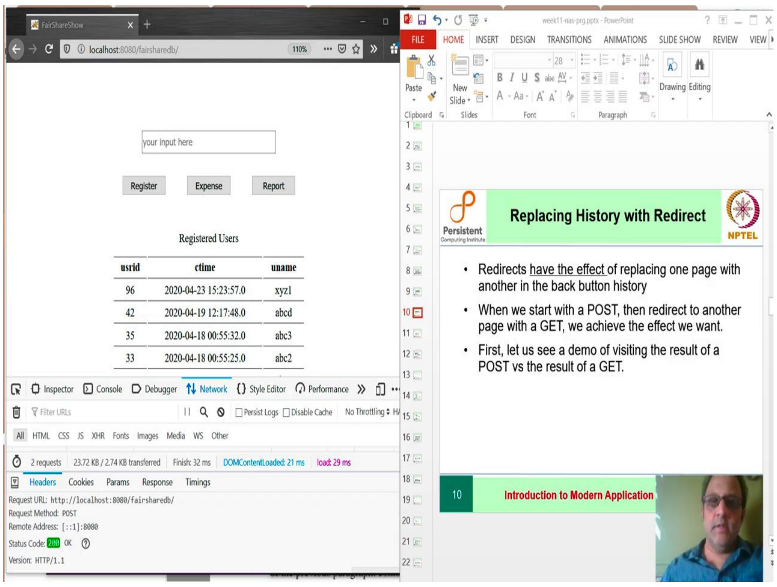
9 Introduction to Modern Application

**Why was redirection invented?**

1. Suppose you visit a URL but the site is undergoing some maintenance then you could tell the user to temporarily go to some other site where maybe old copies are there with the warning that this is an old copy and if you want the latest, to come back sometime later.
2. There was a site at a domain d1 which the owner wants to move to a domain d2. This means permanent redirect which says all the URLs that used to come to domain d1 should now go to domain d2. Search engines try to preserve the ranking of their page in a search engine rank and when you do a permanent redirect the search engine will transfer ranking the URL from the old domain to new one. So, the domains become very valuable in search once they acquire this ranking because of repeated visits. And so how you distinguish between all these is that you send a request.

You get an error code back which says 302, 307 or 308. In the case of 302, the method changes from post to get. 307 and 308 preserve the method, there is the location header which tends to the new URL and the return code which tells whether you should change the method or not change the method.

**(Refer Slide Time: 18:34)**



Registered Users

| usrid | ctime | uname |
|---|---|---|
| 96 | 2020-04-23 15:23:57.0 | xyz1 |
| 42 | 2020-04-19 12:17:48.0 | abcd |
| 35 | 2020-04-18 00:55:32.0 | abc3 |
| 33 | 2020-04-18 00:55:25.0 | abc2 |

```
2 requests   23.72 KB / 2.74 KB transferred   Finish: 32 ms   DOMContentLoaded: 21 ms   load: 29 ms
Request URL: http://localhost:8080/fairsharedb/
Request Method: POST
Remote Address: [::1]:8080
Status Code: 200 OK
Version: HTTP/1.1
```

## Replacing History with Redirect

- Redirects have the effect of replacing one page with another in the back button history
- When we start with a POST, then redirect to another page with a GET, we achieve the effect we want.
- First, let us see a demo of visiting the result of a POST vs the result of a GET.

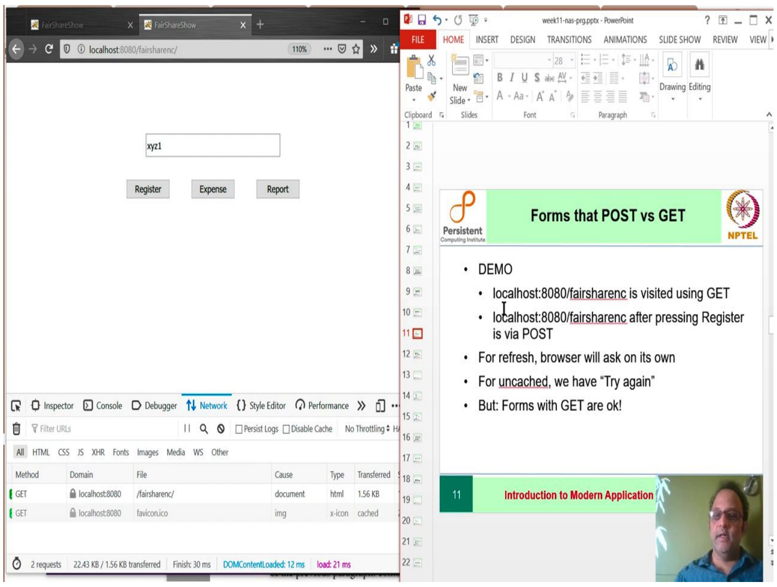
10 Introduction to Modern Application

As we will see, such a redirect has the effect we want, which is of replacing one URL with another in the history of the page. We will start with the post, redirect with get the post is forgotten and the get is remembered.

Let us first look at a demo where we look at the distinction the browser makes, between visiting via get versus via post. So we are going to look at our old non cacheable version.

So, here what we have is the non cacheable version and we visited it with a get of course that is as expected we see a get method. Now, I am going to register a user xyz1 by the way each time I am doing this I am going and cleaning the database so that we can show this behaviour again and again.

**(Refer Slide Time: 19:40)**



| Method | Domain | File | Cause | Type | Transferred |
|---|---|---|---|---|---|
| GET | localhost:8080 | /fairsharenc/ | document | html | 1.56 KB |
| GET | localhost:8080 | favicon.ico | img | x-icon | cached |

```
2 requests   22.43 KB / 1.56 KB transferred   Finish: 30 ms   DOMContentLoaded: 12 ms   load: 21 ms
```

## Forms that POST vs GET

- DEMO
  - localhost:8080/fairsharenc is visited using GET
  - localhost:8080/fairsharenc after pressing Register is via POST
- For refresh, browser will ask on its own
- For uncached, we have "Try again"
- But: Forms with GET are ok!

