

IIT Madras
ONLINE DEGREE

Modern Application Development -II
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Designing APIs-I

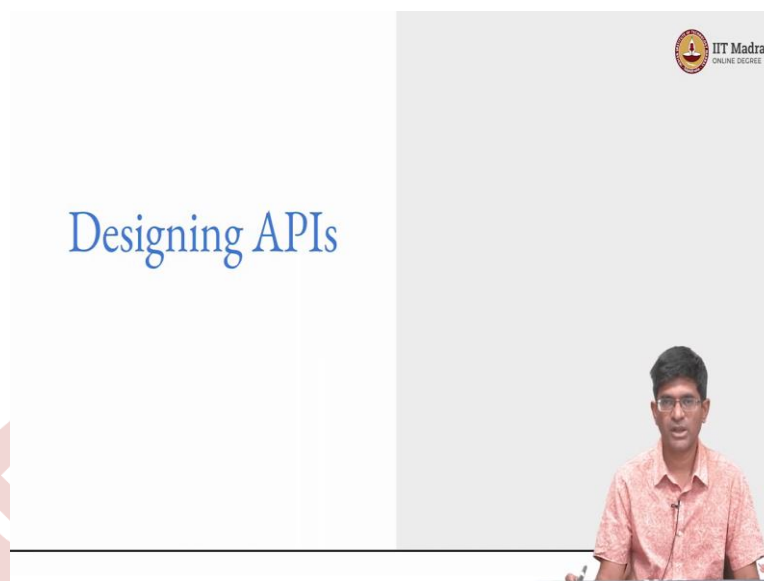
Hello, everyone, welcome to modern application development part 2.

(Refer Slide Time: 0:14)



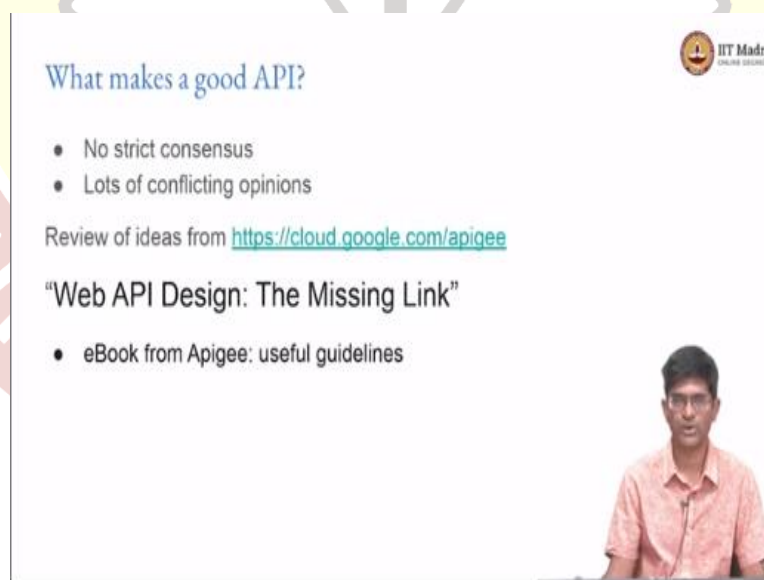
Now we are going to look at a review of a number of topics related to the development of web applications. This is meant more as a sort of summary of many of the topics that we have looked at in the past. And to sort of see what how we can bring all of those together and say this, these are good ways in which to go about designing and building web applications, and sort of summary of all the best practices that we have seen so far.

(Refer Slide Time: 00:44)



So, we will start with the topic of API's, Application Programming Interfaces, which we have already looked at in some detail in the past. But there are some aspects that you will be able to better understand and appreciate now that we have actually been through the process of building more complete applications, and also seeing other API's that have been designed that we can integrate with directly.

(Refer Slide Time: 1:07)



So, the question that we are trying to answer over here is what makes a good API or how should you go about designing an API in some sense? And the bottom line is, there is no strict consensus on this. There is no single answer to the question, what makes a good API or how should I build an API perfectly.

