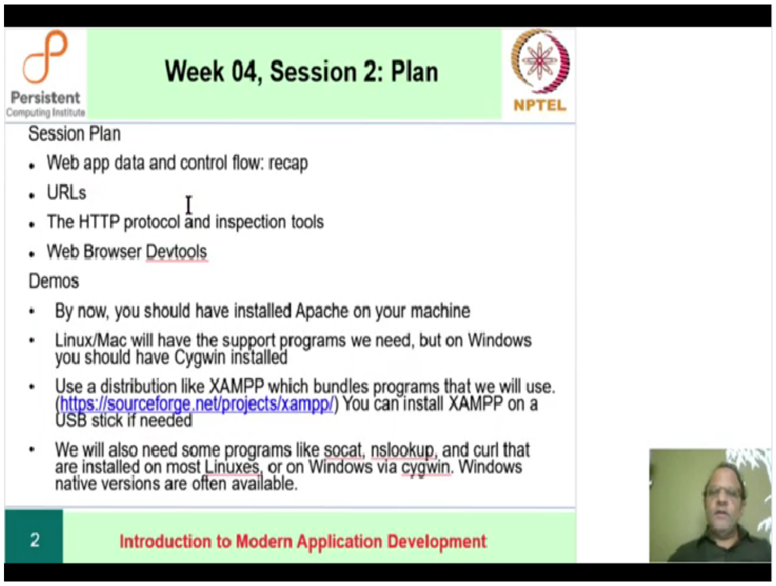
Introduction to Modern Application Development

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

Lecture-14
Session 2-Part 1

Hello, welcome to modern application development. In the previous sessions, we have discussed CLI, and GUI apps.

(Refer Slide Time: 00:11)



Week 04, Session 2: Plan

Session Plan

- Web app data and control flow: recap
- URLs
- The HTTP protocol and inspection tools
- Web Browser Devtools

Demos

- By now, you should have installed Apache on your machine
- Linux/Mac will have the support programs we need, but on Windows you should have Cygwin installed
- Use a distribution like XAMPP which bundles programs that we will use. (https://sourceforge.net/projects/xampp/) You can install XAMPP on a USB stick if needed
- We will also need some programs like socat, nslookup, and curl that are installed on most Linuxes, or on Windows via cygwin. Windows native versions are often available.

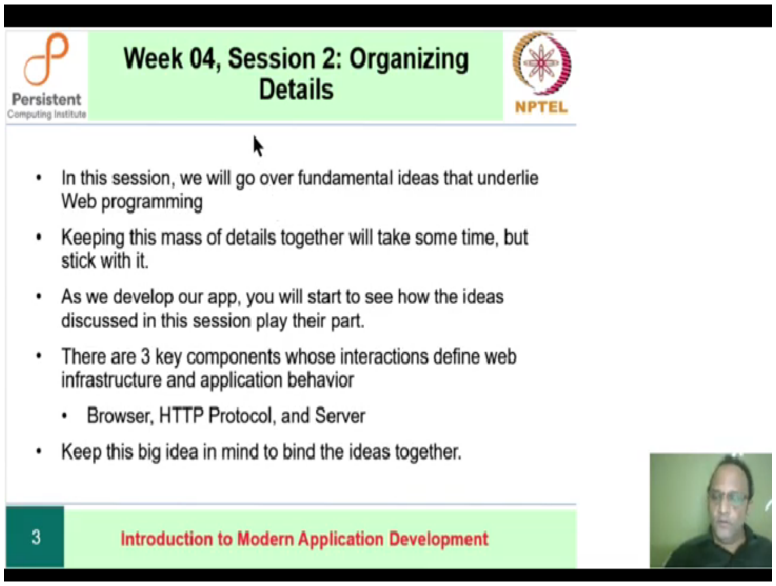
2 Introduction to Modern Application Development

And we have seen how to evolve those apps into a web app by using a CGI program. To begin with, we will look at how we have done the evolution and get a clear idea of the changes in the control flow and the internal architecture of the applications. After that, we will move on to start thinking of URLs as programmers rather than as users. Although, all of us are familiar with URLs as objects that be used daily for various web applications.

We have usually thought of it as users rather than as programmers and as you will see, quite a few technical details come up when you think of them as objects relevant for programming. Our next step will be to look at the HTTP protocol and study the tools that allow us to inspect what is going on when HTTP is used. Lastly, we will study web browser Devtools which are fundamental tools for any web programmer.

During this session, we will be doing several demos and you should have the following programs ready: you can use a distribution like XAMPP which has things like Apache and MySQL and so forth, which you may have already installed for the previous session. If not, you can try doing that now. The other programs we need will be those like **SOCAT, NSLOOKUP** and **cURL.** These are available on most Linux or on Windows, certainly via Cygwin and in some cases native versions might be available.

(Refer Slide Time: 01:54)



Week 04, Session 2: Organizing Details

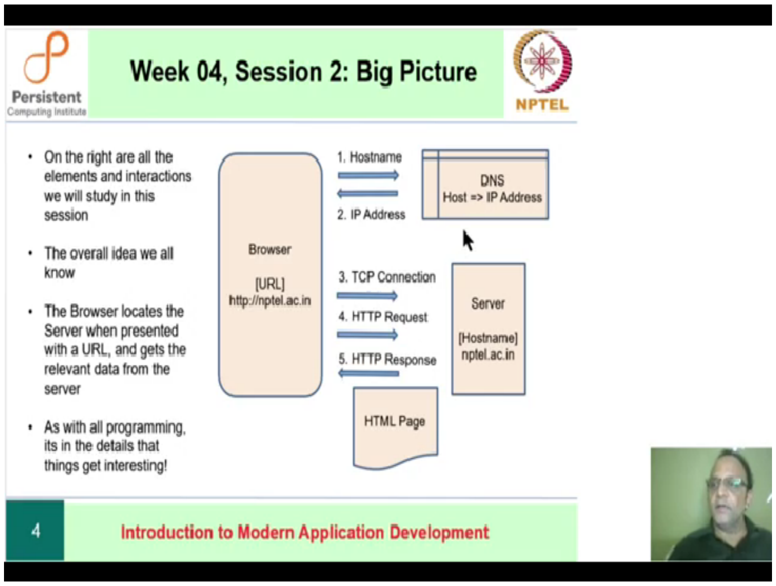
- In this session, we will go over fundamental ideas that underlie Web programming
- Keeping this mass of details together will take some time, but stick with it.
- As we develop our app, you will start to see how the ideas discussed in this session play their part.
- There are 3 key components whose interactions define web infrastructure and application behavior
  - Browser, HTTP Protocol, and Server
- Keep this big idea in mind to bind the ideas together.

3 Introduction to Modern Application Development

In this session, we are going to go over fundamental ideas that underlie web program. These ideas constitute of a mass of details and it will take some time for you to get these ideas organized together, but stick with it, because these ideas underlie much of what is to come. As we develop our app, you will start to see how these ideas play their part in different pieces of the application. To help you organize this, I want to emphasize the big picture view which you should keep in mind, and then slot these ideas in different parts of the big picture as we go ahead.

The main theme is that there are three key components whose interactions define web infrastructure, and as a result, define application behavior as well. These three components are: the browser, the HTTP protocol, and the server. Let us see how these three things work with each other.