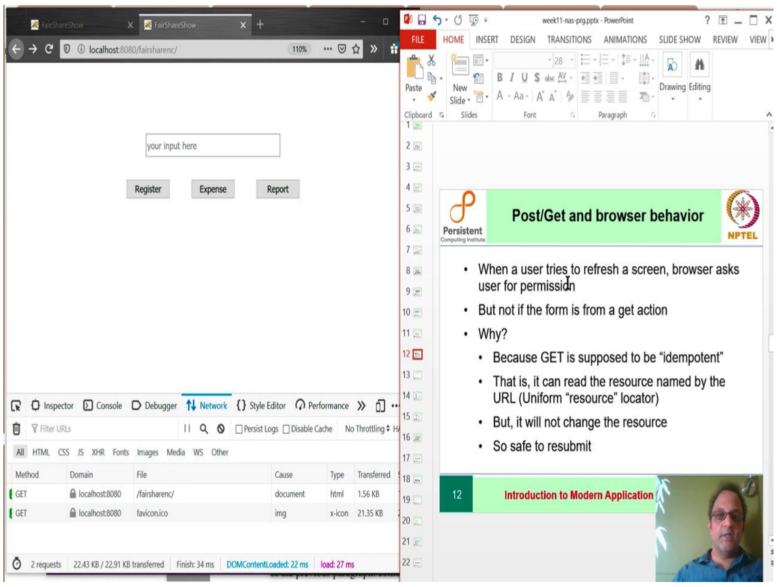
11 Introduction to Modern Application

So, we first visited fair share and see what we get. Now, we register a user and do that via post, this time remember that the post will look at, has a cache-control. So it has no cache, no storage etc. Attempting to revisit this page, the page we are looking at is not going to work but the previous page was retrieved with a get, so we can go back. The browser has no problem whatsoever in going back, but if we go to the cache page, the document is expired.

So for an un-cached you have to explicitly refresh it but you can go back and get this particular form which we retrieved, the get retrieved form with no fuss. And even if you ask to be refreshed, it refreshes it. We can still go ahead and find out that the next page is not visible. The point is that because gets are known to be idempotent because they can be repeated the browser is happy to repeat the gets whenever you ask.

We can even shift reload these things safely again for the same reason that nothing is supposed to change so it is always safe to get it again.

**(Refer Slide Time: 21:21)**



| Method | Domain | File | Cause | Type | Transferred |
|---|---|---|---|---|---|
| GET | localhost:8080 | /fairsharenc/ | document | html | 1.56 KB |
| GET | localhost:8080 | favicon.ico | img | x-icon | 21.35 KB |

2 requests | 22.43 KB / 22.91 KB transferred | Finish: 34 ms | DOMContentLoaded: 22 ms | load: 27 ms

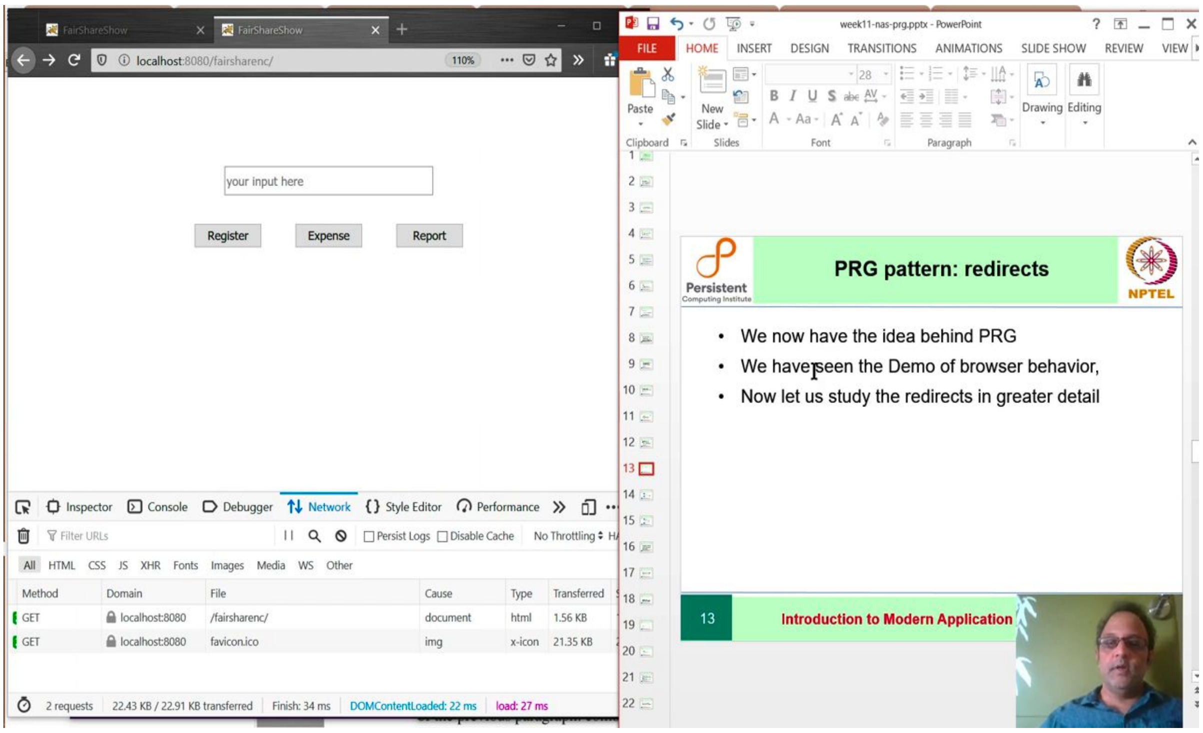
## Post/Get and browser behavior

- When a user tries to refresh a screen, browser asks user for permission
- But not if the form is from a get action
- Why?
  - Because GET is supposed to be "idempotent"
  - That is, it can read the resource named by the URL (Uniform "resource" locator)
  - But, it will not change the resource
  - So safe to resubmit

12 Introduction to Modern Application

When a user tries to refresh the screen the browser will ask the user for permission but not if the form is from get action because get is idempotent submission is always safe. And this is the basic idea behind PRG.