And unfortunately, if you look around online, you will find that there are a lot of conflicting opinions people, there are some people who feel very strongly that one particular way is the right way to design an API, there will be others who look at that exact same method and say, no, there are so many problems with it. And there is something wrong and you should do it in a different way.

In that sense, even the techniques that we have covered in this course, are not necessarily the best in some sense. Although, we try as far as possible to follow best practices, what are considered best practices, as I said, there is no consensus on those. Rather, there is only partial consensus on some of those. Which means that, if you find other techniques that are better suited for your style of writing, or for the application that you are working with, you should be open to looking at those and understanding why they better suit your mode of working.

What I am going to be doing over here in this video, is to review some of the ideas found in this ebook Web API Design: The Missing Link. It is by a subsidiary of Google. It was a company called Apigee that was then acquired by Google. And they have a number of very useful guidelines. It is been nicely written with a few different examples that sort of illustrate what does and does not make a good API. And what are the things that you could look for while designing one.

(Refer Slide Time: 2:49)



Purpose of an API

- To be *used* to build applications
- Design APIs with developers in mind

Remote Procedure Call : used to connect to a remote server and retrieve data based on some function arguments

Web API

The slide is part of a presentation from IIT Madras, as indicated by the logo in the top right corner. A small video inset in the bottom right corner shows a man in a red shirt speaking.

So, the first thing to understand before we get into what makes a good API is to understand what is the purpose of an API. Why are we sort of designing an API in the first place? And

the most important thing that you need to keep in mind though there is you are writing an API not as a program in itself.

Unlike, let us say, a program, like a web browser, or a word document editor or something to do a climate simulation, those are programs that have a functionality in themselves. You write a climate simulation program, just so that you have an answer to the question of what is likely to happen under these kinds of conditions. But, an API is not meant for that you do not write an API just for itself, an API is literally at the end of the day, just a set of functions.

Which means that its primary purpose is to be used by others. It might be something that you are yourself going to use. But very likely, if you are designing it as an API, there is a good chance that you also expect others to make use of it. Because if it was just yourself, you could probably have gotten by with something simpler, where you know all the functions that you are trying to write.

Even then, of course, it does make sense to think in terms of an API, just for the clarity, of design. But, the primary purpose of most API's is to be used by others. It may not be for public consumption, but it may be somebody else in your team. And you just want to sort of separate out your thought process from what the other person on the team is thinking of how a particular set of functions needs to be used. Which means that you need to design the API with the developer in mind, a person who is going to be using your functions and developing their own code.

Now, fundamentally, the API is an evolution of a much older concept called Remote Procedure Call or RPC. And RPC has been used for several decades now. It was initially done for like basic networking client-server kind of relationships. The bottom line is it is used to connect to a remote server and run a function out there. So, I am working on a client a, and there is a server called B and A wants to connect to B and run a function on B.

How should I do that? What are the standard mechanisms that I can expose on B? How should I connect from A to B? How should I send the parameters for the function? How should B send the response back to A. All of those are what are defined in the context of RPC, remote procedure calls.

Now, the relatively more recent development, in the sense that it is only about 2 decades old, rather than 4 or 5 is the web API. And in the web API, essentially, they said, we will take the same concept, I want to run a function on some remote server. And I want to be able to pass

arguments into that function. And I want to be able to get the response back from that function.

And what they said was one of the simplest network communication protocols currently in existence is HTTP. Why is it simple? It is text based, it allows you to send arbitrary request parameters, it allows you to receive arbitrary response bodies. So, pretty much anything that could be done in any kind of procedure call could be implemented in some way on top of a web based API, or web based request response.

And so they said, let us use this in order to build up our application programming interfaces, hence, the Web API. So that, in some sense, is the purpose of an API to be able to, first of all be used by others? For what purpose so that they can run functions on a remote server with certain parameters and get back responses. Once you keep that in mind, the rest of the idea of how do we go about designing an API, it sort of becomes a little bit more clear why people choose certain conventions.

(Refer Slide Time: 6:45)

Data oriented approach

Entities

- Students
- Courses
- Grades

Summaries

- List
- Grade point average
- Top students in courses
- ...