(Refer Slide Time: 03:01)



Week 04, Session 2: Big Picture

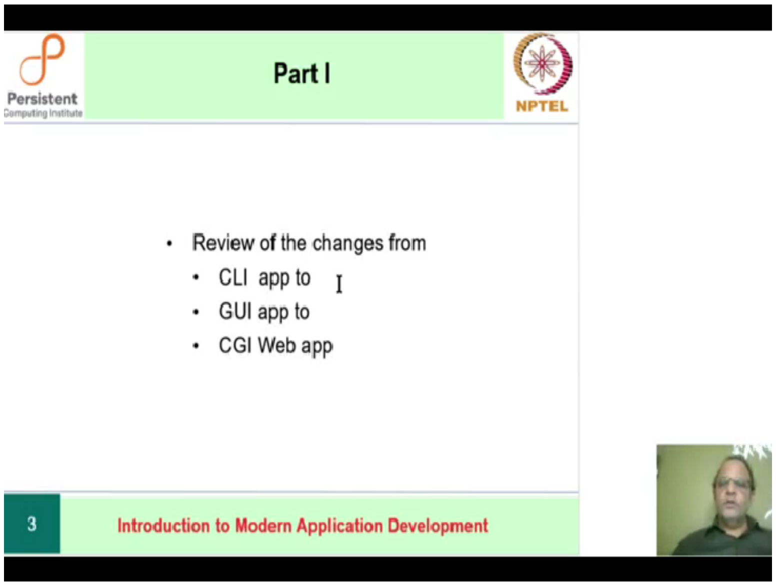
- On the right are all the elements and interactions we will study in this session
- The overall idea we all know
- The Browser locates the Server when presented with a URL, and gets the relevant data from the server
- As with all programming, its in the details that things get interesting!

So, on the right are all the elements and the interactions that we are going to study in this session. We all know what the overall idea is:

- We have a URL, such as nptel.ac.in and it is given to the browser as URL
- The browser somehow figures out where the server is located. This is where the browser interacts with the domain name system which converts the host name into an IP address.
- Once it has the IP address, it establishes a TCP connection and it runs an HTTP request. So, something unusual is happening here, the connection is in one protocol and a different protocol runs on top. What exactly happens here is what we are going to figure out in this session. So that, by the end of this session, this diagram will be meaningful to you.
- After that we have HTTP request and response and there are quite a few intricate details that you have to get right in order for these request and response behaviors to help put

together the web in a scalable way. As with all programming, although the big picture is reasonably clear cut, it is in the details that things get interesting.

(Refer Slide Time: 04:35)



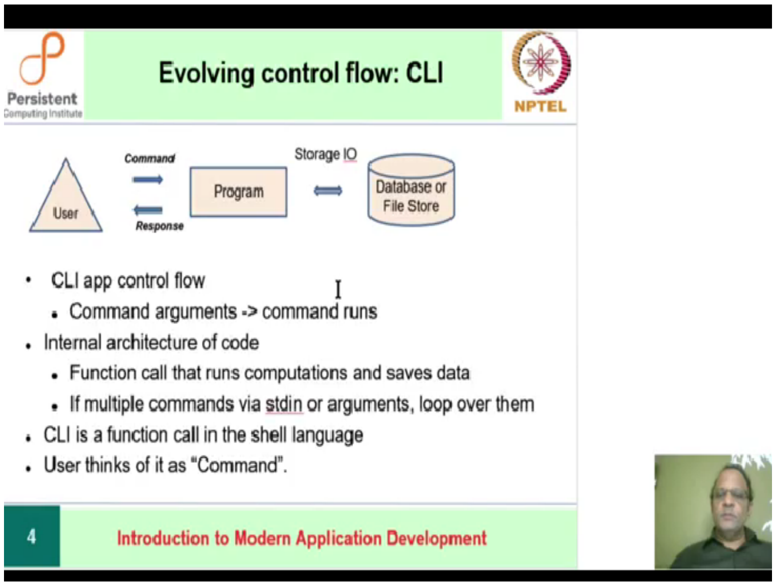
Part I

- Review of the changes from
  - CLI app to
  - GUI app to
  - CGI Web app

3 Introduction to Modern Application Development

In the first part of this session, we will review how we changed the CLI app to the GUI app to the CGI web app, looking at the details of what changes we made to the internal architecture as well as the control flow between different parts of the system.

(Refer Slide Time: 04:59)



Evolving control flow: CLI

- CLI app control flow
  - Command arguments -> command runs
- Internal architecture of code
  - Function call that runs computations and saves data
  - If multiple commands via stdin or arguments, loop over them
- CLI is a function call in the shell language
- User thinks of it as "Command".

4 Introduction to Modern Application Development

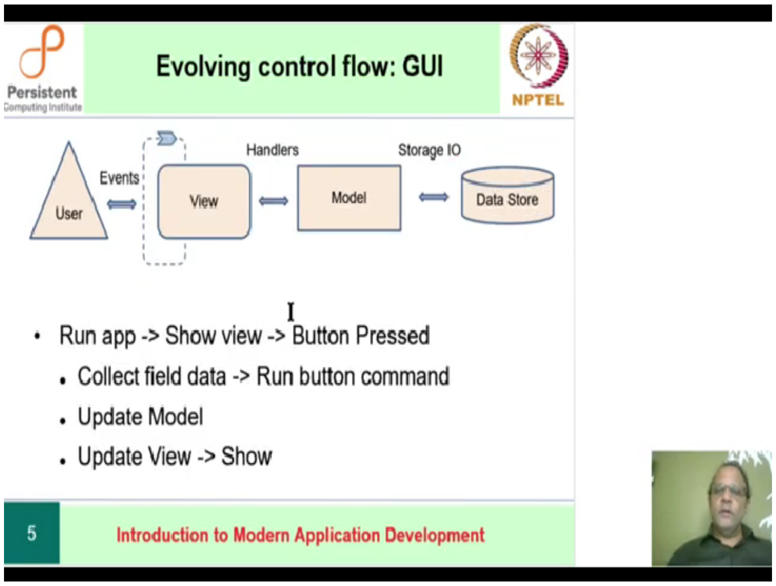
So, let us begin with the standard CLI app. Here, we have three parts of the CLI:

- User
- Program
- Database store

And as we all know, it is a pretty straightforward thing the user puts in command, arguments or standard input, the program does its computation, comes up with the response and prints it out. It feels exactly like a function call to us and indeed the internal architecture of the code is also very similar.

It is simply a series of function calls that run computations and save data. At most if there are multiple commands, then we will loop over them. Also, we notice that as far as the shell language is concerned, the CLI appears exactly like a function call. We programmers can think of it as a function call, but normal users will think of it as a command.

