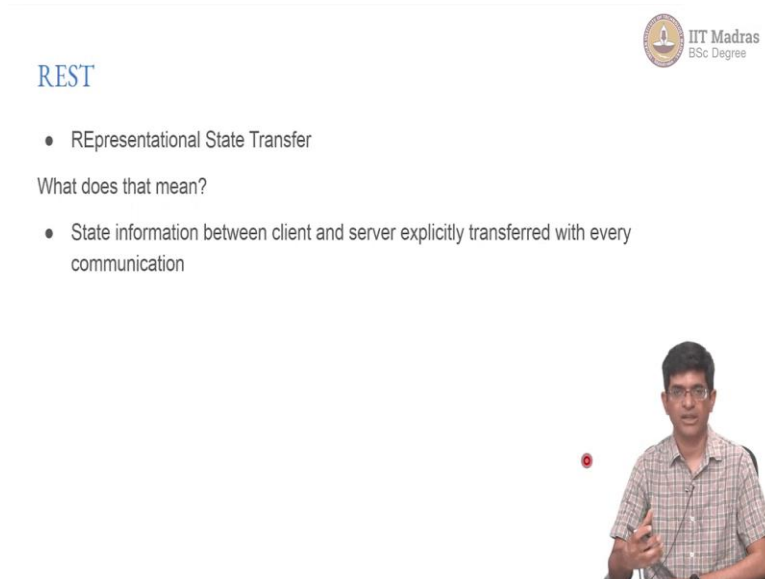


IIT Madras

ONLINE DEGREE

Modern Application Development - I
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
REST

(Refer Slide Time: 0:16)



The slide features the IIT Madras BSc Degree logo in the top right corner. The word "REST" is displayed in blue text. Below it, a bullet point defines REST as "REpresentational State Transfer". A question "What does that mean?" is posed, followed by another bullet point stating that "State information between client and server explicitly transferred with every communication". A small video inset in the bottom right shows Professor Nitin Chandrachoodan speaking.

REST

- REpresentational State Transfer

What does that mean?

- State information between client and server explicitly transferred with every communication

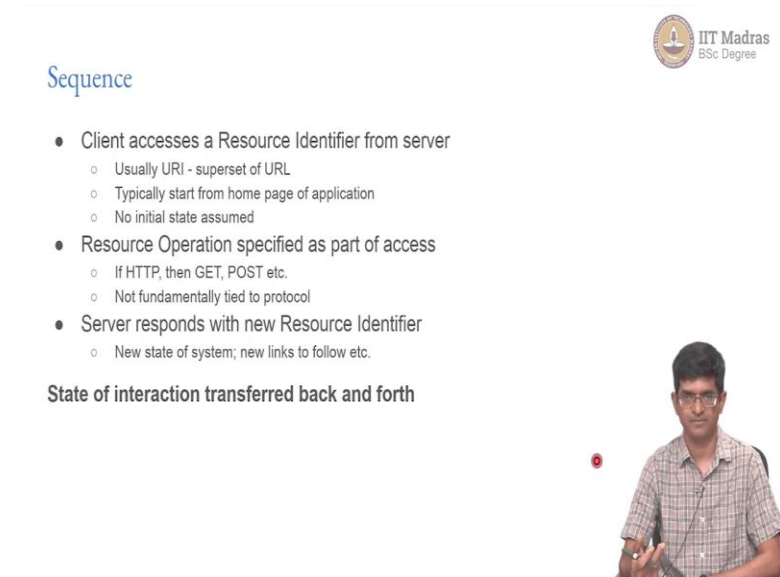
Hello, everyone, and welcome to this course on modern application development. So, with all of that in mind, this notion of REST, as I said, it stands for representational state transfer. What does it actually mean? It means that the state of the interaction between the client and the server, so there is some interaction, if you are on a shopping website, shopping carts, there is a notion of what is the present state of the interaction? Are you in the process of buying a certain object.

A lot of that information needs to actually be transferred back and forth between the client and the server. Okay, now, a shopping cart, of course, is different the client, the server actually retains the entire shopping cart at its end, in a database somewhere. But how to add something to a shopping cart? How to sort of, are you in the process of adding? Are you clicking on something and adding it to the shopping cart?