Actions

- Add, Edit, Delete

Exotic

- Students with names beginning with 'A' who scored more than 85% in the course

IIT Madras
ONLINE COURSES

Now, one of the recommended ways of looking at designing an API is to take a sort of data oriented approach. And we have been having this running example through this course, where we look at a grade book. So, there are students who are enrolled in courses, and who receive grades for the, their performance in the courses. So, all of those can be thought of as entities. They are the nouns or the objects that are there in the system.

Now, in addition to those entities, you also have certain actions. You want to be able to add entities, edit them, update them, delete them, if necessary, the standard CRUD operations,

write, create, read, update, delete. Now, those are actions, okay, that you perform on the entities, which means that the server gets updated.

And if you recall how we implemented all of this in flask, CRUD operations were ultimately boiled down to certain functions that are written in Python, inside the flask framework. And the only place where the web API comes into the picture is that there are certain routes on the web interface that are mapped onto certain functions. So, you can see how the remote procedure call has been set up at that point.

You have the function, which is running on the server. And all I need to do is get data into the function, the arguments into the function and get the response back from the function. And how do I do it, I set up a server in such a way that it can receive those requests as part of an HTTP request, call the function, codify the response into some kind of text along with certain headers and send it back as the response of the HTTP transaction.

So, the entities are the data model, the thing that is actually going to be stored on the server and manipulated on the server. The actions are the kinds of manipulations that are done. So, these are some of the manipulations you could probably think of others may be, joining to different kinds of entities or modifying them in certain other ways.

And after that, apart from these actions, which actually go and modify, mostly, we are interested in actions that modify the entities, there are also actions that are just retrieving data. So, those are sort of what we can call the summaries. They are only reading the data models and retrieving data, which is of course useful for your application.

For the student grade book, it might be something as simple as get the list of students enrolled in a course. But, I might be interested in slightly more interesting examples. That could be something like, well, what about the grade point average of a student, if I look at a particular student, I want to compute their grade point average. Or something even more interesting, I might be interested in who are the top 5 students in a given course.

Now, all of these ultimately are read queries. They are not modifying the database by themselves in any way, but are extracting certain information from the database and trying to pull out the useful context that we have over there. And then finally, we could end up with something which I can only call exotic queries. What I mean by exotic is, these are strange. These are not things that you can plan for.

But somebody suddenly comes up and says, can you give me the list of all students whose name begins with A and who scored more than 85 percent in courses that were offered in January 2021 and which had at least like three lectures. I mean, I am just unnecessarily complicating the question. The point is, these are not, this is not a question you can plan for. This is not a question for which you can have a ready made query sitting in the system, you still need to be able to extract the information if required.

And then you have to plan are you just going to say, look, there is no way you can get this information or do you sort of say, it is a complicated query. But, if you can structure it in a certain way you can get the information you need. The point is, at the end of the day, from the point of view of an API, even this exotic query is a potential function or a potential data retrieval that I might be interested in doing. And that is the important point that an API ultimately is just a set of functions that operates on data.

(Refer Slide Time: 11:04)

Possibilities

- List of students
 - <http://localhost/getListOfStudents>
- Individual student
 - </getStudent?id=xyz>
- Add new student
 - </createNewStudent>
- Edit existing student
 - </editStudent?id=xyz>

Handwritten red annotations: "URL" with an arrow pointing to the first URL, and "endpoint" with a line pointing to the same URL.

So, now let us look at how we could construct the URLs or the endpoints for these API functions, we have already decided that the functions themselves are running on the server, they could be implemented in flask, meaning Python, or could be Node js, or it could be go Golang or it could be Java, it could be a number of languages. They really do not care how the back end, so to say was implemented.

Why? Because we have abstracted that out by saying that the interface between the frontend and the backend is purely through HTTP. And that is one of the real big benefits of the HTTP of bringing the HTTP protocol for separating the backend from the frontend. I do not care how the backend was implemented. Because as long as it can talk, the HTTP protocol, it can

receive certain HTTP verbs and so on, and it will perform a computation, it will call the appropriate function, and give me back a response in the format that the frontend expects, that is all that I care, I do not care how it was implemented.