(Refer Slide Time: 06:01)



Evolving control flow: GUI

- Run app -> Show view -> Button Pressed
  - Collect field data -> Run button command
  - Update Model
  - Update View -> Show

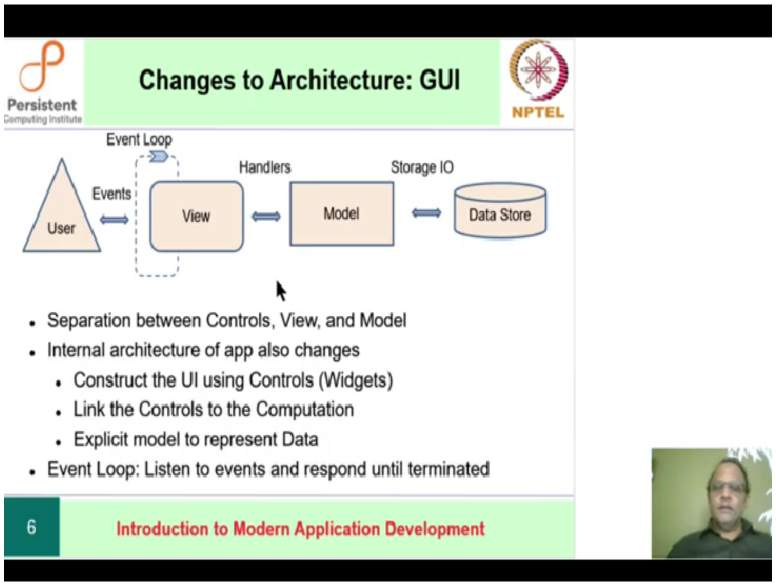
5

Introduction to Modern Application Development

The next step in our discussion is the GUI program. In the GUI program, we made quite a few changes. In particular, the program had to be split into a view part, which shows you which part the user sees, and the model where computation is happening. These two parts are linked by handlers which, in most cases as we have seen, are simple references that tell parts of the view which parts of the model to change.

The control flow here, as we have seen, is should now be familiar to us. When you run the app, the app shows the view; there is an event loop, which looks at button presses then you collect the field data and run whatever command the button has asked you to then you update the model and then finally, update the view and show whatever the results might be.

(Refer Slide Time: 06:58)



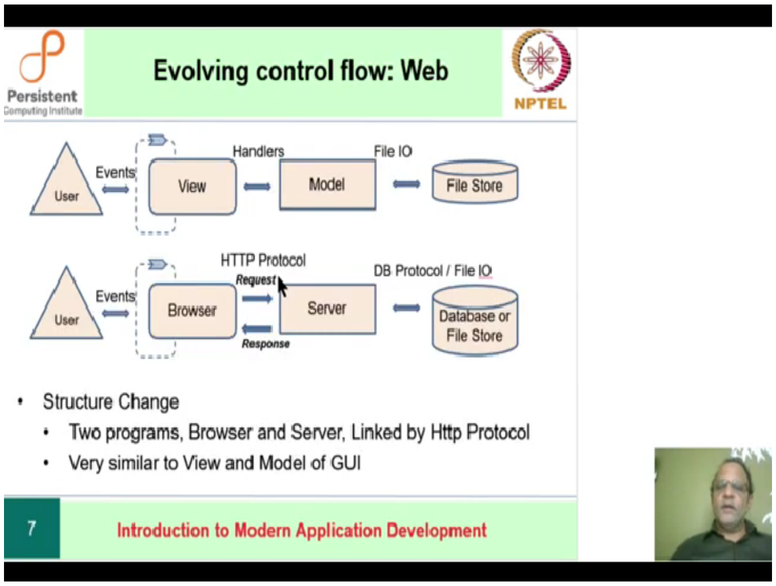
Changes to Architecture: GUI

- Separation between Controls, View, and Model
- Internal architecture of app also changes
  - Construct the UI using Controls (Widgets)
  - Link the Controls to the Computation
  - Explicit model to represent Data
- Event Loop: Listen to events and respond until terminated

6 Introduction to Modern Application Development

Internally, the architecture changes as we discussed below. What happens is that the program gets split into quite distinct parts. One part is responsible entirely for the looks and the other for the computation and the two parts are linked using handlers plus the event loop which takes the job of keeping the program going, listening to users requests and responding until the user decides to terminate the program. From these two programs we constructed the web program.

(Refer Slide Time: 07:38)


Evolving control flow: Web

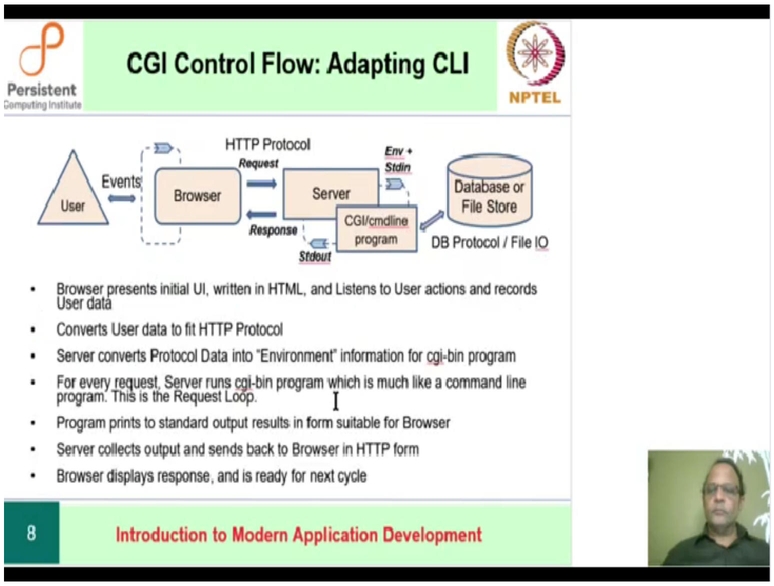
- Structure Change
  - Two programs, Browser and Server, Linked by Http Protocol
  - Very similar to View and Model of GUI

7 Introduction to Modern Application Development

So let us compare what it looks like to relative to a GUI program, just as there is a view and a model in a web program. There are the two components; the browser and the server for a web program. This time, instead of handlers linking the view and the model, the HTTP protocol connects the browser and the server. The analogy between view and model in a single application versus the browser server distinction is reasonably accurate for simple applications.

But as we have discussed earlier, in the GUI session, the analogy can get quite complicated as the complexity of the browser and server programs increases, but that is for a later time. To begin with, this analogy is very useful in understanding what is happening here.

(Refer Slide Time: 08:32)



CGI Control Flow: Adapting CLI

- Browser presents initial UI, written in HTML, and Listens to User actions and records User data
- Converts User data to fit HTTP Protocol
- Server converts Protocol Data into "Environment" information for cgi-bin program
- For every request, Server runs cgi-bin program which is much like a command line program. This is the Request Loop.
- Program prints to standard output results in form suitable for Browser
- Server collects output and sends back to Browser in HTTP form
- Browser displays response, and is ready for next cycle

8 Introduction to Modern Application Development

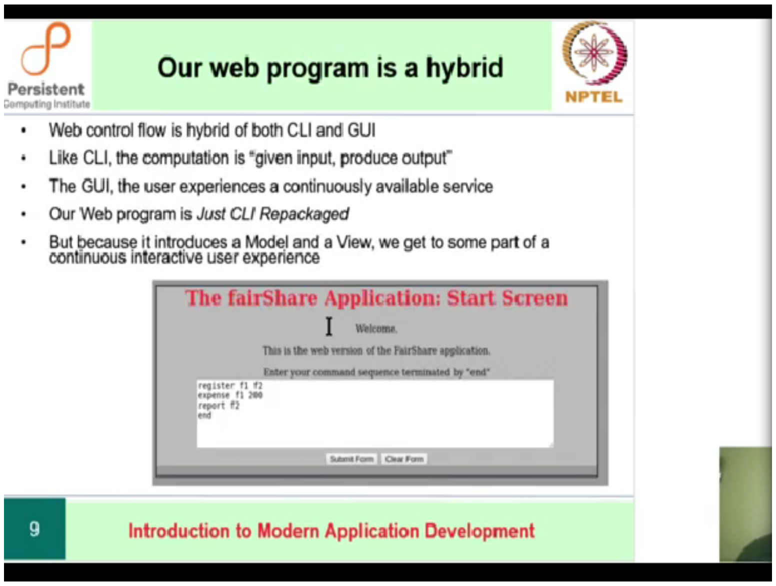
The next step that, we go to is to look at the internal architecture of how it is that we build the server program, and in our case, we achieved what is essentially a hybrid. We took the command line program, repurposed it and let the server executed. So, let's go through the sequence of what happens here. The browser presents the initial UI which is written in HTML and listens to user actions and records user data.

