

# **IIT Madras**

**ONLINE DEGREE**

**Modern Application Development – I**  
**Professor. Nitin Chandrachoodan**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Madras**  
**API Design**

Hello everyone, and welcome to this course on modern application development. Hello, everyone. So far, we have looked at various aspects of the process of developing an app for the web. And we mostly focused on the model view controller approach. We looked at what a model is, how it is used in order to store data? The view, is the presentation that is actually seen by the end user.

The controllers are actions, that are used in order to effect the model and thereby retrieve what data needs to be seen by the user. Now, in addition to that, we also looked at a little bit of background on the web in general, sort of data encoding formats, and so on. And what we are going to do now, is get a little bit more into detail on programming interfaces. In particular, what are called application programming interfaces, or API's.

And in connection with that, there is one very important term which is called REST, REpresentational State Transfer, which we will be looking into in a lot more detail now. This is an important idea from the point of view of understanding certain things about how apps work and how apps should be developed for a web type of architecture or web type of platform. And what exactly do we mean by that is something that we will look into in more detail.

(Refer Slide Time: 1:27)



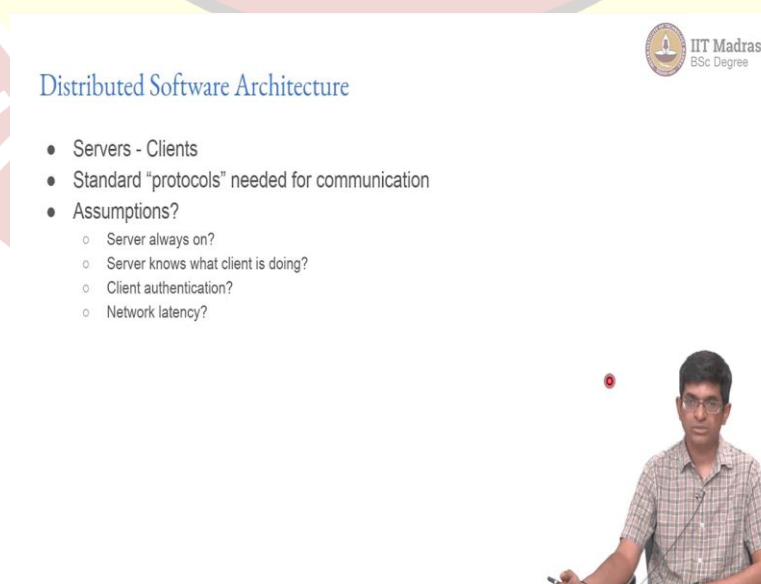
The slide is titled "API Design" in blue text. It features a list of topics: "Web architecture - REST", "API Examples", and "OpenAPI specification". The IIT Madras BSc Degree logo is in the top right corner. A lecturer is visible in the bottom right corner of the slide frame.

## API Design

- Web architecture - REST
- API Examples
- OpenAPI specification

So, let us start by looking at the web architecture, what we mean by an architecture for a web? What is this rest? We will then look at some examples of API's. And then finally go into some detail on something called the open API specification, which is commonly used these days as one way of specifying an API, in a machine readable form. Meaning that, it is something that can be understood both by humans who are writing the specification, and can be sort of interpreted by machines. In such a way that, the machine and the human have exactly the same understanding of things, and there is no ambiguity, note that.

(Refer Slide Time: 2:10)



The slide is titled "Distributed Software Architecture" in blue text. It features a list of topics: "Servers - Clients", "Standard 'protocols' needed for communication", and "Assumptions?". The "Assumptions?" section has sub-points: "Server always on?", "Server knows what client is doing?", "Client authentication?", and "Network latency?". The IIT Madras BSc Degree logo is in the top right corner. A lecturer is visible in the bottom right corner of the slide frame.

## Distributed Software Architecture

- Servers - Clients
- Standard "protocols" needed for communication
- Assumptions?
  - Server always on?
  - Server knows what client is doing?
  - Client authentication?
  - Network latency?

So, let us start with the first part, which is basically the concept of REST. Why is it needed? What is it? And what do we need to know about it? So, the first thing that we are, where we need to keep in mind over here is that, any application that we are trying to develop over the web as a platform, as we have discussed earlier, is in some sense, a distributed architecture. Because the clients and the servers, fundamentally, by definition, are in different places, or at least on different machines most of the time.

It is very, in only in very specific situations, are you considering something where the server and the client are both on the same machine, that can exist, it is still a web app. But that is sort of the exception rather than the norm. So, distributed software architecture, in general, deals with the concept of servers and clients. And because, they are separated from each other in some way, there has to be some standard protocol, that is used for communicating between them.

So, the client needs to request certain things from the servers, and the server needs to be able to respond. So, unless they know what each other is talking about, they cannot really communicate. Now, various kinds of distributed software architecture can be built. And there are many assumptions that you can make. So, for example, one assumption that I could make when I am designing a distributed system, could be that, if I am assuming there is a server, I could assume that the server is always on otherwise, the application is just not going to work.

I could assume that the server knows at any given point in time exactly what the client is doing. So, it has like, complete control over what is being displayed the client end and responds to each and every action on the client end. Is the client authenticated? Does it need to be authenticated? Or, just by the fact that there is a connection between server and client, is that enough to sort of say that, yes, this client is allowed to work with the server?

What happens about network latency? Many distributed applications could be built, maybe for an office kind of environment. Let us say the payroll processing or the workflow systems within an office, where you might make the assumption that, your application is running in your data center, which is under your control. And the people who are going to use it, are all going to be sitting within the same building, same office. So, network latency, network delays, and so on are not really a concern. Now, these are possible assumptions, when you are designing a distributed architecture in general.