**(Refer Slide Time: 21:37)**



## PRG pattern: redirects

- We now have the idea behind PRG
- We have seen the Demo of browser behavior,
- Now let us study the redirects in greater detail

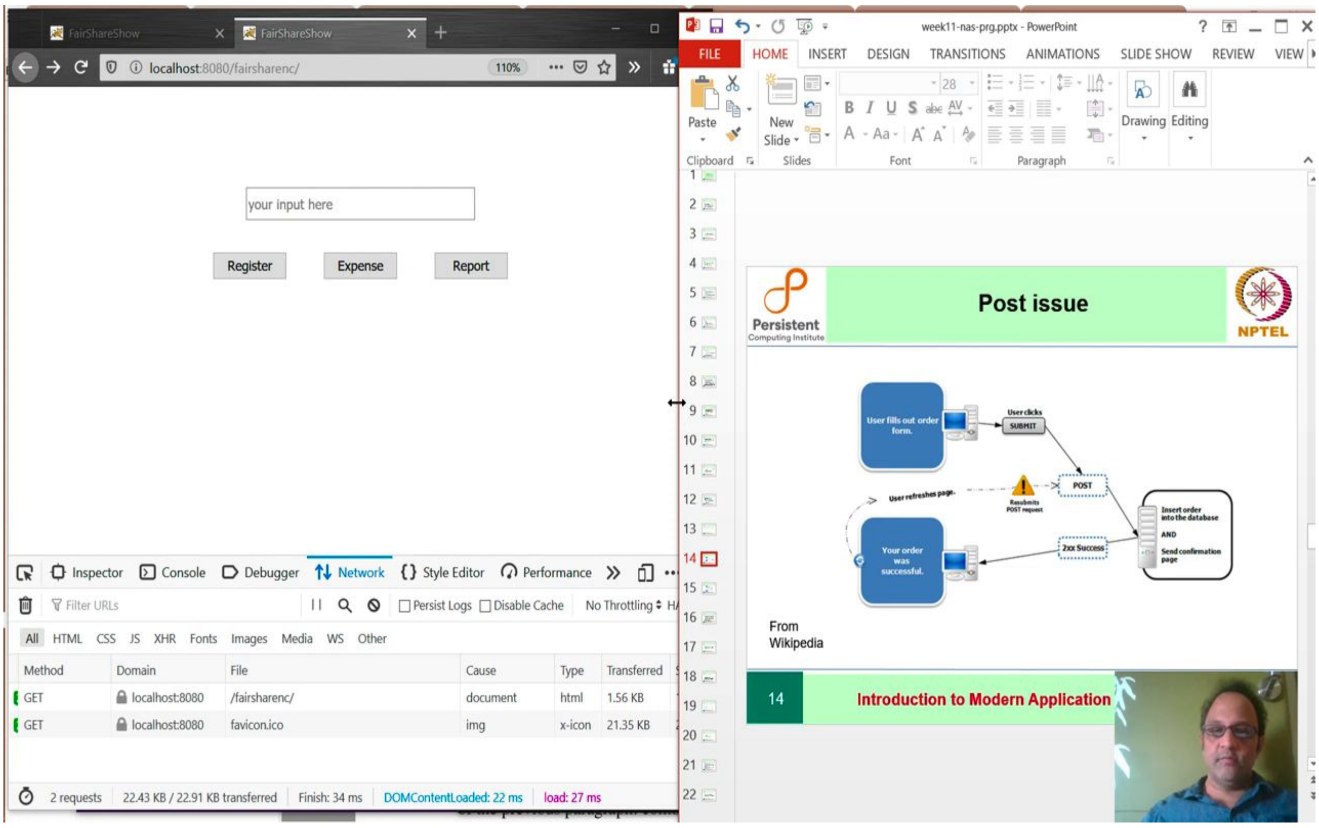
13 Introduction to Modern Application

| Method | Domain | File | Cause | Type | Transferred |
|---|---|---|---|---|---|
| GET | localhost:8080 | /fairsharenc/ | document | html | 1.56 KB |
| GET | localhost:8080 | favicon.ico | img | x-icon | 21.35 KB |

2 requests  22.43 KB / 22.91 KB transferred  Finish: 34 ms  DOMContentLoaded: 22 ms  load: 27 ms

So now that we have seen a demo of how the browser behaves for the two kinds of forms let us look at some pictorial version of what is going on.

**(Refer Slide Time: 22:08)**



## Post issue

From Wikipedia

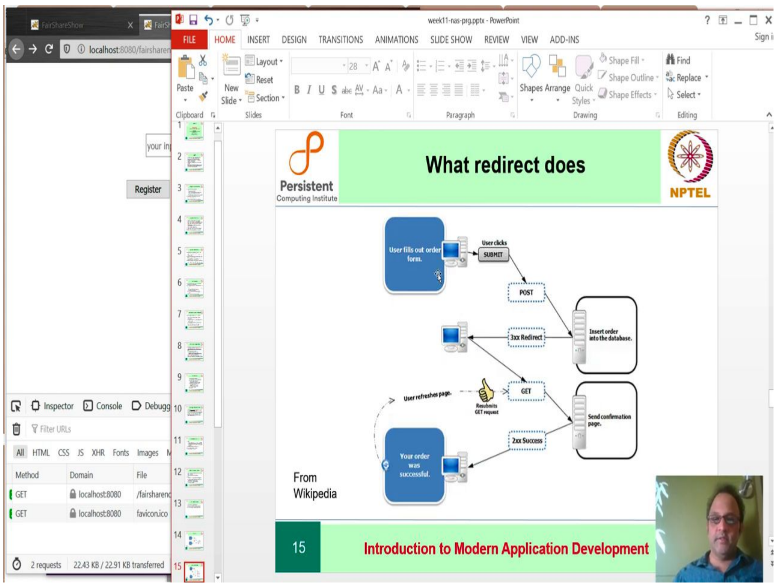
14 Introduction to Modern Application

| Method | Domain | File | Cause | Type | Transferred |
|---|---|---|---|---|---|
| GET | localhost:8080 | /fairsharenc/ | document | html | 1.56 KB |
| GET | localhost:8080 | favicon.ico | img | x-icon | 21.35 KB |