The browser then converts user data in a way that fits the requirements of the HTTP protocol, the server then converts protocol data into environment information for CGI bin program. If you compare the CGI bin versus the command line program, what we notice is that, they are by and large the same except that the way in which they connect to the environment in which they execute, differs.

So, instead of getting input from just the command line, the CGI program gets its input from the environment variables as well as standard in and after it is done it prints its result to the standard output which is, in turn, read by the server. The server then formats it in a form that works for the HTTP protocol and sends it to the browser. The content must be such that the browser can interpret it and display the response getting ready for the next cycle.

What is novel besides the split into the model and view part is that the browser and the server and the database program can exist on separate machines. So, it is as if there is a kind of efficient going on from the architecture of the GUI to the architecture of the web program and some aspects of the command line program can be preserved in this view.

(Refer Slide Time: 10:41)



Our web program is a hybrid

- Web control flow is hybrid of both CLI and GUI
- Like CLI, the computation is "given input, produce output"
- The GUI, the user experiences a continuously available service
- Our Web program is *Just CLI Repackaged*
- But because it introduces a Model and a View, we get to some part of a continuous interactive user experience

The fairShare Application: Start Screen

Welcome.

This is the web version of the FairShare application.

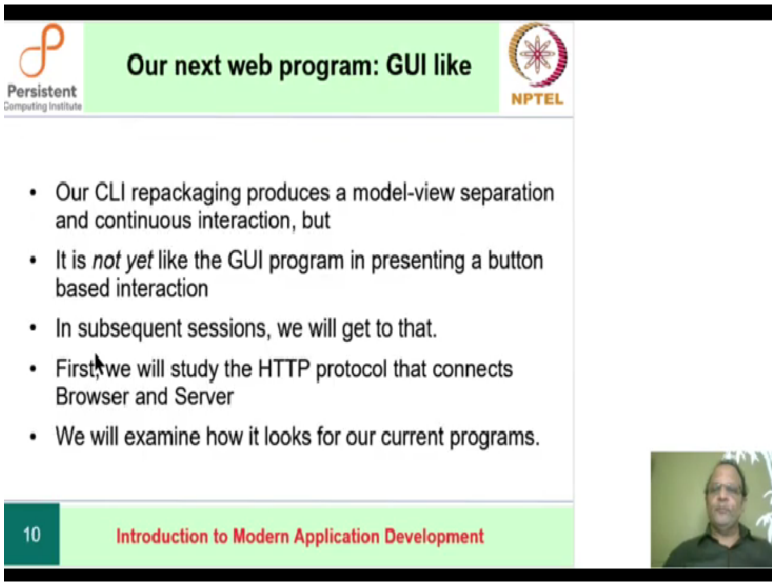
Enter your command sequence terminated by "end"

```
register f1 f2
expense f1 200
report f2
end
```

9 Introduction to Modern Application Development

So, what we have is a program that is basically a kind of a hybrid like CLI, as far as the computation part is concerned, it is essentially the same cycle given put produce output like the GUI; the user experience as a continuously available service. But because we have introduced a model and a view, you get some aspects of a continuous user experience even though the underlying program is much like a command line program.

(Refer Slide Time: 11:07)



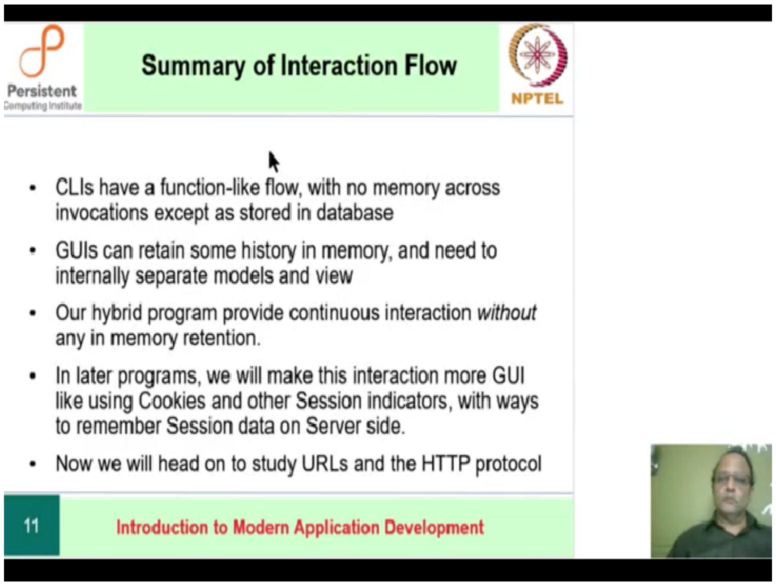
Our next web program: GUI like

- Our CLI repackaging produces a model-view separation and continuous interaction, but
- It is *not yet* like the GUI program in presenting a button based interaction
- In subsequent sessions, we will get to that.
- First, we will study the HTTP protocol that connects Browser and Server
- We will examine how it looks for our current programs.

10 Introduction to Modern Application Development

Our next program, however, will be much more GUI like. Our initial CLI packaging gave us only the model view separation but unlike a GUI program, there are no button-based interactions. But before we go on to that program, first - we will study the HTTP protocol that connects the browser and the server and we will see what it does for the programs that we have written so far.

(Refer Slide Time: 11:35)



Summary of Interaction Flow

- CLIs have a function-like flow, with no memory across invocations except as stored in database
- GUIs can retain some history in memory, and need to internally separate models and view
- Our hybrid program provide continuous interaction *without* any in memory retention.
- In later programs, we will make this interaction more GUI like using Cookies and other Session indicators, with ways to remember Session data on Server side.
- Now we will head on to study URLs and the HTTP protocol

11 Introduction to Modern Application Development

Let us summarize what we have seen of the interaction flow so far:

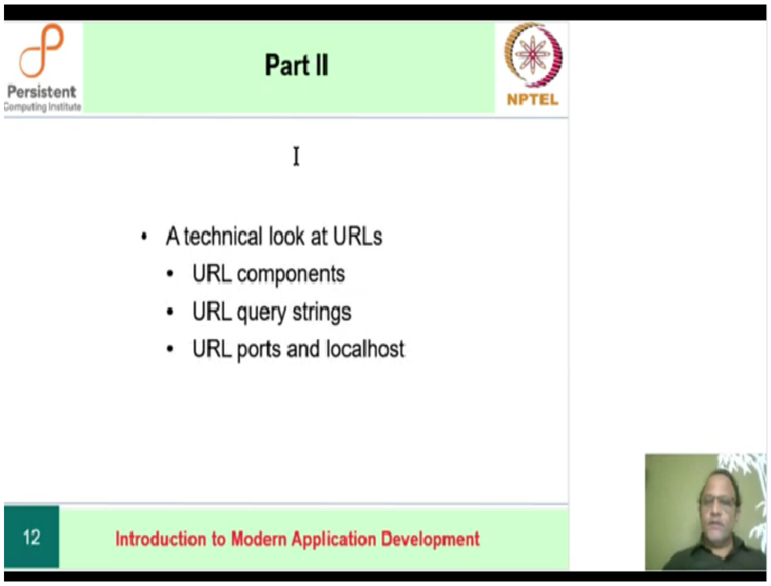
- CLI have a function like flow with no memory access across invocations except as stored in the database.
- In a GUI, you do retain some history in memory and we get the important new ability for the user interaction to depend on ‘how the user has interacted so far’.
- Our hybrid program provides continuous interaction without any in memory retention.