(Refer Slide Time: 4:48)

The Web

- Client - Server may be far apart
- Different networks, latencies, quality
- Authentication? Not core part of protocol
- State?
  - Server does not know what state client is in
  - Client cannot be sure what state server is in

On the other hand, when you are designing something for the web, some of these assumptions have to be changed. So, for example, the client and the server, maybe and in fact, in many cases, are far apart. And what do I mean by far apart? It could literally be on the other side of the world. So, it is not uncommon to have applications that we use on a daily basis, where the servers are located, let us say in the US, which is more or less the farthest geographically, from where we are that we can think about it.

Possibly South America, you could find places that are slightly farther away, but it is the same, similar range that we are talking about. You could also potentially have something in outer space, running as a web server. But that is, again, the exception. But, having something on the other side of the world is common, rather than something which is going to be a rare occurrence on the web.

Not just that, the path from you to the server could go through different networks, depending on whether you are on a mobile cellular network, or you are on a home broadband fiber network, or, some kind of a fiber network in your school or office. They could have different speeds, they could have different latencies, they could have different quality, meaning that, the packet drops or outages, all those things can affect the quality of the network.

What else can we say about the web, we are used, of course, to the idea of authentication, meaning that, you keep talking about logging into a website, log into Google and, log into

Amazon. So, when you say log in, you are basically saying, you have authenticated yourself in some way. The important thing to keep in mind is, that is not a fundamental part of how the web was built, the original idea of the web was simply as a document serving process.

You connect to a server, you ask it for a document, it gives you back documents. The server does not really keep track of who is connecting, what they are supposed to do, there is no context in the original definition of the web. Clearly, that is not where we are now. But still, what it means is that the authentication mechanisms were sort of bolted on after the fact, after the original design of the system.

And what about state? So, in other words, does the server know exactly what the client is doing at any given point in time? And the answer is no. All at the server can be sure of is, at some point, this client asked me for something, and I responded in this way, did the client even get that response? Did the client actually displayed to the user? The user sort of see it and decide what to do about it? None of that is known to the server.

In the same way the server, the client cannot know in what state the server is. An important implication of that is, let us say I am connecting to google.com. I actually do not even know which server, where in the world I am connecting to. But all I can say is that, this is some information coming back from someone who claims to be in google dot com. And because of certain security mechanisms in the way that the domain name system works, and so on, I can be reasonably sure that yes, it actually came from a server owned by Google.

But, I do not know what state that server was in, I do not know what kind of machine it was running, I do not know whether it just started whether it has been running for a long time, none of that is known to the client. So, all of these, in other words are sort of inbuilt or basic properties of the web, as a platform. And anytime you are designing some kind of distributed software architecture for this, you need to keep these in mind.



(Refer Slide Time: 8:22)

Architecture for the Web

- Roy Fielding, PhD thesis 2000 UC Irvine
- "REpresentational State Transfer" - REST
  - Take into account limitations of the Web
  - Provide guidelines or constraints
- Software Architecture Style
  - Not a set of rules!

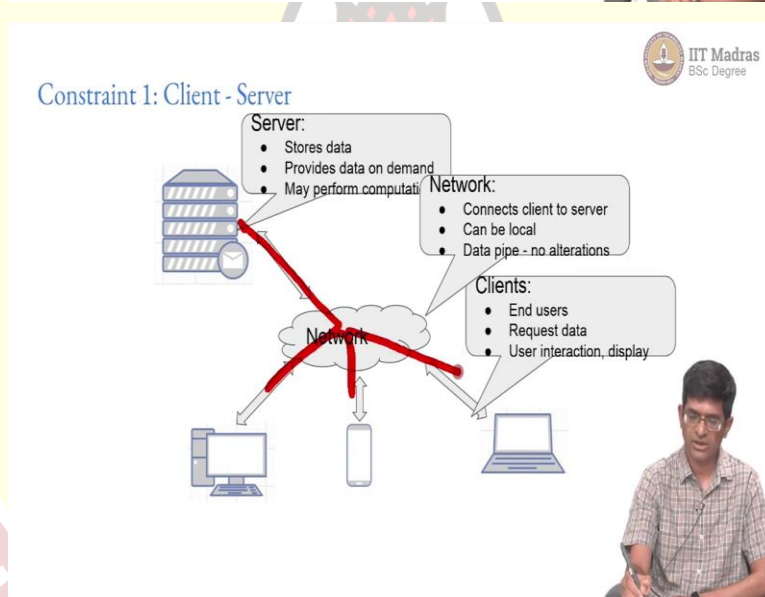
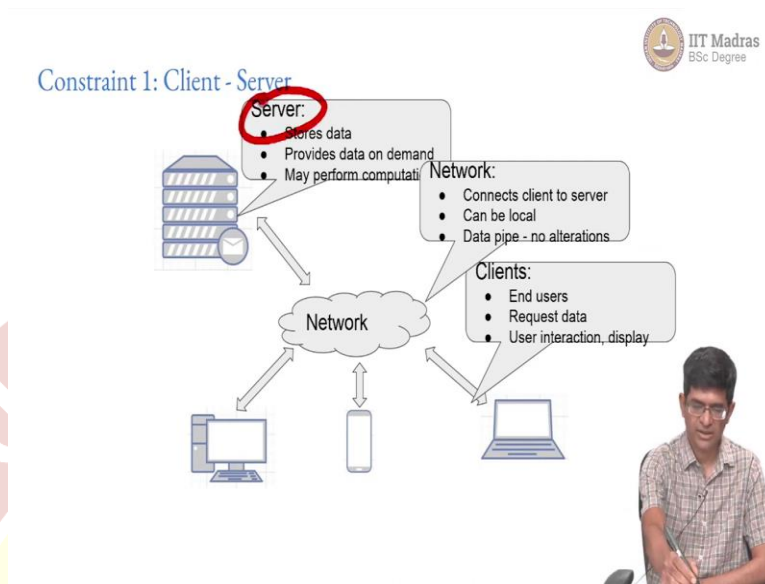
So, now, based on all of this, a person named Roy Fielding, he was originally working on the Apache web server. So, he has a lot of background in the domain of, web architectures, and so on. And he studied this as part of his PhD thesis at University of California, Irvine. And in 2000, that thesis was completed. And one of the main contributions from the thesis was this notion called representational state transfer, which got the abbreviation, REST.

