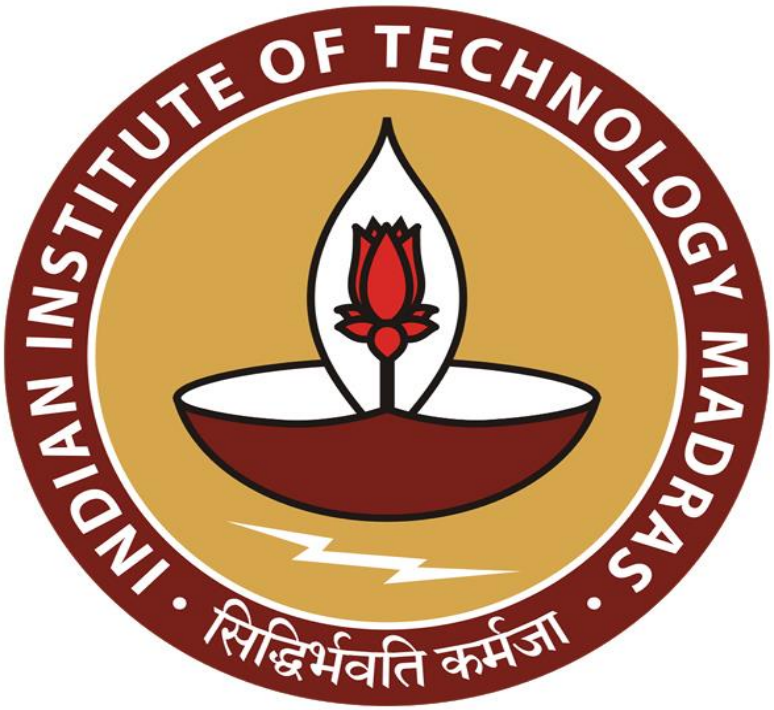


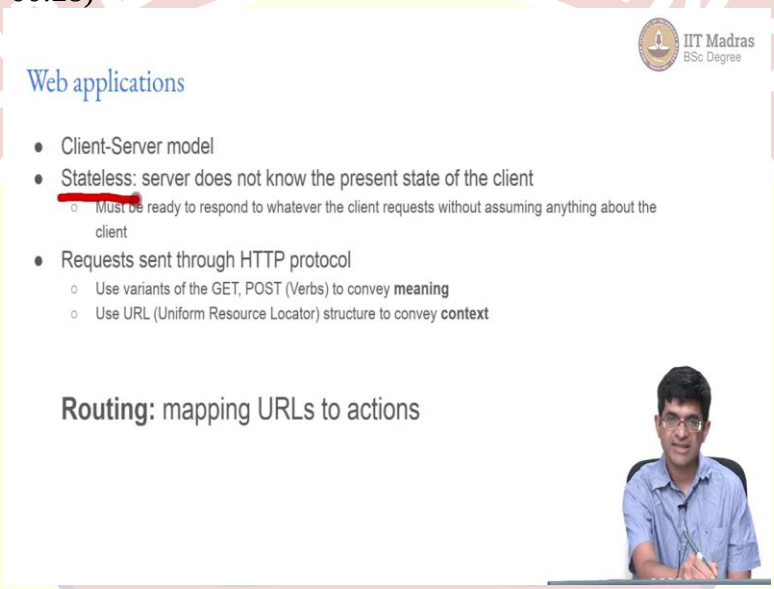
IIT Madras

ONLINE DEGREE

Modern Application Development – I
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Routes and Controllers

Hello, everyone, and welcome to this course on modern application development. So, coming to the last part of our discussion on controllers, how do these controllers actually get implemented in practice in a web application? That is where the concept of routes comes into the picture.

(Refer Slide Time: 00:28)



Now, as I have mentioned, multiple times, when the applications follow a client server model, and one very important aspect to keep in mind over here is that, this is stateless. Meaning that, the server cannot possibly know the present state of the client, a way of thinking about it is you connect to a server, your network connection breaks, the server cannot assume that you are still connected, or that your screen or showing a certain object.