Which means that, I now need to sort of say that this is a URL. And this part of it the latter part is essentially what I will call the endpoint. And I could go and define a, an endpoint that says, slash getListOfStudents. So, the endpoint is basically the part that appears after the server name.

So, getListOfStudents is a natural sort of name that you might give for a function which retrieves the list of students? Is it necessarily a good name for a URL? Well, why not? There is nothing fundamentally wrong with it and that is an important point to keep in mind. But the next question comes, what happens if I want to get the details of an individual student.

Now, most likely, the Python function internally will probably be something like getStudent, and it takes a parameter, ID equal to something. Which means that I could probably write it out like this, getStudent and use a question mark, the query format that is used in HTTP, to pass an ID equal to something.

Now, the important thing that you need to understand over here is this entire query structure, the question mark, and ID equal to something all of that just gets passed in to the underlying program or the server. That is parsing this URL. As something called a query parameter which means that how the server actually then parses it and breaks it up into different parts, is pretty much left up to you.

If you think about how you assign routes in flask, we do sort of go about saying that, we would like to use slashes to separate the different functionality in the URL. But, it is not fundamentally necessary, I could have used a query based URL also which means that it could almost be I can use the exact name of the function that is being called internally, and pass in the parameters using the query string.

Now, what that means is, I would probably have another URL endpoint to create a new student, because most likely, there is a function called createNewStudent somewhere inside my Python code. And if I wanted to edit an existing student, I still need to give the ID of the student and call a function that would allow me to edit that information.

So, the important thing to understand or here is fundamentally, there is nothing wrong with this. You are doing a direct mapping of the names of the functions in your backend code to

endpoints or URLs that can be called by the outside. But there is some benefit in structuring things a little bit better.

(Refer Slide Time: 14:54)

Can get complex

- /getTopStudents
- /createStudentAndAddToCourse

Not fundamentally wrong!

- Hard to remember
- Hard to document, understand

Use Conventions

- List of students:
 - <http://localhost/students>
- Individual student:
 - <http://localhost/student/123>
- Add new student
 - POST .../student
- Edit existing student
 - PATCH .../student/123

So what happens is supposing I want to get the top few students in a class. Now I would have a new function `getTopStudents`. And then I want to have a function that creates a student and simultaneously adds them to a list of courses. `Slash createStudentAddToCourse` might even be one function that I actually write in my Python code.

Now, it is one thing to write the function in your Python code, it is another to expose each of these as a separate URL because what happens is, as part of the API, if I now have so many different functions, and I then give them out to the end user, it is very hard to remember. If I want to add a student, should I be calling just the `createStudent` or `createStudentAddToCourse`?

What happens if I do `createStudentAddToCourse`? Does it behave a little differently? How do I test these two functions to know that, if I create, if I call `createStudentAndAddToCourse`, but do not give any courses as input parameters, will it have the same functionality as `createStudent`? I need to be able to test I need to be able to document. And more importantly, the developer who is using it needs to be able to understand what I meant.

So, while this is possible, it can get complex. Therefore, API's sort of tend to follow conventions. And this word conventions is something that I am going to be repeating many times throughout this talk. And the reason is, because a lot of the information here how we go

about implementing certain kinds of URIs, endpoints, API structures, all of that is based on convention.

And the point is, there is a lot of benefit to following conventions. For example, when I walked into the building over here, I needed to go up to the third floor, I got into the elevator, and pressed an arrow, which was pointing upwards. I knew that or rather, I press the button that was marked 3. Now, if you think about it, that is a convention. Because it says that there are going to be buttons inside the elevator. And if you press the button mark 3, it is going to take you to the third floor.

Now a person who has never seen an elevator before, but gets inside, can probably guess what that is going to be. But a person who has used an elevator in Chennai, but now goes to Delhi and wants to use an elevator will have no problem at all. They do not even have to think they get into the elevator, they know this is the convention. Unless in Delhi, suddenly you press 3 and it takes you to the fourth floor.