Now, this REST essentially takes into account limitations of the web as a platform. And it provides a number of guidelines or constraints on how you should actually go about designing a software application. So, that it works within the confines of this web as a platform. An important thing to keep in mind over here is, this is a software architecture style. So, REST is not a specific set of rules.

So, it is not really correct to say, is this REST or is that REST? It is more a set of guidelines, or it is a design style similar to what we have been talking about MVC, it is a design pattern. Similarly, in this case, it is not a software design pattern in the sense that, it is not, how you can write a specific piece of code. It is more of a software architecture style. It is like one level higher than actually writing code.

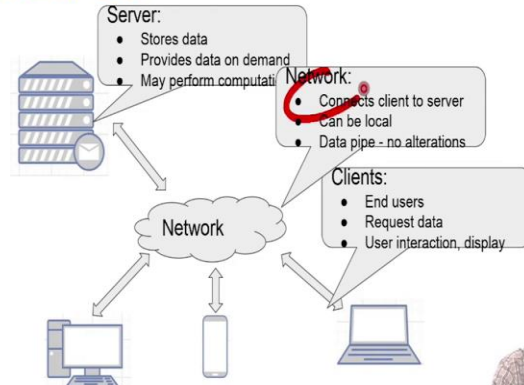
How do you design the architecture of your software? So, it is not a specific set of rules. It is sort of style guidelines, but following that is likely to give you an architecture which is well suited for the, for running on the web.

(Refer Slide Time: 10:06)





### Constraint 1: Client - Server




So, the best way to explain REST is to understand it in terms of the constraints that it imposes on your architecture, or rather, the other way around. It is working within a certain set of constraints, it assumes certain things. And by keeping those in mind, we can sort of achieve the principles of REST. And it also sort of explains why REST is defined the way it is.

So, the first constraint that is assumed for the web architecture, the web platform is that, of course, it is a client server platform. This is something we have been repeating multiple times. So, you have a server that provides data, it stores data connects over a network, to various clients.



So, you have different clients, they could be desktops, laptops, mobile devices, or even some kind of other machine which is actually sitting in a data center somewhere. So, the point is, the client does not necessarily need to even be a human being. What we are talking about is how do we define a software architecture? The important point over here is, there is a client, notion of a client, there is a notion of a server and a network connecting the two.

(Refer Slide Time: 11:24)



### Constraint 2: Stateless

- Server cannot assume state of client:
  - which page are you looking at
  - is a request coming from an already logged in user just because of the address?
- Client cannot assume state of server:
  - did server reboot since last request?
  - is this request being answered by same server?

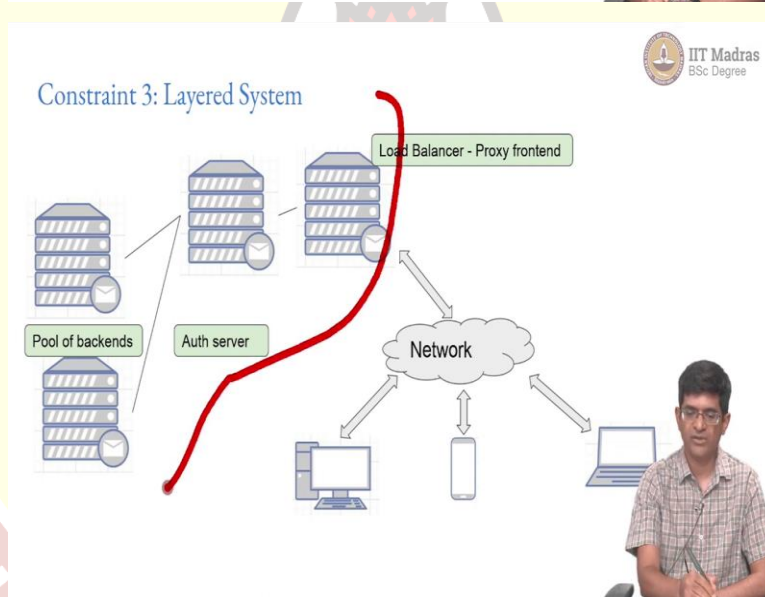
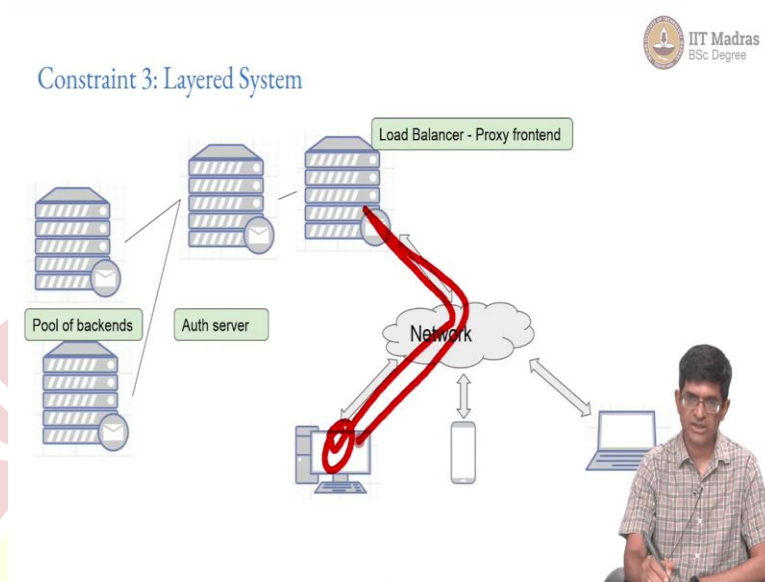