The next time that you connect, you will have to once again make a certain request to the server. And the server should be able to respond without knowing in what state you were before that. Now, in practice, of course, you have probably come across many websites, that sort of when something like this happens, they break, and they sort of tell you that, oh, something went wrong, please refresh the page, or do not refresh the page, just go back to the beginning and start all over again.

Yes, a server could potentially just tell you that, go back to the beginning and start all over again, maybe because you were in the middle of a form and something did not get filled. But, the ideal kind of design should be such that it at least never breaks it. It should not sort of say that, oh, you sort of stopped filling out a form midway through, therefore, the entire thing is broken, you cannot fill the form, again, from scratch.

At the same time, the one that you have got half-filled you cannot go back and edit it, that is a bad design. So, those are things that you need to keep in mind. The server should ideally be stateless. If something happens to break the connection between server and client at any point in time, it should be possible to recover cleanly from it. So, in other words, the server must be ready to respond to whatever the client requests, without assuming anything about the state of the client.

And then the other thing is, that all requests in the case of web applications are sent through the HTTP protocol. What do I mean by that? The HTTP protocol is a text-based protocol. We already saw that, you basically connect on port number 80, or in the case of HTTPS port number 443, of a web server. And you send certain information, which is basically text, I can actually connect to that port and, type information there and it will go through, and the server will respond.

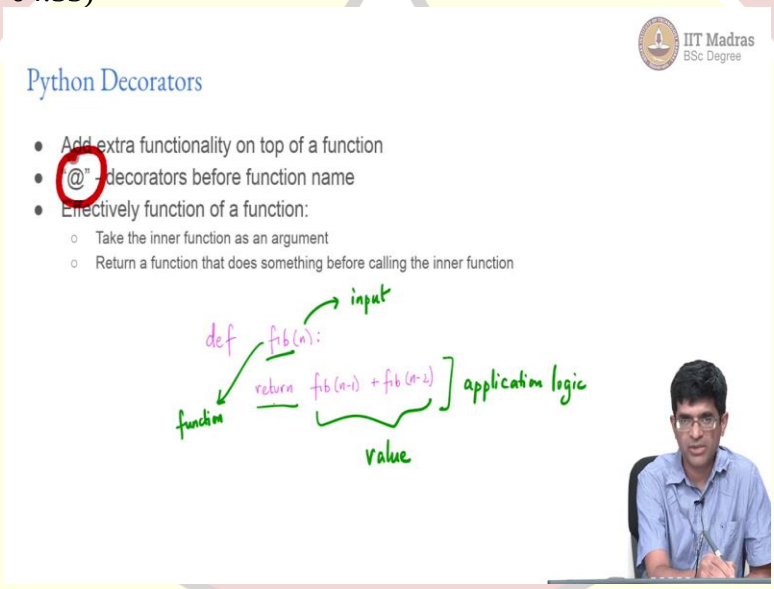
So, that text that I am sending is usually, a verb, which says, what kind of request I have, plus some URL, some information about, which gives me more detail about what I am requesting. And the verbs are the most common one, of course, is GET. But there is also POST, which is probably a close second. I mean, it is used in a lot of cases. They convey meaning, what it is that you are trying to do.

But the rest of the URL, the Uniform Resource Locator actually conveys the context. Going back to the example that I showed earlier about, the swayam courses, the context, which course am I in? What am I looking for? Am I looking for an assessment? Or am I looking for a lecture? Or am I looking for an about page? Is conveyed by the URL. So, was it a GET or a POST? Was it that I am, I have finished a quiz, and I am posting the solutions to that, I am posting my answers to that on the quiz?

Or am I just getting a new page, because I want to see what the next page contains. That is the, the verb, which conveys the meaning what it is that I am trying to do. And the URL conveys where I am trying to do it, the context. Which means that, ultimately, anything that implements actions under this framework, has to have a way of mapping URLs to actions. So, this whole business of how do I take a URL, split it up in a certain way.