Are you removing something from the shopping cart, what is the present state of the shopping cart, all that information needs to sort of go back and forth between the client with every communication, the entire contents of the shopping cart do not, but the exact nature of the specific interaction that you have at this point has to sort of shuttle back and forth. And that state

transfer between the two, right is happening by means of some kind of text based representation. It not ultimate, it may not be text based, but as a lot of standardized representation format. And that, essentially, is what allows us to sort of build such distributed architectures in a robust way.

(Refer Slide Time: 1:53)



Sequence

- Client accesses a Resource Identifier from server
 - Usually URI - superset of URL
 - Typically start from home page of application
 - No initial state assumed
- Resource Operation specified as part of access
 - If HTTP, then GET, POST etc.
 - Not fundamentally tied to protocol
- Server responds with new Resource Identifier
 - New state of system; new links to follow etc.

State of interaction transferred back and forth

Okay, so a typical sequence of how a REST based communication could happen between the client and the server would be something like this. A client accesses a Resource Identifier from a server. So, what is a Resource Identifier? It is something that identifies, maybe a shopping cart, right? Usually, this would be a URI, a Uniform Resource Identifier, right, which stands, I mean, so URI stands for Uniform Resource Identifier, you are probably more familiar with URL, which is a Uniform Resource Locator.

The URLs are that line of text, which comes on the top of the browser, which is sort of you know, the address of the website, or the page that you are visiting. The URI is sort of a superset of the notion of a URL, right? It can refer to certain things which are not necessarily HTTP web pages. And it has a very specific, it includes a protocol as well as some kind of information about what identifies this particular resource.

And the way that the client would go about accessing this typically would be that you start from the homepage of the application, and in that there are a number of other links that tell it, what are the other resources that can be accessed from this place. Once the client has access to a resource in identifier, along with accessing the Resource Identifier, you also need to specify a resource

operation, what do you want to do with the resource? Do you want to just get it that is read it and you know, display it? Or do you want to put some new information onto it?

Do you want to, let us say, create a new blog post somewhere. Or post a comment on Facebook. Now, in the case of HTTP, that resource operation would typically be the verb. Something like HTTP GET or an HTTP POST. But the important thing to keep in mind is REST is not necessarily tied to HTTP, just so happens that HTTP is one good way, one good sort of platform that can be used for implementing these ideas.

And finally, so what is the client, then it has requested a Resource Identifier along with a type of operation, the server looks at both of those, decides what to do, identifies what it needs to send back and basically sends back a new Resource Identifier. When I say new Resource Identifier, this could just be basically a new page that comes back to the client. And the client then decides, what to do next.

So, in this way, look at what has happened. I mean, we have not really assumed any kind of state on either side, client sends a request, the server just looks at the request, it does not assume anything else, and responds based on that. Now, the server has other information. It, for example, might know from the request, basically, maybe there is a cookie as part of the request, which tells the server that yes, this client is actually logged in. And based on that I need to respond in a different way. All that is fine. So, that is not what REST is talking about.

REST is not saying you should not do that. That is perfectly fine. But what it is saying is at any given point in time, if this was a request from the client, and it included this cookie, it showed authentication and so on. This is how the server should respond. You cannot assume anything else about the client, such as the you know, that the client is currently on or that the user is staring at the screen, none of that is possible.

So, this state of interaction at any given point in time, do I want to add something to the shopping cart, all of that information, what is required over there needs to be transferred back and forth between the server and the client. Not all of it, I mean, I do not need to transfer the entire contents of the shopping cart, but sufficient information that I can update the state and get a new representation of where I am in the system.

(Refer Slide Time: 5:43)

HTTP

- One possible protocol to carry REST messages
- Use the HTTP verbs to indicate actions
- Standardize some types of functionality



HTTP

- GET: Retrieve representation of target resource's state
- POST: Enclose data in request: target resource "processes" it
- PUT: Create a target resource with data enclosed
- DELETE: Delete the target resource



HTTP is one possible protocol to carry rest messages. It is not the only one, but it is the most common. And what is done in this case is that the HTTP verbs are used in order to indicate actions to be performed, GET, POST, PUT, DELETE, most common ones. And certain kinds of functionality can therefore be standardized by having specific meanings associated with those operations.