One very important constraint, imposed by the web is this notion of statelessness. And this is something that is important to keep in mind. And to understand, when you are designing applications for the web platform, the main sort of, at a high level, like I have already said, the important things that you need to understand over here is the server cannot assume the state of the client.

All that the server knows is, it sent back some information to the client, if the client receive it, the client display it has, is the user actually looking at it? None of that is known to the server. Similarly, the client cannot assume any state of the server. Is this the same server I was talking to, that I spoke to yesterday? Is this response coming from the same geographic location? Has this server been up for like, one day or one hour?

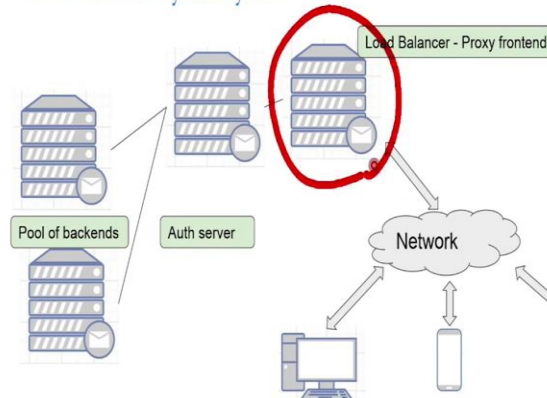
Or did it just get rebooted a few seconds back? None of that information is typically known to the client. And it should not be known or rather need not be known. So, statelessness basically means that you have to design your software, as I said, it will even work that it will work without knowing any of these things in detail.

(Refer Slide Time: 12:37)



सिद्धिर्भवति कर्मजा

### Constraint 3: Layered System

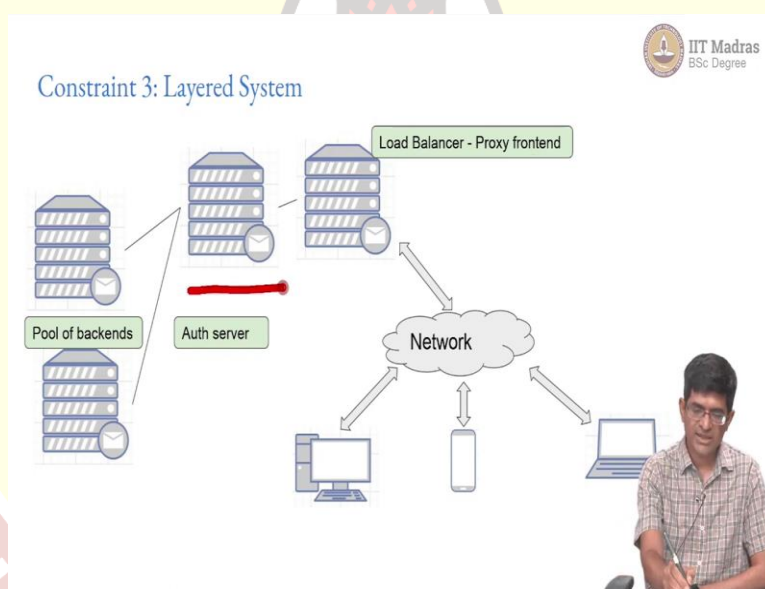
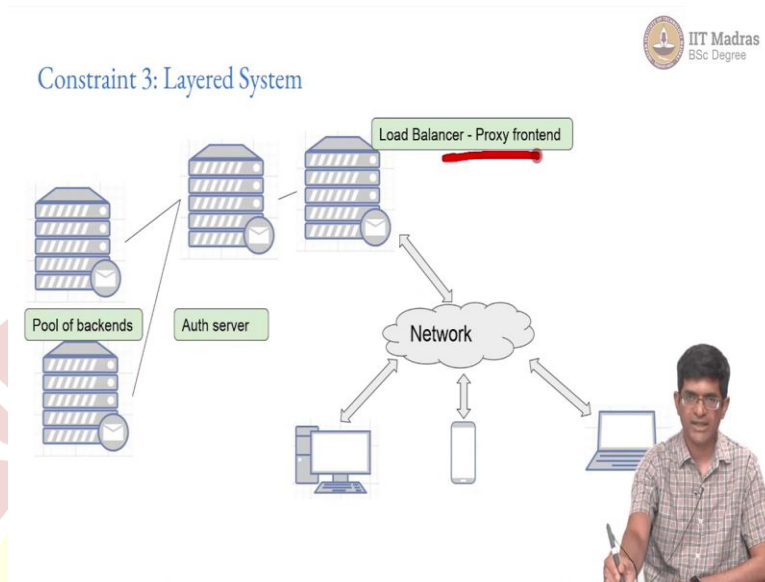


Now, the next constraint or assumption that was made about the web as a platform, is that it is a so called layered network. And this requires a little bit of explanation, because we have not talked about it so far. I have generally mentioned that, you have a client, the client goes through a network to a server, and gets back a response. Now the point is, as far as the client is concerned, it does not really know where that response is coming from. It just knows that it came from somewhere in the network.