Now, that is not going to happen. We know that. But if it did, then it would be very confusing to the person who is actually using it. So, that is where conventions actually have a lot of value. It is not just something for convenience. So, there is a difference between something for convenience and something which is a convention, which actually needs to be followed quite strictly.

When we say convention, what we are saying is, look, let us follow some kind of standard ways by which we construct these endpoints. So for example, slash students would be a list of all the students presented this system. An individual student would be directly identified as slash students slash 123.

If I wanted to create a new student, do not create a new endpoint or a new URL. We still have the HTTP verb that we have not used so far. Why do not I use POST, in order to create a new student? Let us say I want to edit the details of one particular student, instead of creating a new URL or a new endpoint for that, and not knowing whether it is update student or edit student. I just say patch and students slash 123, which basically means that I can take the existing information of the student and just modify some part of it. So, these are conventions.

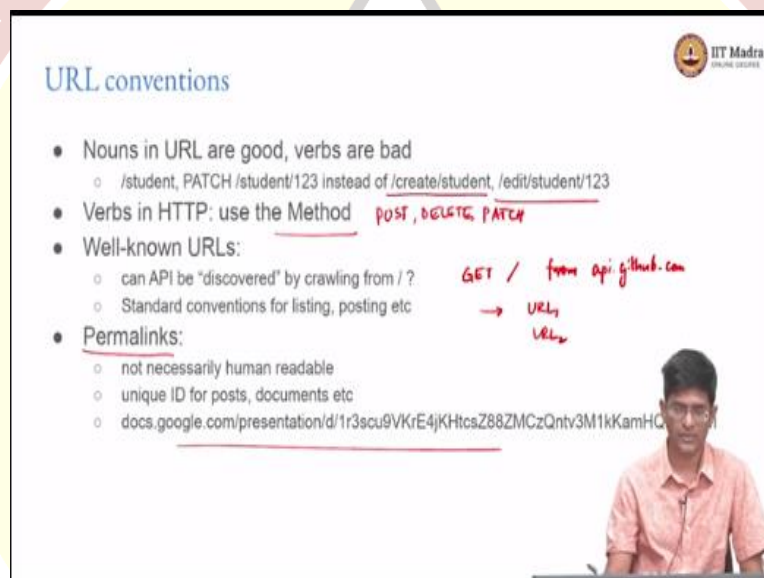
And this is part of what has got encapsulated and is nowadays popularly called part of being a restful API. Now, you are already been through the definition of rest and what rest brings to

API design. So, it is clear that you know, just using these verbs is not the point of REST, you can have all of those you can have CRUD, that is not exactly what is meant by REST.

REST is more a question of an architectural design style, it applies to more than just the create, read, update, delete operations. At the same time, it does not even require you to use HTTP ultimately, having said that, the most popular implementations revolve around using HTTP and therefore building web API's, which is why today to a large extent, I mean, it is almost synonymous, you say REST API, you assume that is for the web.

But keep in mind that they are not fundamentally the same. And potentially, you could have a different architecture on which you are using REST.

(Refer Slide Time: 19:44)



URL conventions

- Nouns in URL are good, verbs are bad
 - /student, PATCH /student/123 instead of /create/student, /edit/student/123
- Verbs in HTTP: use the Method **POST, DELETE, PATCH**
- Well-known URLs:
 - can API be "discovered" by crawling from / ? **GET / from api.github.com**
 - Standard conventions for listing, posting etc **→ URL, URL**
- Permalinks:
 - not necessarily human readable
 - unique ID for posts, documents etc
 - docs.google.com/presentation/d/1r3scu9VKrE4jKHtcsZ88ZMCzQntv3M1kKamH...

So, a little bit more on URL conventions. A general sort of rule of thumb is that nouns in the URL are good. This is once again related to the data entity oriented approach. We are sort of focusing our entire API naming method NISM around the idea of entities. And generally speaking verbs are bad. What I mean by that is, rather than trying to give things like create student, edit student and so on, avoid having the verb directly as part of the URL, instead, use the method.