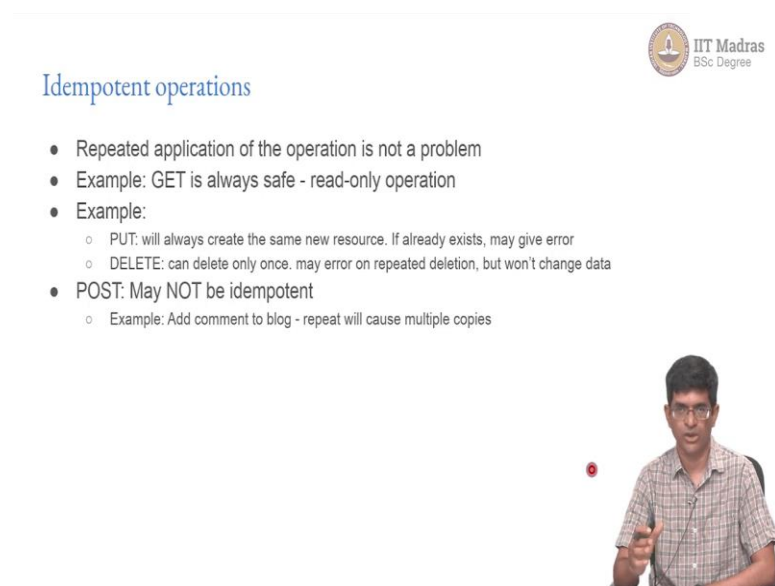
So, GET typically means retrieve certain representation of a target resources state. POST, on the other hand, will typically have some enclosed data, whenever you are doing a POST request on HTTP, it means that there is usually some kind of you know, a text box or something that you

have filled out and submitted or some other parameters that you have dipped in a form. And what happens is that you are effectively telling the server, the target resource, everything, every request to the server is actually a request to a specific target resource.

Because after all, the way that we have looked at it is any request that we are sending is a URI of some resource, which is there on the server. So, the way to look at this is the POST is also providing some input to that resource, which tells it how to update either create a new instance of something, or update some internal value, and return a new resource identify what should the user go to next, or what should the other side, the client do next after this?


PUT is sort of a bit more clear than POST it sort of says that, usually the understanding of PUT is that it just basically creates a new resource. So, assumes that there is nothing already in place, just creates space for a new resource puts all the data that you have sent over there, and sends back a success or failure, response. DELETE, easy to understand, right, just delete the target resource. So, these are all ways by which the HTTP verbs have been adapted for specific functionality within the context of REST.

(Refer Slide Time: 7:50)



Idempotent operations

- Repeated application of the operation is not a problem
- Example: GET is always safe - read-only operation
- Example:
 - PUT: will always create the same new resource. If already exists, may give error
 - DELETE: can delete only once. may error on repeated deletion, but won't change data
- POST: May NOT be idempotent
 - Example: Add comment to blog - repeat will cause multiple copies



There is also this notion of what are called idempotent operations. And this is useful to understand it basically says that repeated application of a certain operation is not going to cause problems. A simple example of that is that a GET operation is almost always safe, I mean, when you are when you are understanding of GET is just to read data. So, you can basically read

information as many times as you want, without causing problems, it does not change the server in any way.

So that is idempotent, you can do as many GETs as you want, it is not changing the state of the server. A PUT is also idempotent, it is changing the state of the server because it is creating a new element. But because of the nature of PUT, what it says is if you put an object once, or at least the way that PUT is defined, it basically says you are putting a specific object with a specific ID.

Next time, if you try putting the same object again, it will probably respond saying the object already exists. You cannot do this one more time. But it is not going to create an extra object. So that is, it is partly also how we define PUT. So, if we say that PUT basically has this meaning that I am going to put a specific object with a specific ID, then it is idempotent, because if you try putting it more than once it will just basically referencing, it is already in the database, you cannot do this one more time.

DELETE is idempotent, as long as you have given the ID to delete, let us say I say delete item number 42, it deletes it. It does not change the IDs of any other items in the database. So, next time you call delete Item Number 42, it might just tell you look, you already deleted it, it is not there but it is not going to change anything further. So, you can repeat the process without causing problems.

The POST operations in general though need not be important because very often what happens with POST is that we are seeing create or update this so if there is something which is already there, it might try to update it. But if you have not given an ID, it will try to create a new object with a new ID, so one possible thing, which you might even notice in certain cases is that sometimes there is duplication of comments.

You have a blog or something of that sort. And somehow, what happens is that you find that, you know, there are multiple comments with exactly the same text, what happened, a person basically clicked on it more than once they click the submit button more than once. There is really no way that the server could know that this was not supposed to happen. It received two submit requests of the form, it is not going to compare them and say, hey, these look the same.