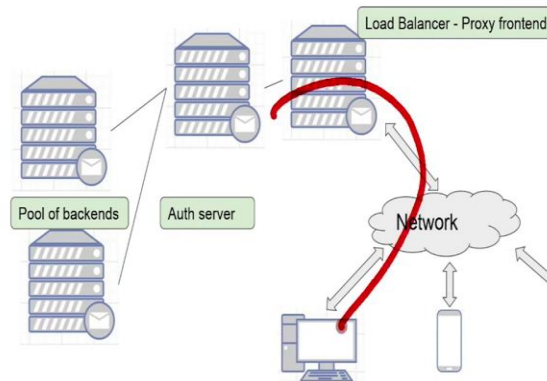
Now, in practice, what could have happened is that the server was not one unique server. It could be had multiple machines involved in the process. Very likely, the very first machine that you encounter on the network, after you have got to google.com, for example, is something called a load balancer. And what does a load balancer do? The load balancers job is simply to say, look, I am not going to do any computation, I am not going to do anything fancy.

All I know is that, there are clients who are making requests. And behind me, there are many servers who are capable of handling those requests. All that I am going to do is keep track of which of those servers is free at any given point in time, and handover the request to that server. So, that even if one server starts getting more and more load, because more people have been making requests, I can find another server and pass the request on to them.

(Refer Slide Time: 14:09)



### Constraint 3: Layered System



So, this, it is basically acts as some kind of a proxy. It is a frontend to the servers, this is not the same as the user frontend, it is not the user facing frontend. In this case, it is basically acting as something the frontend that is seen by the client side. It is the frontend of the server. And what happens in this case is the load balancer, its functionality, in other words, if, if you had only a load balancer, you do not have an application.

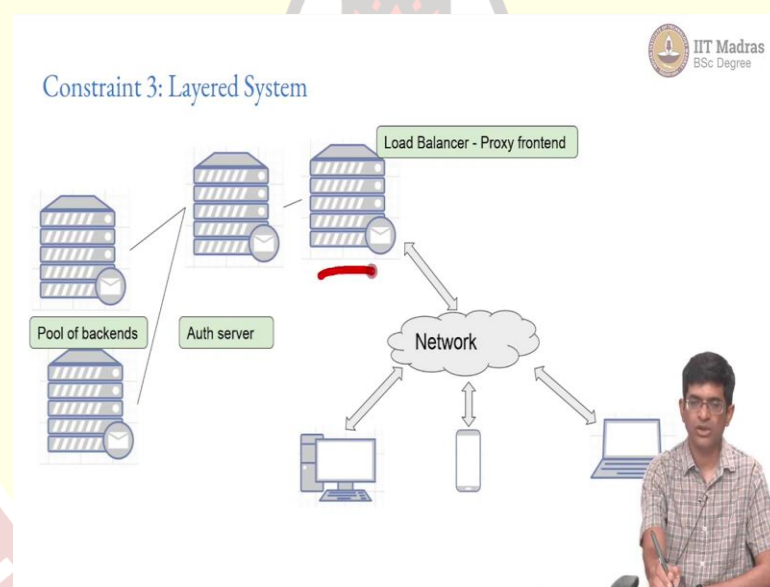
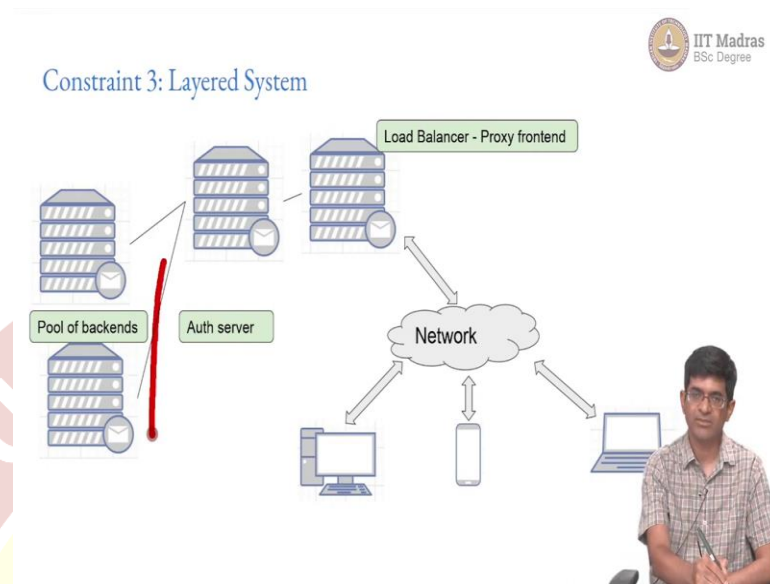
Because, it does not do any computation. It does not do any real work in the sense of, digging into a database, pulling out data, nothing of that sort. It has one very specific specialized task, which is to take requests and send them to the next free server. So, that is one. You might have something else, which is sometimes called middleware. It basically sits in the middle of your path. And after the proxy has sort of said, look, one of you servers, please handle this.

Someone else comes along, another server over there says, wait a minute, has this request been authenticated? You are trying to visit, let us say mail.google.com. But have you already logged in? And possibly what could happen is, right at this point, it intercepts and says, no look, I do not see a login information over here, please send something back to the client, asking them to log in. So, show them a login page.