Use POST, or DELETE, or PATCH. Each of these can be used in order to achieve some kind of functionality. Those are standard verbs defined in HTTP. In principle, you could define your own work, though, that is probably a bad idea. For the same reasons that we have been discussing before. It breaks convention, it makes it harder to document your API, people are

not used to it. So they would not know how to go about using some new fancy thing that you have introduced over there.

So, these two things are sort of a nice way of combining the noun, that is the entity with the action, the verb. And by combining both of these, you can get fairly nice URL structures that are easy to understand. And once you understand one of them, you find that, other API's are also easy to interpret in the same way. There will be certain things that you expect them to do. And hopefully, when you actually try certain things, yes, they do exactly what you expected of them.

Now, it is also probably a good idea, in a lot of cases to have certain well known URLs, where the entire API is possible can be sort of discovered by crawling through the links, what is crawling, crawling essentially means you access one link, you find out the links that are listed under that particular data. And from there, you then proceed to other links in the system.

And, in fact, this is one of the things that is done quite often, right, I'll be discussing a little bit more about this later when we talk about including links in the responses. But it could be, for example, that a GET slash from something like let us say, api dot github dot com, will give you back a number of different URLs.

And these URLs in turn, are going to be useful information. Either they will give you a list of users, or they will give you a list of projects or public repositories, things of that sort. Essentially, in other words, the system is, in some sense, trying to be self discoverable. You go to the main page of the API without really knowing what is possible over here.

And it gives you back a set of links saying, look, maybe you want to try one of these. Okay, and not just that, when you see that one of the URLs that is returned by GitHub, say something like slash user, you can straightaway look at the name and say, this probably gives me more details about the list of users in the system. And similarly, something about slash projects probably gives you more information about the project that are there.

So, in other words, this idea of discovering the capabilities of an API, not just through the documentation, but also through the embedded links that are there in the responses is also a powerful way of creating things that are going maybe a little bit beyond what you normally expect in any API.

And one other thing, which is something to keep in mind when you are designing these URLs, right. So you have things like all this slash student and so on. There are you also want to have some way of expressing what are called permalinks, permanent links. A permanent link is something which is a direct way of getting to a particular entity.

So, let us say that I have an entity or entry in GitHub and I have some way by which I want some way by which I can directly get to my profile page, my data page over there. The thing is, GitHub allows me to change my username. So, if I try giving github dot com slash user slash my username, it is not actually a permalink. Because at some point, I might change my username, of course, that will break a lot of other things that people are interested in, but it will also break the link to my profile page.

On the other hand, I know that I also have a unique ID inside GitHub which I do not usually use, but in fact, if you go to the API, you will find out what your login ID is. Now, based on that login ID, it is possible to create a unique permanent link, which will always work, even if you change your user name, or something else happens in GitHub. Unless of course, GitHub themselves change the way that they are handling things, which is unlikely. The whole idea is they are trying to develop permanent links.

So that right, but as long as you use the ID and not the name, you can be reasonably sure that even if the person changes their name, you will still be able to get to the same URL. You will notice, for example, that Google docs has this horrendous looking URLs. And why do they do this and not just have something which says, this is the user name slash, this is the title of the presentation.

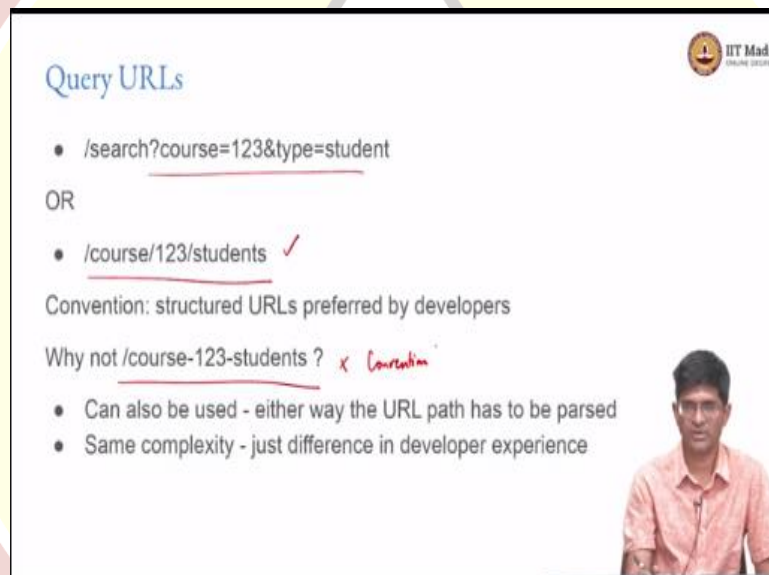
The point is exactly the same even now. That it might be changed later, maybe I am working on a presentation, but after having updated a little bit, I find that you know, the title I gave to start with was not really making sense. Let me change it. What should I do go and change the link. Similarly, I create a presentation. And after some time, I say, I am done with this, I am going to transfer ownership to someone else, how can you go and modify that in the URL.