And say that this URL, this part of this request, should correspond to invoking a particular action and getting some results back. That is ultimately where your entire web application design comes in.

(Refer Slide Time: 04:33)



Now, in order to understand this in the context of flask, which is what we will be using. There is one additional piece of information that you need to know a little bit about. You can get by without really knowing how it works, but you should at least know what it is, which is the idea of a Python decorator. Now, what a decorator does in Python. By now you should all be familiar with the idea that I can use the def keyword in Python in order to define a new function.

So, if I give def something, and whatever, some function name, so if I go back to this, def fib(n), something like this. Clearly there is a broken function, but does not matter. I am not trying to go for correctness over here. This is a function, what does it do? It takes this is the input, this n, over here is the input. And this is the function name itself.

And the functionality that we are dealing with essentially corresponds to some, this is the sort of, well, we can call it application logic, or, different forms of logic, depending on the type of what you are trying to do. Now, in this case, what is it returning? It is returning a value, which, if we look at it, we can probably infer is an integer. Because at some point, finally, this is going to sort of collapse into some kind of integers.

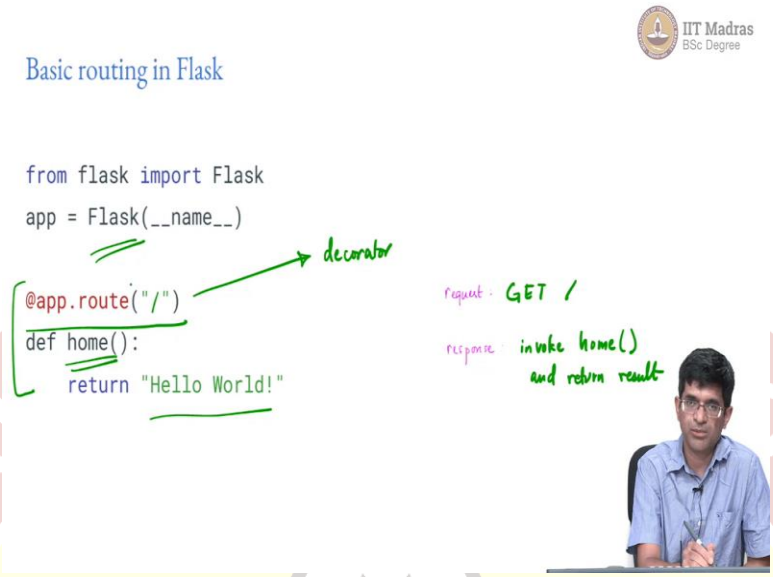
In this particular case, of course, it is a recursive function without a base case. So, now this does not really work. But in general, the Fibonacci function would have been returning integers. The point is, it is going to return a value. But I could also have another kind of function, which in turn returns a function itself, as its return value. What does that mean? It basically means that, the, when I invoke that function, the top level function, the results that I get back from it is itself a function that can be applied in a different ways.

That is basically what a decorator is trying to do. And the decorators in Python, you would see are implemented with an @ symbol before the decorator name. We will look at examples of those going forward. But the important thing to keep in mind over here is, all that a decorator is doing is it is adding additional functionality on top of a function. So, one way of thinking about it is to say that I have a function, which does a certain job.

I want to add certain extra functionality on top of it, I put a decorator in front of it. So, what the decorator does is it says, now if you call this inner function, it is going to basically first do something else. And then get back to doing whatever it is that you have written inside this code. It is not a trivial concept to understand.

But it is worth sort of thinking about it a little bit and understanding a bit of what it does. I am not getting into that in this particular context, because it sort of takes us away from the idea of understanding routing. But why it is useful or here is, because we are going to use that in flask in order to actually implement route.