2 requests  22.43 KB / 22.91 KB transferred  Finish: 34 ms  DOMContentLoaded: 22 ms  load: 27 ms

The first case: User fills up form, clicks submit, does a post. Insert order into the database and send a confirmation page. This is what we have been doing when we did, do post redirect and get. Now if you hit refresh it resubmits the post.

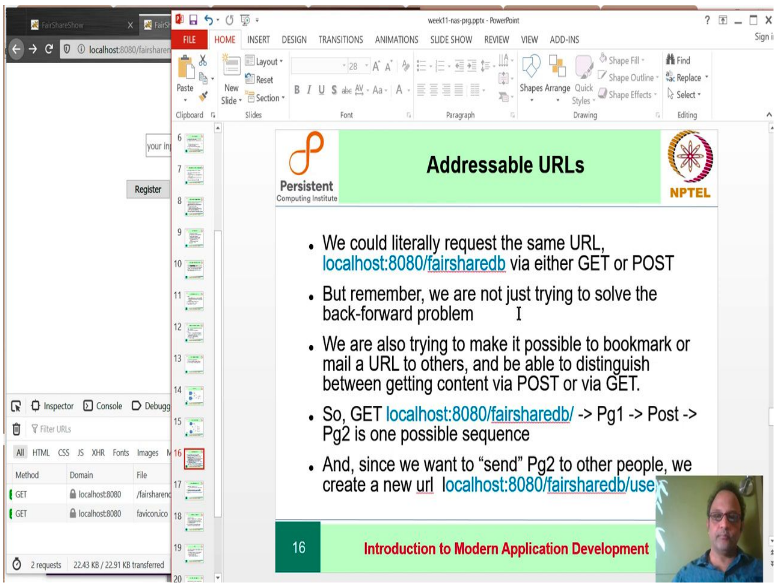
**(Refer Slide Time: 22:35)**



What redirect does

User fills out order form.
User clicks
SUBMIT
POST
Insert order into the database.
3xx Redirect
GET
User refreshes page.
Resubmits GET request
Send confirmation page.
2xx Success
Your order was successful.
From Wikipedia
15
Introduction to Modern Application Development

Instead what we would like to happen is this, user fills out a form, clicks submit, post, insert order into the database. Then its ends with what is called 302 redirect. We will see an example shortly. At this point the users machine remembers this one, originates a new request that is the get, then you send the confirmation page you are told your order is successful and refresh it otherwise request it you get the same thing back. **(Refer Slide Time: 23:08)**



Addressable URLs

- We could literally request the same URL, localhost:8080/fairsharedb via either GET or POST
- But remember, we are not just trying to solve the back-forward problem
- We are also trying to make it possible to bookmark or mail a URL to others, and be able to distinguish between getting content via POST or via GET.
- So, GET localhost:8080/fairsharedb/ -> Pg1 -> Post -> Pg2 is one possible sequence
- And, since we want to "send" Pg2 to other people, we create a new url localhost:8080/fairsharedb/use

16 Introduction to Modern Application Development

**What URL should the get ask to retrieve?**

It does not say after the 3xx redirect what URL you are supposed to retrieve just say redirection to any URL is possible. And we could literally request the same URL using the method get versus the method post. But nearly requesting the URL in two different ways is not our only goal remember.

Our other goal is to deal with creation of addressable URLs.

We are not trying to solve just back and forward problem. We are also trying to make it possible to bookmark or mail a URL to others and to be able to distinguish between getting the content via post versus via get.

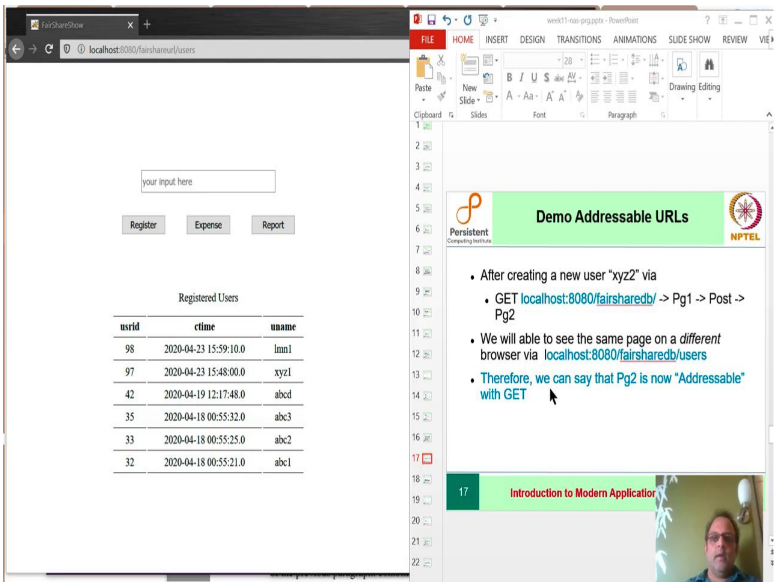
Let us begin by visiting fair share URL with the get.
We are following the first page get localhost fair share URL.
We arrived at PG1 and now we are going to post a new user lets say uvw1 and we will register this user.
.But one difference you will notice, the URL has changed to say to add a new URI or a new URI component called users.
First time when we press the register button, we actually did it with a post. So the post is a familiar thing by now. We have Post and if we do the edit resend you can see the body this is the post body lmn1 and register.
Header has the new location, should be fair share URL dot user.