On the other hand, if you use some kind of a link like this, and the reason why it has this long string over here is it is using some kind of a globally unique identifier, plus some other probably information that is encoded into this. But, the point is, this is guaranteed to be unique across all documents ever made by Google, which means that I can change the document, I can change the ownership, I can change the title, I can change the colors, everything about it, but I will still be able to reach the document by using this link.

So, you do not want to use this as part of something that is seen by a human being, but on the other hand, you probably will have some way of aliasing it. Meaning that I should be able to say, docs dot google dot com, some other way by which I can provide a URL, which will actually get me to this presentation. And without me having to remember all of this.

In the case of Google, of course, they do not even bother, they say, you want to get to your presentation, first log in, go to the main page, then follow the folder structure until you reach your presentation. And if you think about it, that folder structure that you have to navigate through the folders one by one is precisely a sort of nested hierarchy. Some way of you can think of it as a URL structure, Google has not implemented it that way for directly accessing your files. But, it achieves the same thing on a regular file system.

(Refer Slide Time: 27:03)



The slide is titled "Query URLs" and features the IIT Madras logo in the top right corner. It lists two URL examples: `/search?course=123&type=student` and `/course/123/students`, with a checkmark next to the second one. Below the URLs, it states "Convention: structured URLs preferred by developers" and asks "Why not /course-123-students ?" with a red 'x' and the word "Convention" written in red. At the bottom, it lists two reasons: "Can also be used - either way the URL path has to be parsed" and "Same complexity - just difference in developer experience". A man in a pink shirt is visible in the bottom right corner of the slide frame.

Now, you could also have other kinds of query URLs. So, rather than trying to say, I want to find out all the students who are there in course, number 123, I could perform a search, because if you think about it, at the end of the day, inside your Python function, you are still connecting to the database and conducting a search. There is an SQL query that is getting triggered.

And the query basically says, get me, from course equal to 123, all entities of type equal to student. So, why not just encode that query directly in your search URL? It could be done. But on the other hand, URLs, like the structured URL, where it says slash code slash 123, slash students, is preferred by developers, it is just easier to sort of for a human being to understand this. And to write it out and say, this is a standard way by which I can retrieve these URLs.

Ultimately, you need to remember that, you know, you are still calling a function that is performing a query at the backend. But why should the developer, the end user have to worry about the kind of query that you write? Because they do not know what database you are using? They do not know it, they do not even know whether it is an SQL or one of the No SQL type of databases.

So, rather than being concerned about how to construct the query, can they just focus on a URL and get the information they need? Now, one question to keep in mind or here is why not use something like this course dash 123 dash students? The important thing to understand is there is nothing fundamentally wrong with this. You could do it. Because after all, like I said, that entire thing slash blah, blah, blah, course whatever something students is just being passed as a string into Python flask or whatever server you are running.

And the server then has to parse it up, break it up and say, these are the components and this is the function that needs to be called. Could you have written your routes in a different way? Yes, probably. The point is, once again, convention, it is preferred to write something like this, rather than something like this. And this is primarily a matter of convention and not fundamentally of correctness. You could have written a URL this way it would work. But, it is not going to be easy for other people to understand what you mean when you do things like this.