(Refer Slide Time: 08:14)



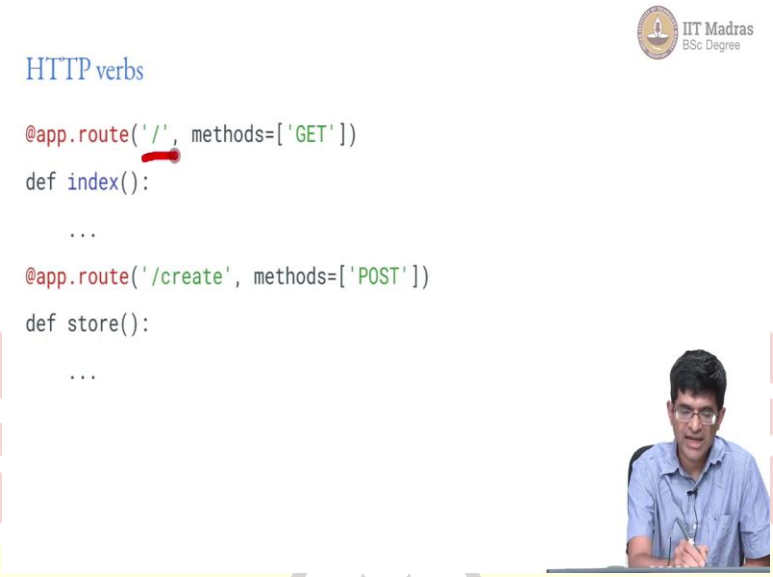
How does it work? This is an example, a very simple piece of code in Python, which basically uses the flask libraries. So, basically, there are some standard parts over here, this is the basic flask import. And then you sort of say app is equal to flask, blah, blah, blah, and then eventually, there will be an app dot run somewhere, which I have not shown. And this is the important part, this here, is the decorator, the app.route.

And what is this app.route decorator doing is, basically saying that any time that you call this home function, it is essentially going to first have this thing, this routing, which basically applies that, there is this whenever there is a call to slash(/) that needs to happen, and then call the home function. Now, what that means is, in turn that when the flask the model, which is, the inner running of the application.

App is equal to flask of name and app.run. When the app.run happens, it basically takes all of these app.route information's and creates a table out there, which says oh okay, now I know that anytime there is a request for slash(/), I need to call the function home. So, in other words, GET/, this would be the request. And the corresponding response, would be invoke the function, home and return the result.

Which in this case, of course, it is just hello world. So, where does this exactly happened? because you put in this decorator out here, that in turn would happen inside the flask object. Then the flask object runs, it sees this decorator app.route. And it brings a mapping, saying that slash(/) should get mapped to the function home.

(Refer Slide Time: 10:24)



So, how do we sort of take this further? What it means is that, because I can do a mapping, as simply, as just doing app.route, I can now start putting various different kinds of maps out there. I can say that, if there is a request to slash(/), using the particular verb GET, then call the function index. The interesting thing about this is, it says that, if I call slash(/) with post, it actually does not know what to do.

So, rather than sort of giving you back the same function, the flask application should actually sort of die at that point saying, I do not know how to handle this. You are asked for post to slash(/) and I do not know what to do. So, it is not going to be like a regular web server that tries, best effort to understand what the user may have had in mind. No, over here, you need to be a bit more clear. And the reason for that is very simple.