This is how the server tells the browser that it should redirect and at that point the browser automatically issued a get request. And this get request looks like a normal get request. But if you look at the history here, you have only two pages the post in the middle is missing and if we go back and if we go forward you see a get and you see a lmn1 and the only call that was made was to this. You know favicon which itself was retrieved via a cache.

So our history became only two components even though there were 3 requests and we are able to see the same content as before this lmn1 but via a different URL. And in fact there is more than that I can take this I can take the user's URL I have taken the user's URL and I have a fresh browser instance which is different from the one I normally use, I hit this and I can see the same data again. So from 2 completely different browsers I am able to retrieve the same data.

And this way the new application state, the one which contains the user lmn1 which is the latest application state, has become visible through a new URL. It says here, let us kill this. The get localhost PG etcetera was one possible sequence and now we have created a new URL and this URL can be used anywhere, you can send it, you can retrieve it whatever you want to do. It has enough information via these added users to be able to show us the page with registered users.

**(Refer Slide Time: 28:50)**


FairShareShow
localhost:8080/fairshareurl/users
your input here
Register
Expense
Report
week11-nas-prg.pptx - PowerPoint
Persistent
NPTEL

Registered Users

| usrid | ctime | uname |
|---|---|---|
| 98 | 2020-04-23 15:59:10.0 | lmn1 |
| 97 | 2020-04-23 15:48:00.0 | xyz1 |
| 42 | 2020-04-19 12:17:48.0 | abcd |
| 35 | 2020-04-18 00:55:32.0 | abc3 |
| 33 | 2020-04-18 00:55:25.0 | abc2 |
| 32 | 2020-04-18 00:55:21.0 | abc1 |

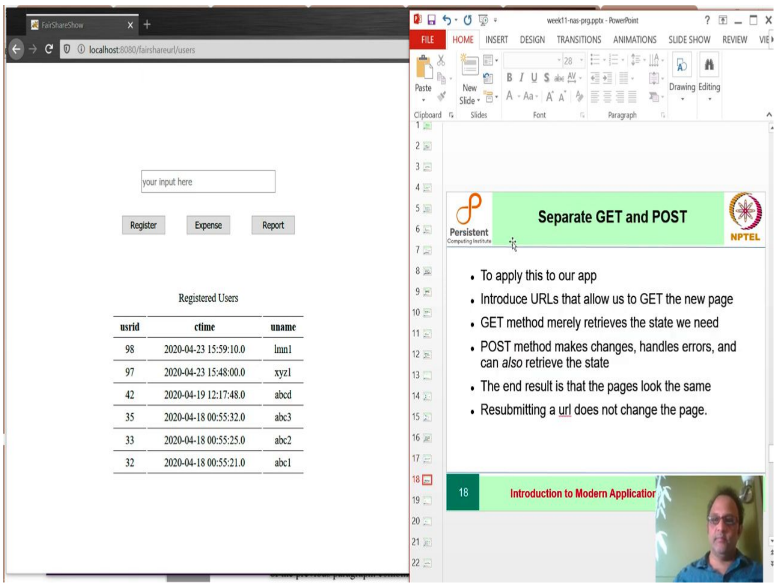
Demo Addressable URLs

- After creating a new user "xyz2" via
  - GET localhost:8080/fairsharedb/ -> Pg1 -> Post -> Pg2
- We will able to see the same page on a *different* browser via localhost:8080/fairsharedb/users
- Therefore, we can say that Pg2 is now "Addressable" with GET

17 Introduction to Modern Application

As recorded in the slide here, the demo showed us that after creating the new user, I have created a user called lmn1 via get fairshare URL I was able to visit that same page on a different browser via a new URL. And so we can say that pg2 is now addressable with get. That is what addressable URLs mean.

**(Refer Slide Time: 29:25)**


FairShareShow
localhost:8080/fairshareurl/users
your input here
Register
Expense
Report
week11-nas-prg.pptx - PowerPoint
Persistent
NPTEL

Registered Users

| usrid | ctime | uname |
|---|---|---|
| 98 | 2020-04-23 15:59:10.0 | lmn1 |
| 97 | 2020-04-23 15:48:00.0 | xyz1 |
| 42 | 2020-04-19 12:17:48.0 | abcd |
| 35 | 2020-04-18 00:55:32.0 | abc3 |
| 33 | 2020-04-18 00:55:25.0 | abc2 |
| 32 | 2020-04-18 00:55:21.0 | abc1 |

Separate GET and POST

- To apply this to our app
- Introduce URLs that allow us to GET the new page
- GET method merely retrieves the state we need
- POST method makes changes, handles errors, and can *also* retrieve the state
- The end result is that the pages look the same
- Resubmitting a url does not change the page.