Because all the retention happens or should happen on the server side but the way the CGI program is written, it is essentially just a repackaging of the command line program. On the other hand, something important has already been achieved, because we have moved our program to the web infrastructure. We will find it very easy later, even if we were to keep the program just like CGI bin to add more facilities like multiple users and the ability for distributed users to share this program ,as we all know, from web apps.

So, besides the GUI like behavior, this is the other important thing that the web infrastructure gives us. And in later programs, we will see that you can recover the ability of the GUI to retain history by using cookies and other session indicators with ways to remember session data on server side which is distinct from the computational data.

But those are things that we will come to later. For now on, we will head on to study URLs and HTTP protocol. So, that is the next part of our session.

(Refer Slide Time: 13:26)



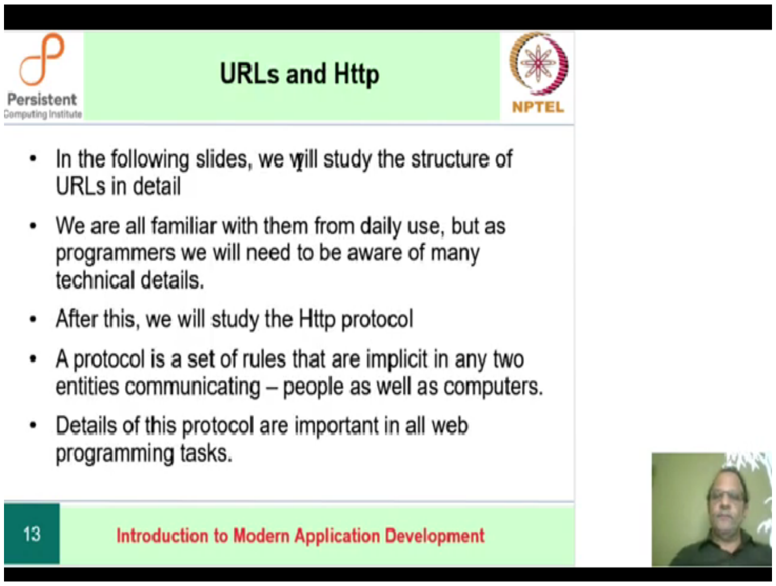
Part II

- A technical look at URLs
  - URL components
  - URL query strings
  - URL ports and localhost

12 Introduction to Modern Application Development

In this part, we will take a look at the components of the URL, especially query strings and other aspects like ports and local host. These are details that will start mattering when we start thinking of URLs as a kind of an interface to a web application.

(Refer Slide Time: 13:45)



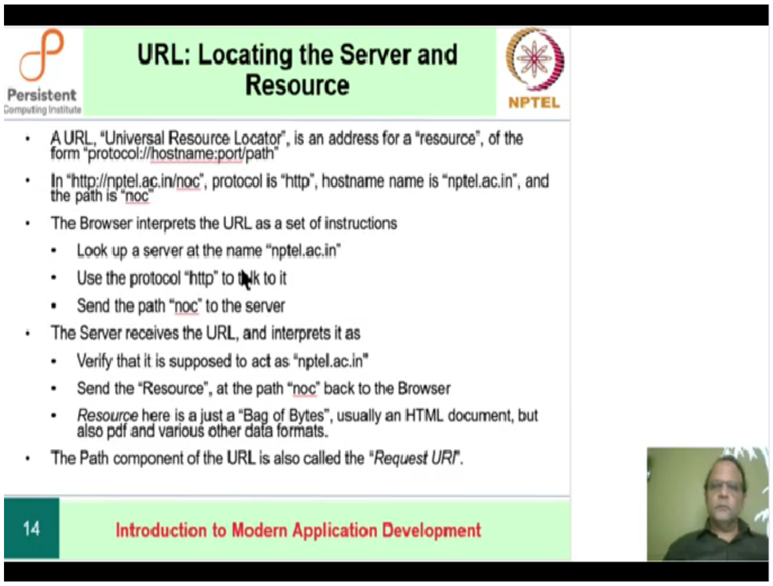
URLs and Http

- In the following slides, we will study the structure of URLs in detail
- We are all familiar with them from daily use, but as programmers we will need to be aware of many technical details.
- After this, we will study the Http protocol
- A protocol is a set of rules that are implicit in any two entities communicating – people as well as computers.
- Details of this protocol are important in all web programming tasks.

13 Introduction to Modern Application Development

So, although we are familiar with them from daily use, we need to understand them together as componentized interfaces plus participants in the HTTP protocol. A protocol is a set of rules that is implicit in any two entities that communicate whether it is people or computers and the details of how that communication happens matters in all web programming, too.

(Refer Slide Time: 14:17)



URL: Locating the Server and Resource

- A URL, "Universal Resource Locator", is an address for a "resource", of the form "protocol://hostname:port/path"
- In "http://nptel.ac.in/noc", protocol is "http", hostname name is "nptel.ac.in", and the path is "noc"
- The Browser interprets the URL as a set of instructions
  - Look up a server at the name "nptel.ac.in"
  - Use the protocol "http" to talk to it
  - Send the path "noc" to the server
- The Server receives the URL, and interprets it as
  - Verify that it is supposed to act as "nptel.ac.in"
  - Send the "Resource", at the path "noc" back to the Browser
  - *Resource* here is a just a "Bag of Bytes", usually an HTML document, but also pdf and various other data formats.
- The Path component of the URL is also called the "*Request URI*".

14 Introduction to Modern Application Development

A URL is the acronym for universal resource locator. It is, as it says, a locator an address for a resource. The structure of a URL is of the form protocol colon slash host name colon port slash path. This structure has two important parts; one is the hostname and port which identifies the server, the path which identifies the resource inside the server and the protocol, which for most common uses just turns out to be either HTTP or HTTPS.

For example: in `http://nptel.ac.in/noc`, the protocol is HTTP. The hostname is `nptel.ac.in` and the path is `noc`. We can think of this as a resource locator but to the browser, it functions like a set of instructions. When the browser is asked to fetch a URL, it takes the following steps: It first looks up the server and the name '`nptel.ac.in`' which is then located in the form of (what is called an) **IP address.** Some of this you may already be familiar with.