It is just taking the data and deciding what to do with it, creates two comments with exactly the same information. So, POST, in other words, in that context, at least, is not idempotent. So, this notion of idempotent operations is useful to keep in mind, when you are thinking about the consistency of your application, you need to sort of, remember that the end user could potentially do that.

I mean, we all know that sometimes, the machine gets stuck. And we just go click, click, click multiple times, right? We are just furious with it, we wanted to do something, we click on a button more than once. That could be a terrible idea, if it is going to, basically charge you for 10 railway tickets instead of one that you wanted to buy. So, how do you prevent something of that sort from happening? Those are notions that you need to sort of keep in mind while designing.

(Refer Slide Time: 11:19)



CRUD

- CRUD - database operations
- Typically a common set of operations needed in most web applications
 - Good candidate for REST based functionality

REST != CRUD

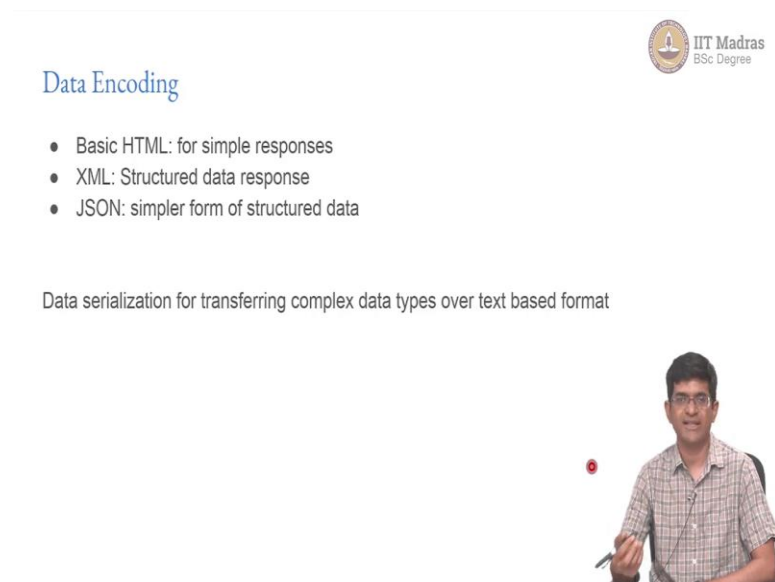
But they do work well together

So, finally, so, so far, we have got an idea of what REST is, it basically, it is that whole notion of representational state transfer, how it works, we will look at the examples later. But why it works is what I was trying to explain all this way. Now, one important thing to explain over here is, we already looked at CRUD, which is the Create, Read, Update, Delete set of operations. And these are database operations. Remember that they are not fundamental to the web, they actually come from the notion of a database.

But they are typically a common set of operations that are needed multiple times in most web applications, which means that they are a good candidate for REST based functionality. These

are actions that the user typically wants to perform. And what usually ends up happening is that CRUD operations are implemented using REST, but it is very important to keep in mind that REST and CRUD are not the same thing. It is just that they work very well together. So, you can use the notions of REST in order to implement CRUD functionality. But CRUD applies to the database, REST applies to the architecture of the web application. So, they are sort of talking about two different things.

(Refer Slide Time: 12:39)



IIT Madras
BSc Degree

Data Encoding

- Basic HTML: for simple responses
- XML: Structured data response
- JSON: simpler form of structured data

Data serialization for transferring complex data types over text based format

Now, in all of this, so we have all these things that can be done using REST. One important thing to keep in mind is also this thing called the data encoding. Normally, in a web server, when you make a request to the web server, you expect to get back an HTML page. But now that we have got this notion of REST, REST is no longer talking about HTML pages that are going to be seen by a human being, it is talking about more machine readable information.

So, I want to be able to transfer some state information. And how that gets displayed to the end user is a different story. You know, you could have some other clients that decides how to actually display it, I want this to be done in such a way that it can be transferred between machines in an efficient manner, which means that just HTML may not be the best way of doing it.

So, then people thought, maybe XML, Extensible Markup Language, yes, XML can be used, was largely used for a long time in order to transfer sort of machine readable data between systems,

right, it is a structured data format, which is more extensible than HTML. But at the same time, also, you can impose better constraints on it to prevent, like, the kind of badly written HTML would not generally be accepted in XML.

On the other hand, XML is also verbose, it is a bit difficult to sort of edit, and write, and so on, and even transfer around. And what came along at some point was this notion of something called JSON, Java Script Object Notation. The important thing is all of these techniques are basically looking for ways by which you can take complex data types and transfer them over a text based format in some way.