18 Introduction to Modern Application

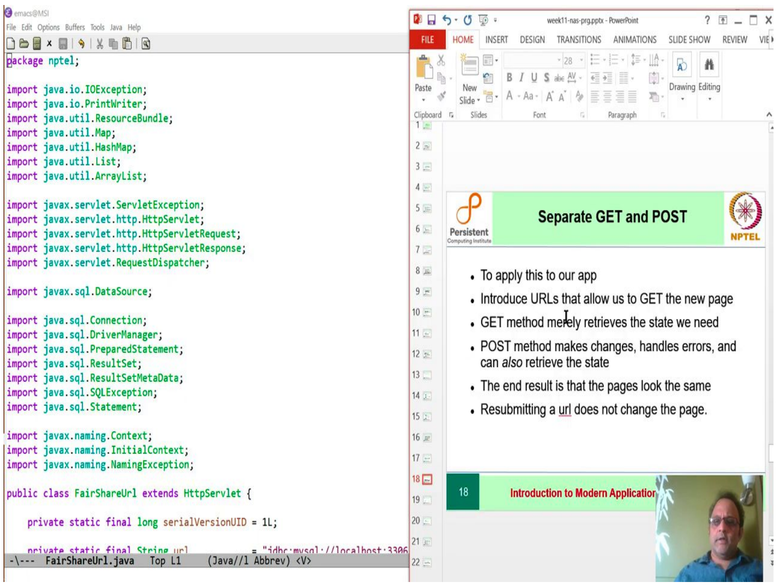
How do we apply this to our app as a whole?

Let us take a look at the code for fair share URL. Here we are, let us look at this code and compare it with the code for fairshare DB, here is the code for fair share URL and the code for fairshare DB.

We first introduce URLs that allow us to get the new page and this is true not only for the user URL right it is also true for lmn1 111.

If I say expense you will see the same sequence there is post there is get and there is a expenses URL and indeed I can copy it, go to a different browser hit return and be able to see the same page here again.

**(Refer Slide Time: 31:13)**



```
package nptel;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.ResourceBundle;
import java.util.Map;
import java.util.HashMap;
import java.util.List;
import java.util.ArrayList;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.RequestDispatcher;

import javax.sql.DataSource;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class FairShareUrl extends HttpServlet {

    private static final long serialVersionUID = 1L;

    private static final String url             = "jdbc:mysql://localhost:3306
-\---  FairShareUrl.java    Top L1     (Java//l Abbrev) <V>
```

Separate GET and POST

- To apply this to our app
- Introduce URLs that allow us to GET the new page
- GET method merely retrieves the state we need
- POST method makes changes, handles errors, and can *also* retrieve the state
- The end result is that the pages look the same
- Resubmitting a url does not change the page.

emacs@MSI
File Edit Options Buffers Tools Java Help
week11-nas-prg.pptx - PowerPoint
FILE HOME INSERT DESIGN TRANSITIONS ANIMATIONS SLIDE SHOW REVIEW
Paste
New Slide
Drawing Editing
Clipboard Slides Font Paragraph
Persistent
NPTEL
18
Introduction to Modern Application

We were able to introduce new URLs for each state even for report and the same sequence post redirect get and a new URL.

And as usual there is a location header which says where the post should be redirected to and the get which get this URL.

To implement it, you have to have URLs like report, user etc which generate the same page by getting the same data, but they do not take input which they are going to apply to the database.

The context path is part and the extension is this report URL and it checks that URI is either this or this users expenses report. This business of looking at the shape of the URL and transmitting it into different places, whether it is done in the code like this or whether that is done in for example methods like web in WEB.XML. So, you can do this kind of mapping either in an XML file like that or you can do it directly in the code differently. So, the web dot xml for fair share URL is not different, look at fair share URL webinf, web dot xml here what it does is, you take all these URL pattern slash report user expenses and channel them all into the single app.

App itself receives all the things and starts analysing it. In production code there are different reasons to put this thing in different places, but for sure we can rely on simple routing methods as shown in the code. And here what we are doing is if it is either the none URL, users URL or expenses URL etc what we do is, if it is users then we get all registered users. If it is expenses all expenses, if it is expense report etc. And we are returning actually the same JSP filled in the same way with the same data but no content in the input field.

In some cases you can give prefilled input fields etc for our application we do not need it so we have not done that. Let us take a look at the difference between the post methods in these two cases. So, here is post from fair share URL and here is post from fair share DB. As before we analyse buttons the analysis of the buttons is no difference in the two cases. This code has some cache-control stuff, but it does not really matter at this moment I really ignore that just focus on the part that matters.

In fact I will for the purpose of this comparison I just deleted it, the comparison and there we are and here we go and so try connect etc. The distinction is that in the case of fair share DB we had no gets there we had posts. So we just register the user and get all registered users. Instead here we do something different. We say register the user and if you are successful then you send the redirect.

Now we are not directly calling all registered users or anything post in the success case. In the success case we go back to the browser, the browser sends the request to the app we are looking at again and then it is processed as a get request. One question is why not do it in the browser and save the round trip. Well, we can analyse this further but for you, let us just say that for whatever reason the protocol has turned out this way that doing it via 2 requests like this makes more sense compared to the other alternatives.

What happened though is that the post is completed and its result is thrown away in the browser. That is why there is no point in getting all the data even if you add it the way things stand. You could have designed a different protocol other than http in which we could have saved the round trip but that is the protocol that we have. Therefore in this protocol all the post does is change the database send to redirect and then the effect of the post is retrieved via get, this is important.

The effect of the post is retrieved via get, let us, important part, effect of post is retrieved via get that is the flow that we have done which gives you smoothness in the browser interaction. What should we do in the failure case? Now in the failure case in our app what we are done is? We followed the old way that is just show the error, right and show the data and the reason we can do this is because even if you have duplicate submit because of user being here nothing will change.