Now, what that means is, now the request could just have come from this authentication server. On the other hand, let us say that I just went to google.com. The front page of google.com is not something that changes very often.



(Refer Slide Time: 15:44)



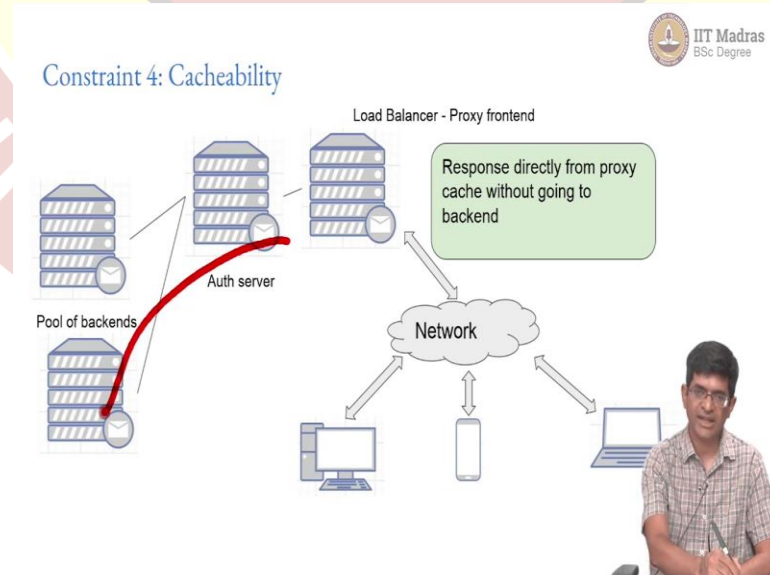
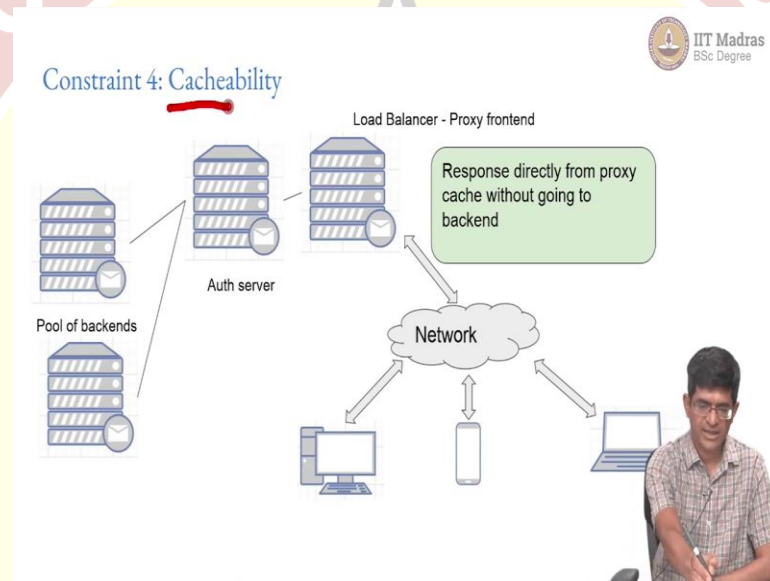
I do not even need to go until these backends, I do not need to go to my search engines. So, why not just let my load balancer itself give me back the front page, it is just static HTML more or less. So, in other words, from the end users point of view, I do not know whether the response is coming from this server, which is the frontend, this server, which is somewhere in the middle, and acting as an authentication middleware, or one of these servers that is actually doing the search. So, this layered architecture, I have one layer of the frontend.

I have this middleware layer, I have this backend. I could have multiple other layers as well. So, this is just a three layer structure that I have drawn, I have explained. But you could have more

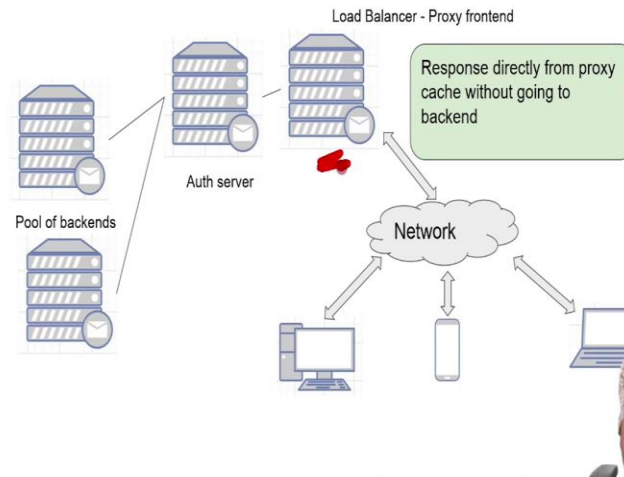
layers as well. And the important thing to keep in mind this, the client should not necessarily be aware even of which layer, the response came from. It need not even know how many layers are there in the architecture.

And it should not be concerned about which layer the response came from. In other words, if I add a new layer, something else to do authentication, or if I add another layer to may be clean up the requests before they go in, or another layer to log data. None of that is visible to the client, all of that is transparent. And the server can be modified without affecting the client.