(Refer Slide Time: 14:27)



JSON

- JavaScript Object Notation
- Nested arrays:
 - Serialize complex data structures like dictionaries, arrays etc.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

IIT Madras
BSc Degree

So, the idea of Java Script Object Notation was that if I am trying to transfer, let us say, some complicated information, like the complete user record, right, I want to transfer all the information corresponding to a user because there was a GET request asking for details of a user. I do not, from the server side, I do not want to sort of format it into HTML and say, this should be the size of the font, this should be the location on the screen. I want to send the data and let somebody else take care of actually deciding the presentation part, the view.

So, the controller by itself does not want to get back HTML from the server. It wants to get back data, which can be fed into a view. That is the important part to keep in mind over here. And that data might be complex, it might have arrays of information, it might have a dictionary, which has like different hash map between labels and values.

An example of that would be something like this, let us say this is a user record, and what I have here, for example, is the, it starts off with starts and ends with curly brackets. So, this entire this thing is one Java Script object, it is a JSON object. And inside this JSON object, there are many different parameters out here. And if you look at it, what is happening is, it is the sort of looks like a dictionary.


In Python, we already know what a dictionary is, it basically has a key and a value. Now, even though this is JavaScript, this might actually even parse directly as Python code, or at least it is very easy to get something which will be able to read this and say, yeah, this can be converted into a Python dictionary. So, what would the contents of the dictionary be, it would have one key called first name, which would have the label John, another key called last name, which would have the value Smith, a key called age with the value of 27.

Notice that 27 is not within quotes, therefore, it is a different data type. In this case, it is a number or an integer, whereas John and Smith are within quotes, and therefore are strings. What about address, this is a key, what is the value corresponding to that? That is not a single number, or a string, even at this point, the value corresponding to address is itself a dictionary, which has street address and postal code as its contents.

Similarly, the value for phone numbers now uses the square brackets to indicate that it is an array, the first element of the array has a type home and number some value. The second entry also has a type and number value. So, in other words, this is an array of similar objects that are there inside. So, what has effectively been done as you can, it is very easy to think of this as a class definition or an object, it is basically the complete data corresponding to a given instance of an object.

And what has been done over here is, that whole thing has been serialized, meaning that it has been converted into one string of characters that can either be printed out or can be sent over a network protocol. So, this serialization of complex data structures is what is handled using methods like JSON.

(Refer Slide Time: 17:58)




API data transfer format

- Input to API: text - HTTP
- Output: complex data types - JSON, XML, YAML etc.
 - JSON most commonly used
- Different from internal server representation
- Different from final view presentation

YAML

- Yet Another Markup Language - common alternative, especially for documentation and configuration



And typically, what we find is that any API transfer format, whenever we are constructing an API, as we will see later, the assumptions we will make is that the input to the API is going to be some kind of a text format, like HTTP, the verbs in HTTP, the output could have been HTML, but it is actually more useful to have it in some format like JSON or XML, which is a bit more verbose, or there are other languages such as YAML, yet another markup language. Now, why are these things used?

Well, because they are compact, but complete representations of complicated data structures. The important thing is they are not the same as the internal server representation. What is the internal server representation, it is a set of entries in columns inside a database table. What is the final view presentation, it is going to be some nice looking screen with, colors and text of different sizes, and everything laid out nicely in an array.

This information on the other hand, which is actually being returned by the server, to the controller, so that it can pass it on to the view, is going to be in a neutral format, we typically tend to use JSON, or maybe one of the other variants, depending on what the conditions are, JSON is probably the most common at this point. So, keep that in mind what is happening over here, the model is returning this JSON data to the controller, which in turn passes it on to the view for display.

And that is how an API works. It basically does this separation between the model and the view, and allows them to communicate by means of the controller by transferring data in this format. Now, just one additional word I mentioned this thing called YAML Yet Another Markup Language. It is a very common alternative to JSON. It is used in a slightly different context. It is primarily used for documentation and for configuration, so let us say you have some kind of new application and you want to provide some configuration information for that.

When you want to start it up, you want to say how many servers should be started, what the website name should be, some information, some what is called metadata, about the application. YAML is very useful and effective for doing something like that. Why am I mentioning this because later when we get to the OAS, the Open API Specification, this is actually one of the formats that is commonly used to represent specifications.