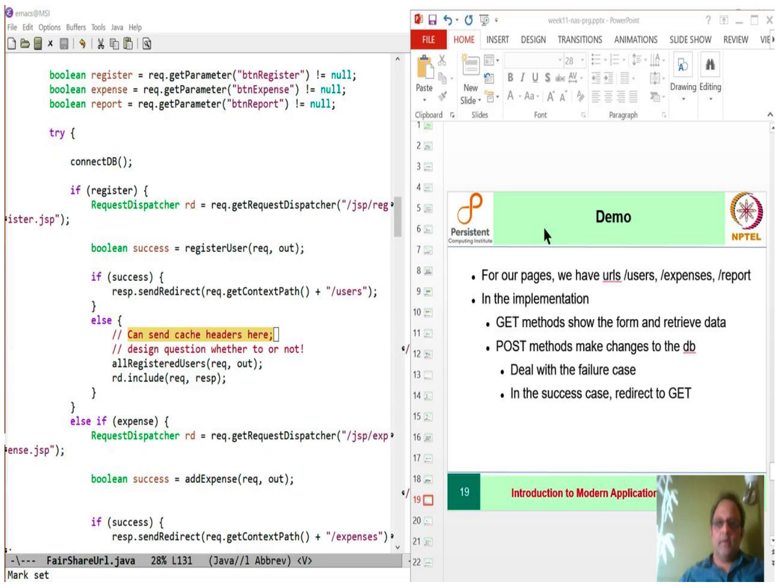
We will still get; our database would not be harmed in any case. So, let us see what that looks like. I am going to try to register a user like this lmn1 as expected. I see this button here and we have not done a redirect, this is just a straightforward post. Now if you put in a proper registration, then in the next request has a post get with the post getting for the button. As for the history is concerned there were these 2 URLs, one which has failed and if you try to go back to the URL, you are simply able to go back because even if you submit something nothing will go wrong.

So, at least that is the design of this app, actually the thing is that let us see what happens if we hit expense. If we hit expense, we get an exception because the way we have done that validation job is incomplete in our toy application. Here 2 things were expected, only one thing happened. And so we are ok that our database does not actually change. So, hardly enough in this app this kind of half hearted thing, where we get these failures is tolerable but that is a very unusual thing.

And in a real app, these cases will not make any sense. In real app we would do something about going back. We might say that for example we might turn this button un submittable or have a warning that says look you are looking at old state or whatever makes sense for **business logic** of your application that is what matters. And the pattern here that is redirects in the case of success and return as usual in the previous case.

You could do It differently, you could send these cache invalidation headers and then the page would not show up just show up. That is ok to again up to you for your particular app what you do. And thus we have created shareable URLs and we have taken care of application state sharing.

**(Refer Slide Time: 41:20)**



```
        boolean register = req.getParameter("btnRegister") != null;
        boolean expense = req.getParameter("btnExpense") != null;
        boolean report = req.getParameter("btnReport") != null;

        try {

            connectDB();

            if (register) {
                RequestDispatcher rd = req.getRequestDispatcher("/jsp/reg
ister.jsp");

                boolean success = registerUser(req, out);

                if (success) {
                    resp.sendRedirect(req.getContextPath() + "/users");
                }
                else {
                    // Can send cache headers here;
                    // design question whether to or not!
                    allRegisteredUsers(req, out);
                    rd.include(req, resp);
                }
            }
            else if (expense) {
                RequestDispatcher rd = req.getRequestDispatcher("/jsp/exp
ense.jsp");

                boolean success = addExpense(req, out);


                if (success) {
                    resp.sendRedirect(req.getContextPath() + "/expenses")
-\---  FairShareUrl.java   28% L131   (Java//l Abbrev) <V>
Mark set
```

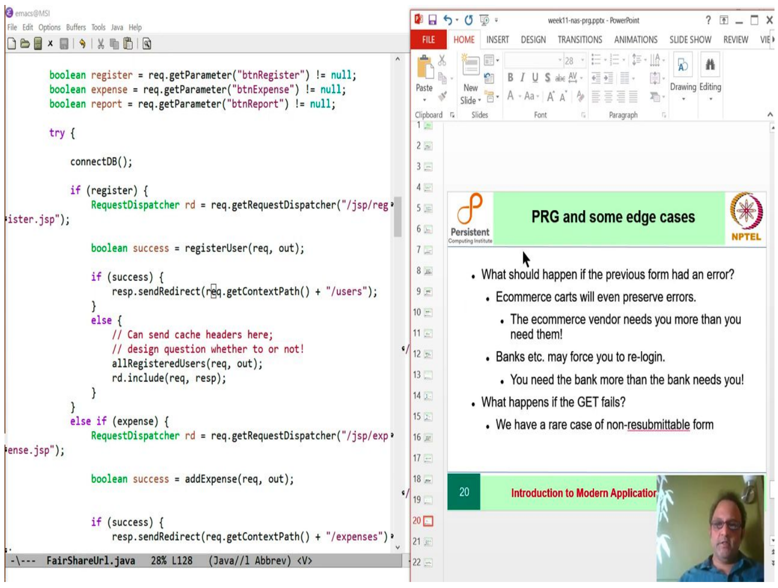
## Demo

- For our pages, we have urls /users, /expenses, /report
- In the implementation
  - GET methods show the form and retrieve data
  - POST methods make changes to the db
    - Deal with the failure case
    - In the success case, redirect to GET

19 Introduction to Modern Application

So a summary of our demo is that for our pages we have URL user expenses report. In the implementation we have get methods which show just the form and retrieve the data. Post method make changes, deal with the failure case in the success redirect.

**(Refer Slide Time: 41:47)**



```
        boolean register = req.getParameter("btnRegister") != null;
        boolean expense = req.getParameter("btnExpense") != null;
        boolean report = req.getParameter("btnReport") != null;

        try {

            connectDB();

            if (register) {
                RequestDispatcher rd = req.getRequestDispatcher("/jsp/reg
ister.jsp");

                boolean success = registerUser(req, out);

                if (success) {
                    resp.sendRedirect(req.getContextPath() + "/users");
                }
                else {
                    // Can send cache headers here;
                    // design question whether to or not!
                    allRegisteredUsers(req, out);
                    rd.include(req, resp);
                }
            }
            else if (expense) {
                RequestDispatcher rd = req.getRequestDispatcher("/jsp/exp
ense.jsp");

                boolean success = addExpense(req, out);


                if (success) {
                    resp.sendRedirect(req.getContextPath() + "/expenses")
-\---  FairShareUrl.java   28% L128   (Java//l Abbrev) <V>
```