(Refer Slide Time: 17:04)



#### Constraint 4: Cacheability

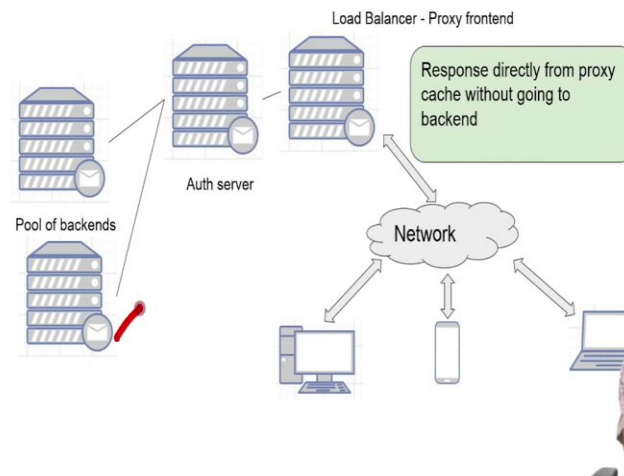


The related thing is what is called cacheability. And a cache, c a c h e, a cache is basically some kind of a temporary storage. What it is doing is it is saying that, look, rather than going all the way to these backends every time and then giving you back information, if this frontend could somehow store the information that you need, let us say that you are making many requests for the same thing.

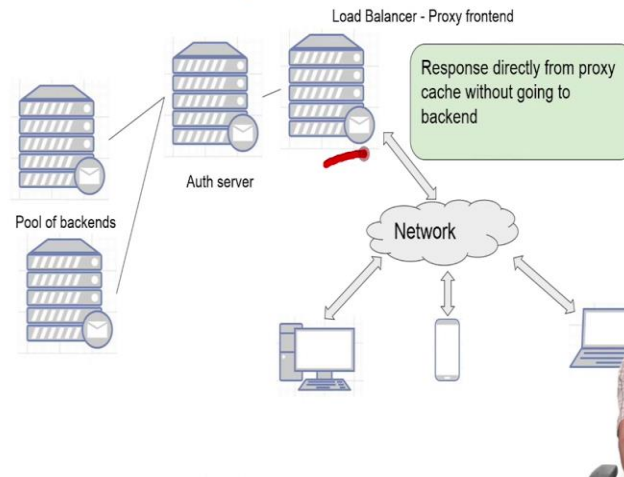
And the frontend has already sort of stored it, why not just give you the data from the frontend without going to the backend? An example of that is what I mentioned earlier, the front page of Google. The front page does not change very often.

(Refer Slide Time: 17:42)

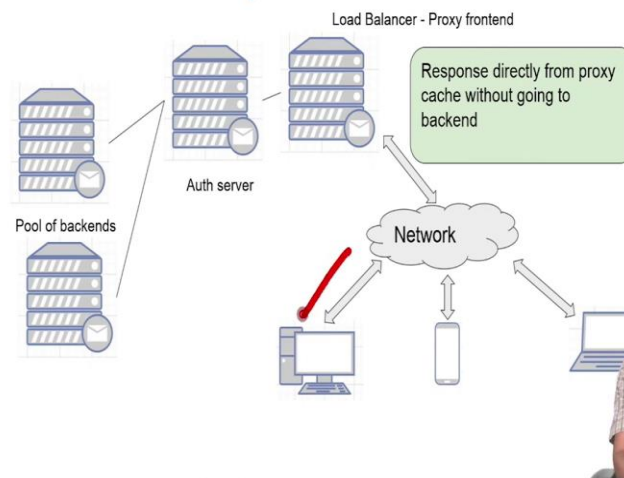
#### Constraint 4: Cacheability



#### Constraint 4: Cacheability



#### Constraint 4: Cacheability



And I definitely do not need to keep going back to the backend to pick that up. And let us say I have 1000 users coming in. If each one of them was going back all the way to the backend, it is probably unnecessary load, why not just let this proxy frontend, whose main job is, just to answer requests as quickly as possible.

Let it just send back the HTML corresponding to the front page. Now, you need to understand this a bit carefully. Cacheability does not necessarily say you have to implement caches, or you have to, make a cache of a certain size or you have to use this software for running a cache. No, it is talking about whether data can be cached or not, without the client knowing anything about it.

In other words, what it is saying is the server finally can make a decision on what things are permitted to be cached at different levels. So, usually, as part of the HTTP thing, you will see that along with the response coming from the server, you will also see some additional information which say something like cacheable on or off, and similarly expires, which is until when can you cache this. Which means that, that information about cache ability could also be picked up at different layers over here.