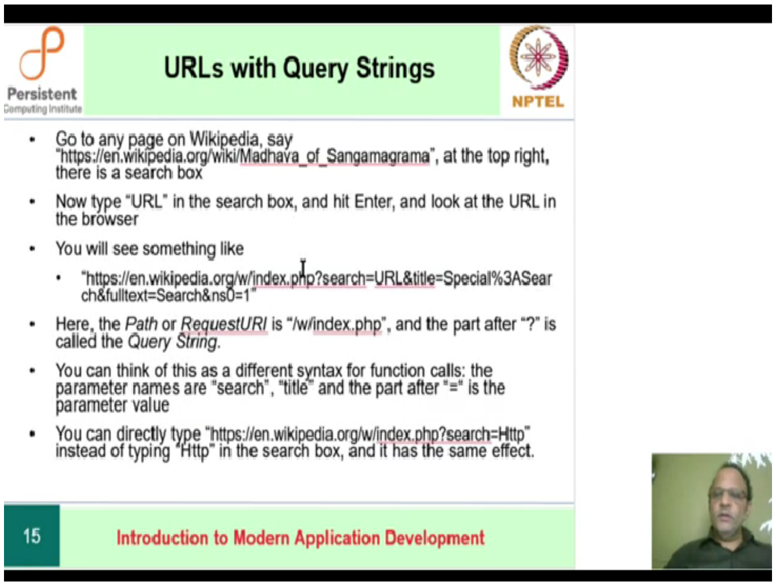
But, I am going to repeat these things in detail for those who are not. The mapping from this name to the IP address is done by something called a **DNS** (Domain Name Server), which we will briefly take a look at later. Then the next part of the instruction to the browser is that having located the server, the browser should use the protocol HTTP to talk to it. As part of talking to the server, it sends the path `noc` to the server and then the server takes over.

The browser now it just waits for the server to return the contents of the URL and here is what the server does next, it receives the URL and interprets it as the following sequence of instructions: first, verify that the server is supposed to act as the server called nptel.ac.in. This is done by means of configuration which is known to the server. The next step is to take the resource at the path noc, which it knows how to locate again by means of various configurations.

Then read the contents and send it back to the browser. The resource here is just what we can think of as a bag of bytes. Usually it is an HTML document, but also as we know, PDF and video and various other data formats are also available as resources. The path component of the URL is also called the **request URL**. We will summarize these technical terminologies such as resource locator, hostname and protocol, etc., later, but those are more familiar to us.

As programmers, it is the new unusual terminology like request URL that becomes important in discussing just exactly how we program the web. URLs like `nptel.ac.in/noc` are the simplest kind of URL. There are many more complicated URLs that we will have to take a look at in a minute. So, let us begin.

(Refer Slide Time: 18:01)



URLs with Query Strings

- Go to any page on Wikipedia, say "https://en.wikipedia.org/wiki/Madhava_of_Sangamagrama", at the top right, there is a search box
- Now type "URL" in the search box, and hit Enter, and look at the URL in the browser
- You will see something like
  - "https://en.wikipedia.org/w/index.php?search=URL&title=Special%3ASearch&fulltext=Search&ns0=1"
- Here, the *Path* or *RequestURI* is "/w/index.php", and the part after "?" is called the *Query String*.
- You can think of this as a different syntax for function calls: the parameter names are "search", "title" and the part after "=" is the parameter value
- You can directly type "https://en.wikipedia.org/w/index.php?search=Http" instead of typing "Http" in the search box, and it has the same effect.

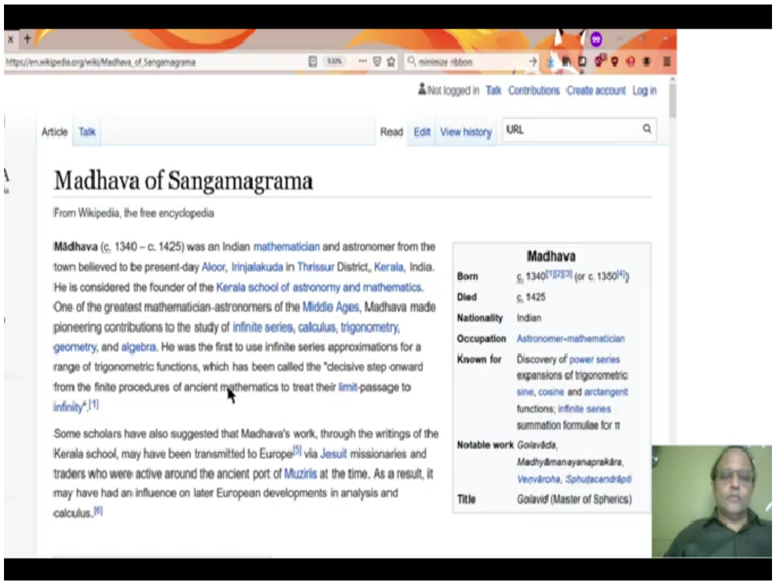
15 Introduction to Modern Application Development

The next kind of most common URL is what is called a URL with a query string. Try the following: go to any page on Wikipedia, say “`en.wikipedia.org/MadhavaofSangamagrama`” and then at the top right, you will see a search box. Now type URL in the search box and hit enter. And then look at the URL in the browser, you will see a complicated URL of the form shown below. And now let us try to identify the parts of the URL.

In this case, we will see that the path or what is called the **request URL,** is this part that I am going to highlight it is `/w/index.php`. The part after the question mark is called the **query string** and in the current example it has things like search and title and so on. And we go on until the end of these special characters. So, you can this part after the question mark is called the query string. You can think of this as a different syntax for function parts.

The parameter names of the function are things like search title, and the part after every ‘equal’ is the parameter value. You can directly type URLs, like en wikipedia.org, etc. and put a search term directly in the browser. And it will have the same effect as doing the search from the box, even without some of the extraneous parameters that come from the box. So let us try this out.

(Refer Slide Time: 19:51)


https://en.wikipedia.org/wiki/Madhava_of_Sangamagrama

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history URL

Madhava of Sangamagrama

From Wikipedia, the free encyclopedia

Mādhava (c. 1340 – c. 1425) was an Indian mathematician and astronomer from the town believed to be present-day Aloor, Irinjalakuda in Thrissur District, Kerala, India. He is considered the founder of the Kerala school of astronomy and mathematics. One of the greatest mathematician-astronomers of the Middle Ages, Madhava made pioneering contributions to the study of infinite series, calculus, trigonometry, geometry, and algebra. He was the first to use infinite series approximations for a range of trigonometric functions, which has been called the "decisive step onward from the finite procedures of ancient mathematics to treat their limit-passage to infinity".[1]

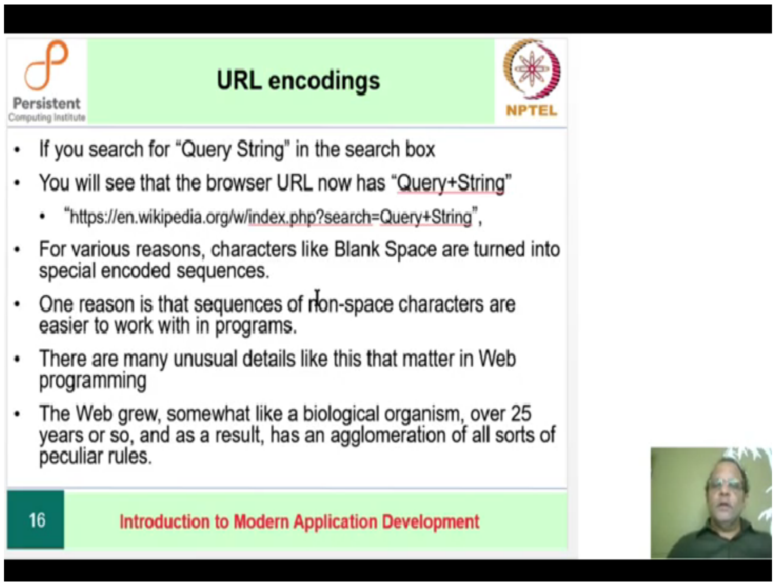
Some scholars have also suggested that Madhava's work, through the writings of the Kerala school, may have been transmitted to Europe[5] via Jesuit missionaries and traders who were active around the ancient port of Muziris at the time. As a result, it may have had an influence on later European developments in analysis and calculus.[6]