## PRG and some edge cases

- What should happen if the previous form had an error?
  - Ecommerce carts will even preserve errors.
    - The ecommerce vendor needs you more than you need them!
  - Banks etc. may force you to re-login.
    - You need the bank more than the bank needs you!
- What happens if the GET fails?
  - We have a rare case of non-resubmittable form

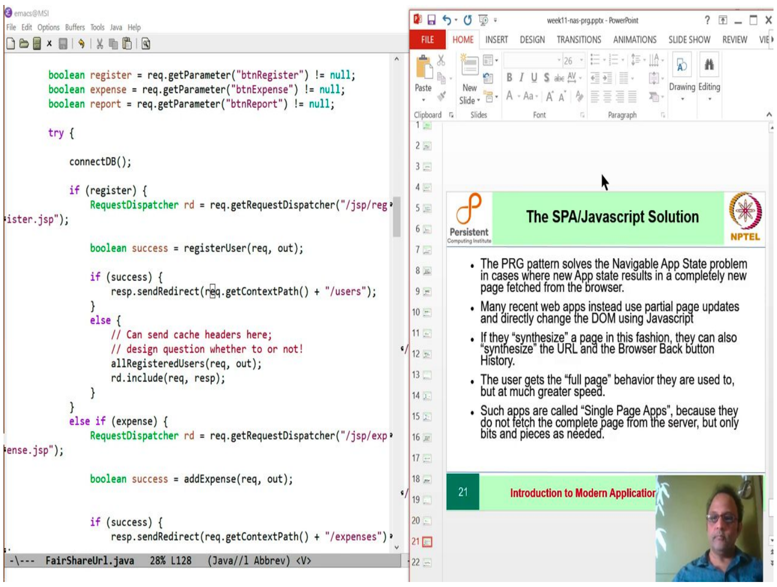
20 Introduction to Modern Application

There are few cases one which we looked at, is what to do if there is an error. If it is an ecommerce cart you will actually find it is useful to preserve the error that the user goes back and change it. For that we cannot rely on the local cached form. We actually have to make sure that the form with the error is stored in the database and retrieved from there. So, we might make the form uncacheable even with a post redirect in spite of errors such that when the browser goes back it can retrieve from the database.

This is because the E-Commerce vendor needs you more than you need that you can always go to some other vendor. But the bank will force you to re login. It is sometimes for the case of security at other times because you need the bank more than need you. One other failure case which we did not look at is the case where in this redirect the post goes through successfully but it is actually the get which fails.

If that happens we are now stuck in strange state and that would be would consider it reasonably rare. So, will have to figure out how to deal with that for example we may have rare case of non resubmittable stuck state in which you tried the get but the get did not work. And so you encounter an ordinary browser failure of being unable to retrieve a page. In which case you can let the user submit again actually so not non resubmittable is a type of; it is a rare case of resubmittable form which because it is retrieved by get can be re-fetched. So that is the overall flow of the post redirect get.

**(Refer Slide Time: 43:55)**



```
        boolean register = req.getParameter("btnRegister") != null;
        boolean expense = req.getParameter("btnExpense") != null;
        boolean report = req.getParameter("btnReport") != null;

        try {

            connectDB();

            if (register) {
                RequestDispatcher rd = req.getRequestDispatcher("/jsp/reg
ister.jsp");

                boolean success = registerUser(req, out);

                if (success) {
                    resp.sendRedirect(req.getContextPath() + "/users");
                }
                else {
                    // Can send cache headers here;
                    // design question whether to or not!
                    allRegisteredUsers(req, out);
                    rd.include(req, resp);
                }
            }
            else if (expense) {
                RequestDispatcher rd = req.getRequestDispatcher("/jsp/exp
ense.jsp");

                boolean success = addExpense(req, out);


                if (success) {
                    resp.sendRedirect(req.getContextPath() + "/expenses")
-\---  FairShareUrl.java   28% L128   (Java//l Abbrev) <V>
```

## The SPA/Javascript Solution

- The PRG pattern solves the Navigable App State problem in cases where new App state results in a completely new page fetched from the browser.
- Many recent web apps instead use partial page updates and directly change the DOM using Javascript
- If they "synthesize" a page in this fashion, they can also "synthesize" the URL and the Browser Back button History.
- The user gets the "full page" behavior they are used to, but at much greater speed.
- Such apps are called "Single Page Apps", because they do not fetch the complete page from the server, but only bits and pieces as needed.

21 Introduction to Modern Applicatior

One last thing before we are done with this session, just a few words of about a different solution for this state synchronization problem or the application state problem. This navigable app state problem in cases where the new app state that gives your completely fresh page we use the post redirect get pattern. But *many recent web apps instead use partial page updates*. So instead of changing the whole page when you have changed, say, suppose you add a new user.

Then you just want to add in this case if I add a new user lmn1 my entire page should not change I should just add a new row over here that would actually be ideal from efficiency and responsiveness perspective. So you can do partial page updates and directly change the

DOM. DOM is this one right you remember from before so which is documentary. So you can directly edit the documentary using JavaScript and just insert one row by fetching the content from the database.

So such an app; if the app synthesizes a page in this fashion, then in order to make such addressable URLs and their behaviour look the same as a full page refresh. They can also synthesize the URL and the history and in fact search history API is actually available in JavaScript these days. So, the user gets something like a full page behaviour, but at much greater speed and such apps will be called single page apps.

But as far as the user experience is concerned they will try to preserve as much of this addressability is necessary for their application. So, with that we have done one of the more perhaps the trickiest parts of the new kind of design that the web requires. And next session we will add cookies and session and talk about one thing which we have not seen before is form resubmission by users.

Resubmit tokens, separating users using cookies and so far so forth. First we will deal with separating the users using cookies and then go on to the form resubmission issue.