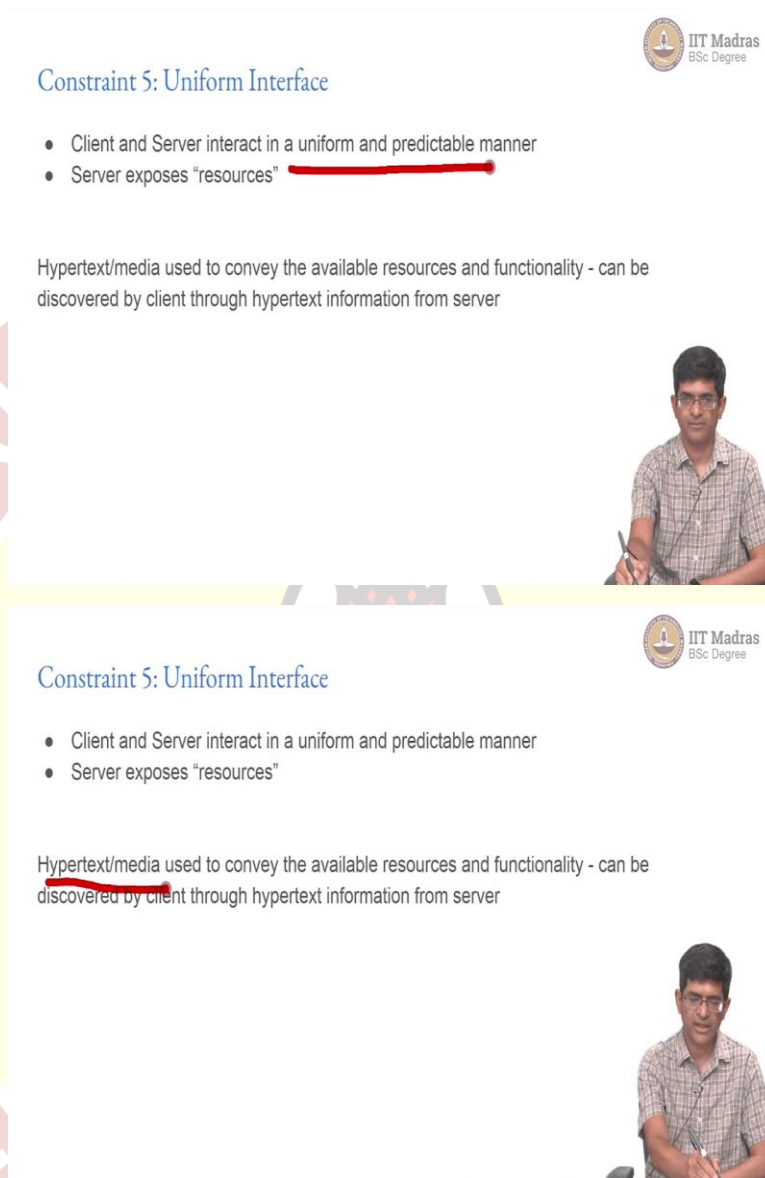
So, it is possible that this proxy frontend could pick up something from this backend. And by looking at the information, it realizes that yes, it is safe to keep this cached for, let us say, the next one day. And then any response coming in or the network, it will just basically respond to them very quickly from its cache.


Similarly, you might also give information all the way to the client saying, look, this information is cacheable. Their Google front page, or the fonts being used on the page are not going to change, at least until tomorrow. So, just keep the same font and how many hour times you are visiting Google today, use the same font to make the display. So, the cache ability information is finally controlled by the server.

It has to give that information and that information can then be propagated to various parts, and they can act accordingly. So, that notion of cacheability is what is important, not what software you use, or how big your cache is? For the fact that, that can be controlled. That is what is important from the context of, a web application.



(Refer Slide Time: 19:58)




 IIT Madras  
BSc Degree

### Constraint 5: Uniform Interface

- Client and Server interact in a uniform and predictable manner
- Server exposes "resources"

Hypertext/media used to convey the available resources and functionality - can be discovered by client through hypertext information from server

 IIT Madras  
BSc Degree

### Constraint 5: Uniform Interface

- Client and Server interact in a uniform and predictable manner
- Server exposes "resources"

Hypertext/media used to convey the available resources and functionality - can be discovered by client through hypertext information from server

Then, there is another constraint, which is called a uniform interface, in its original sort of formulation, this is a slightly complicated statement. But the simple way to think about it would be to say that, the client and server are expected to interact in a uniform and predictable manner. In other words, if I give a certain request, 10 times, I should get back the same kind of response. It may not be the exact same data.

Because, the data might depend on what was pulled out of the database, or it might depend on the time of day, but the type of response should be similar. And another way of looking at it is to say that the server ultimately is exposing certain resources. And a standardized way of



interacting with those resources. So, in the case of, let us say, a shopping cart application, you could think of an item that you would want to buy as a resource.

And the server essentially sort of shows this is the set of resources that I have available with me. And it provides a standard way of let us say, taking a certain resource and adding it to another resource, which is a shopping cart. So, the shopping cart itself could be thought of as a resource, you can add items to the shopping cart, you can list out the contents of the shopping cart, you can empty the shopping cart, lots of things that you can do over there.

And in general, from the context of, how to design a web application, one of the insights over there was, you could actually use some kind of hypertext or media, which comes when you connect to the server in the first place, in order to convey what are the available resources and the kind of functionality that can be expected. And this information can actually be discovered by the client, through hypertext information that comes from the server.

So, the client can basically connect to the server some standard, something like a home page, which in turn will tell it, these are the things you can do now. And from there, you can infer it. This part of it is slightly, I mean, it is a bit deeper in terms of what exactly the architecture of the web is, it is not something very obvious, or, that you need to know right off the bat.

But, it is also part of, especially this notion of resources exposed by the server and this uniform interface, the consistent interface that they present is important to keep in mind when you are designing.

(Refer Slide Time: 22:28)



(Optional) Constraint 6: Code on Demand

- Server can extend client functionality
  - Javascript
  - Java applets

Part of the overall architecture - these are not hard rules



Now, one last constraint was also there in terms of, how this was defined, which is that the server can in some cases, extend client functionality. Now, this was considered an optional constraint, it is just something that is permitted. Basically, to make this whole notion of, having Java Script or at that time, in the year 2000, JavaScript was not a big deal, Java applets were however.

So, the idea that I could actually send a Java applet, which would run on the client side, and would then do certain part of the functionality for me, was also sort of integrated into this. So, you could sort of share part of the work between the client and the server by pushing such information.

So, what is the bottom line? These are sort of the 6 constraints within which REST is defined. And they are all part of the overall architecture. Like I said, these are not hard and fast rules. It is not that, you can just go look at some software and say, this follows all of these rules, and therefore it is raised or this is not. It is more these are guidelines on how you should think about the system, if you want to design a software architecture for it.