Madhava	
Born	c. 1340[1][2][3] (or c. 1350[4])
Died	c. 1425
Nationality	Indian
Occupation	Astronomer-mathematician
Known for	Discovery of power series expansions of trigonometric sine, cosine and arctangent functions; infinite series summation formulae for π
Notable work	*Golavāda*, *Madhyāmanayanaprakāra*, *Veṇvāroha*, *Sphuṭacandrāpti*
Title	*Golavid* (Master of Spherics)

Okay, let us try out the case that we have just seen. Here we are at the Wikipedia page for Madhava of Sangamagrama. And here we have the search box with URL written in it. **(Video Starts: 20:07)** If we hit enter here now, we can see the URL with a query string which contains things like search and title and full text search, which tells us that we have asked Wikipedia, to basically look at everything with the word URL in it, especially the ones that contain the title of the given URL.

And like I said, we can now go to the query string and remove the URL part and put HTTP. And now you will see that here we are getting everything about HTTP, for example, hypertext transfer protocol, HTTPS, etcetera, etcetera are somewhat different behavior happens if you remove the extraneous parameters and try just HTTP. Now, because there is a unique page desired. We go straight to the HTTP page and notice that the URL, although we typed it as in the query string form, has changed into a normal looking URL. **(Video Ends: 21:15)** We see further interesting behaviors once you start becoming aware of how the URL search engine.

(Refer Slide Time: 21:23)



URL encodings

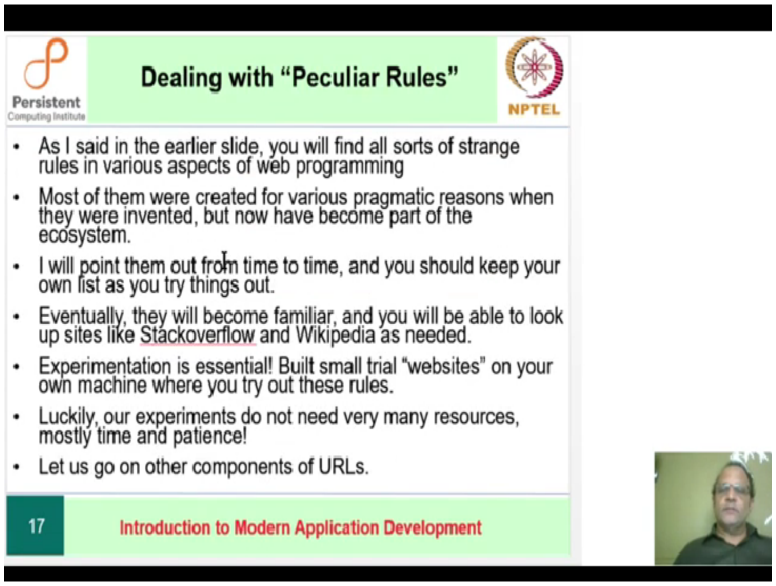
- If you search for "Query String" in the search box
- You will see that the browser URL now has "Query+String"
    - "https://en.wikipedia.org/w/index.php?search=Query+String",
- For various reasons, characters like Blank Space are turned into special encoded sequences.
- One reason is that sequences of non-space characters are easier to work with in programs.
- There are many unusual details like this that matter in Web programming
- The Web grew, somewhat like a biological organism, over 25 years or so, and as a result, has an agglomeration of all sorts of peculiar rules.

16 Introduction to Modern Application Development

For instance, if you search for the phrase query space string in the search box, you will see that the browser URL turns into a query plus string. For example, in the Wikipedia case, we have search equals query plus string followed by the other search parameters. For various reasons, characters like blank spaces are turned into special encoded sequences. One reason for this is that sequences of non space characters are often easier to work with in programs.

But there are other historical reasons why certain encoding exist. There are many unusual details like this that matter in web programming. As a web programmer you often learn these as you go on but you have to be aware that they exist. One reason why there are such strange rules is that the web grew somewhat like a biological organism for 25 years or so. And as a result, it has an agglomeration of all sorts of peculiar rules that were created when different people made changes to the web for different reasons. And the web we see today is actually a result of all those changes that happened through many years.

(Refer Slide Time: 22:42)



Dealing with "Peculiar Rules"

- As I said in the earlier slide, you will find all sorts of strange rules in various aspects of web programming
- Most of them were created for various pragmatic reasons when they were invented, but now have become part of the ecosystem.
- I will point them out from time to time, and you should keep your own list as you try things out.
- Eventually, they will become familiar, and you will be able to look up sites like Stackoverflow and Wikipedia as needed.
- Experimentation is essential! Built small trial "websites" on your own machine where you try out these rules.
- Luckily, our experiments do not need very many resources, mostly time and patience!
- Let us go on other components of URLs.

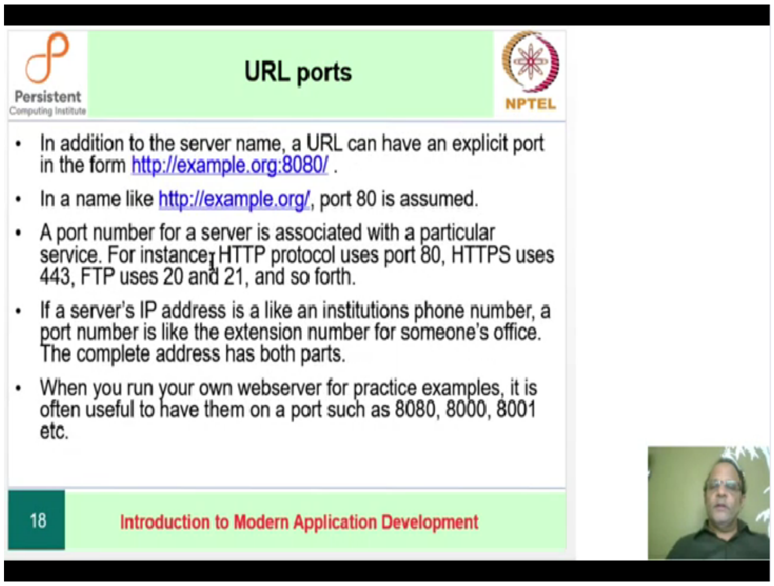
17 Introduction to Modern Application Development

So, how do we deal with these peculiar roots? Most of such roots were created, like I said, for reasons that were good when they were invented, but now they have simply been absorbed into the ecosystem. From time to time, I will point out these kinds of things to you and you should also keep your own list of things that you will notice. For me after so many years, many of these things have become second nature.

But for you who are encountering them for the first time making a list is perhaps the best thing you can do. In the end, they will become familiar to you. And you will be able to look up sites like *Stackoverflow* and *Wikipedia* as needed for idiosyncrasies which people used to know at one point in time talked about it. And luckily for us, they are available for us to learn from. In all this experimentation is essential.

You should build small trial websites on your own machine, where you try out the rules and understand them in detail. Luckily, unlike scientists or other people who need to work with physical things, our experiments do not need many resources. They just need time and patience.

(Refer Slide Time: 23:56)



URL ports

- In addition to the server name, a URL can have an explicit port in the form http://example.org:8080/ .
- In a name like http://example.org/, port 80 is assumed.
- A port number for a server is associated with a particular service. For instance, HTTP protocol uses port 80, HTTPS uses 443, FTP uses 20 and 21, and so forth.
- If a server's IP address is a like an institutions phone number, a port number is like the extension number for someone's office. The complete address has both parts.
- When you run your own webserver for practice examples, it is often useful to have them on a port such as 8080, 8000, 8001 etc.

18 Introduction to Modern Application Development

The next component to think about is a **URL port**. In addition to the server name for URL can have an explicit port in the form, say 8080. This URL does not actually exist, but the site example.org exists. Example.org was a site created in the early days of the web, for the sole purpose of being, like it says, *an example.* And in a short name like example.org, the port 80 is actually assumed.

You can pause this video now and go visit example.org to see what it says. The port number because it is common for most URLs when it is 80 it is never, it is not mentioned by the browser. You may have noticed that browsers nowadays even hide protocols like HTTP, although at one time they never used to do things like this. So, what is a port number? The port number for a server is associated with a particular service.

For example:

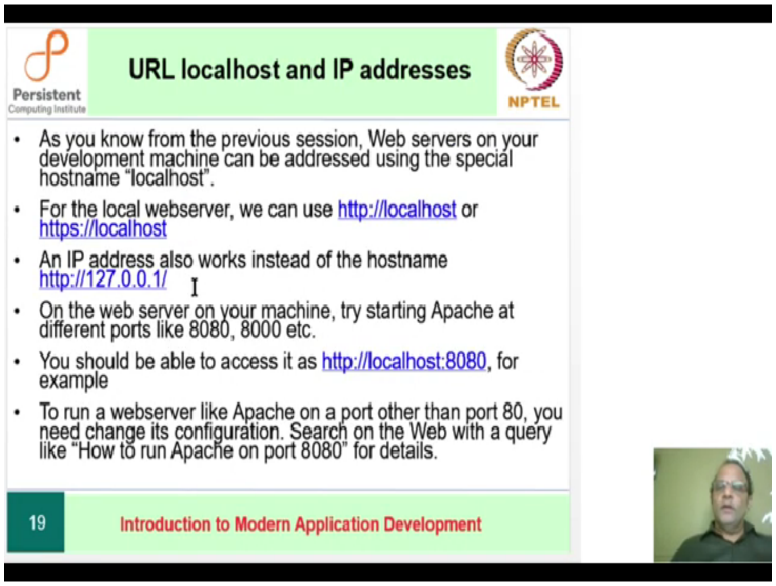
- the HTTP protocol uses port 80.

- HTTPS uses 443.
- FTP uses 20 and 2.

If a server's IP address is like an institution's phone number, the port number is like the extension number for someone's office in that institution and the complete address for a person in the case of the office is both parts. Similarly, the complete address for a service is the name of the server followed by the name of the port.

When you run your own web server for practice examples, it is often useful to have more than one web server running on different ports such as 8080, 8000, 8001… etc. Port numbers below 1024 are result for many purposes. So practice site ports are often it related for historical reasons you can use any name you want, any number you want, and it makes no difference. So, let us see how does it look like to run multiple local host services on your machine.

(Refer Slide Time: 26:12)



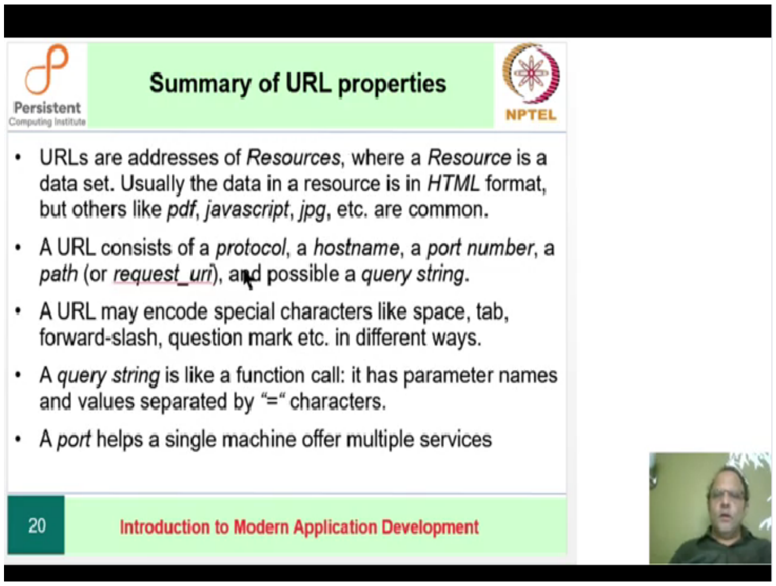
URL localhost and IP addresses

- As you know from the previous session, Web servers on your development machine can be addressed using the special hostname "localhost".
- For the local webserver, we can use http://localhost or https://localhost
- An IP address also works instead of the hostname http://127.0.0.1/
- On the web server on your machine, try starting Apache at different ports like 8080, 8000 etc.
- You should be able to access it as http://localhost:8080, for example
- To run a webserver like Apache on a port other than port 80, you need change its configuration. Search on the Web with a query like "How to run Apache on port 8080" for details.

As we have seen from the previous session, web servers and development machines can be addressed using the special hostname localhost. For the local web server, we can use `http://localhost`, or `https://localhost`. Typically, nowadays a web server like apache will be configured to present both HTTP and HTTPS endpoints. And IP address will also work instead of the hostname.

For example, for the local host you can use "`127.0.0.1`". And on the web server on your machine, you can try starting apache at different ports. To do this you have to be able to change the configuration for apache or any other web server that you use. For whatever web server you use, search on the web with a query, like how to run apache on port 8080. And usually the first answer will tell you exactly what to change in the configuration.

(Refer Slide Time: 27:17)



Summary of URL properties

- URLs are addresses of *Resources*, where a *Resource* is a data set. Usually the data in a resource is in *HTML* format, but others like *pdf*, *javascript*, *jpg*, etc. are common.
- A URL consists of a *protocol*, a *hostname*, a *port number*, a *path* (or *request_uri*), and possible a *query string*.
- A URL may encode special characters like space, tab, forward-slash, question mark etc. in different ways.
- A *query string* is like a function call: it has parameter names and values separated by "=" characters.
- A *port* helps a single machine offer multiple services

20 Introduction to Modern Application Development

Alright, now that we have seen quite a few details about URLs, let us summarize them. URLs are addresses of resources, where a resource is a data set of some time. Usually the data in a resource is in HTML format, but others like PDF, JavaScript, JPG, … etc. are common. The URL consists of *a protocol, the host name, a port number, a path, which is also called a request URL, and possibly a query string*

A URL can include special characters like space tab, forward slash question mark, etc. in different ways. A query string is like a function call. It has parameter names and values that are separated by equal characters. But port helps a single machine offer multiple services. This terminology is so common that we will be having exams and assignments that ensure that you learn this terminology well.

Alright, so now we are done with URLs. And our next step will be to look at details of the protocols.