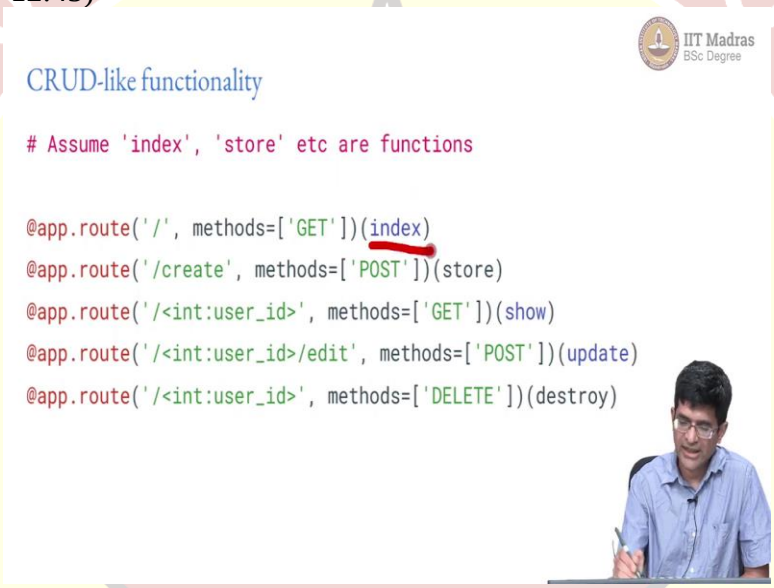
When you are building an application, you cannot assume that your users are friendly. So, to say, you have to assume that somebody will try to break your system. Which means that, it is probably better to say I will accept only GET requests, I do not want POST requests. If you send a post, I am just going to, say I do not know how to handle this. And maybe, rather than giving an error message on the screen, that there could be something inside the app which says, show a valid page. Which basically says, look, I do not know how to handle that, or I do not know what you are trying to do?

Now, what could I do instead, I could also take the route slash(/) create. Now I say, if I POST to slash(/) create, then store the information. You remember the example of Laravel, where I told

you that if there was a post, to slash(/) photos, slash(/) create, or whatever, do this. And that is essentially what it is saying. It is saying, this was a POST request, which means it probably should have had the file contents inside it, take that information and post it into the place, store it somewhere in a database.

So, these HTTP verbs, the get post etc, can all be individually be defined. And the most important thing over here of course, is that we are using the routing, the mechanisms within flask in order to decide which function gets called based on the URL.

(Refer Slide Time: 12:45)



Which means, we can now start doing more fancy things. Now, if I assume that, the index, store, show, update, destroy these are all names of functions that have already been defed before, defined somewhere before. I can also do the decorator in a, in a different way. I can basically call the app.route over here. One possible error is, since you are now not calling it at the function definition stage, I may have made an error out here.

And I think you do not require the at symbols, but the basic idea still holds. What you are trying to do is, take the function definition out here and add some functionality on top of it, basically the routing. And the app.routing, what it says is, if it is slash(/) with a method of GET, call the the function index. So, when the flask app starts up, it sees this call out here and it builds a mapping from slash(/) to index. Similarly, it builds a mapping from slash(/) create to this store out here. But only for the method POST.

And now, this is more interesting, I can actually straightaway ask for slash(/) and this is not a regular part of something that you will see in a HTTP request. But flask understands it differently. The moment that you have written something like this in flask, it says, if what follows slash(/) is an integer, then treat it as a user ID. And if the method was GET, then call the function show and pass this particular value into the function show as the parameter called user underscore(_) ID.

So, a lot of information basically packed into one simple line of code out here. But, the sort of underlying piece of information that you are trying to convey is very simple. There is some functionality which has been defined in terms of your Python def functions. There is a method that you want to use to get to that function. In other word, how do I invoke that particular action or that particular controller?

And this basically gives me the route mapping, which basically says, if this is the URL, if this is contained in the URL, invoke this function with this particular ID (())(15:16). You can get a bit more fancy with it. And basically, say that, if I have a slash(/) edit, after the user ID. It will straightaway call an update. But now, the method should have been a POST, not a GET. So, if I call this slash(/) edit with a GET, it will most likely tell me I do not know how to handle this.

If I call just slash(/), one, or slash(/) 42 with POST, most likely, it will again, not know what to do. But if I call slash(/) 42, slash(/) edit with POST. It will call the update function on user ID equal to 42, with whatever parameters have been provided in the post information. And the last one you can see is, look at these two. Both of them correspond to the same route, slash(/) int colon user ID, this is also slash(/) into colon user ID. What this says is, if the method is GET then called the function show, but this says if the method is delete, call the function destroy.

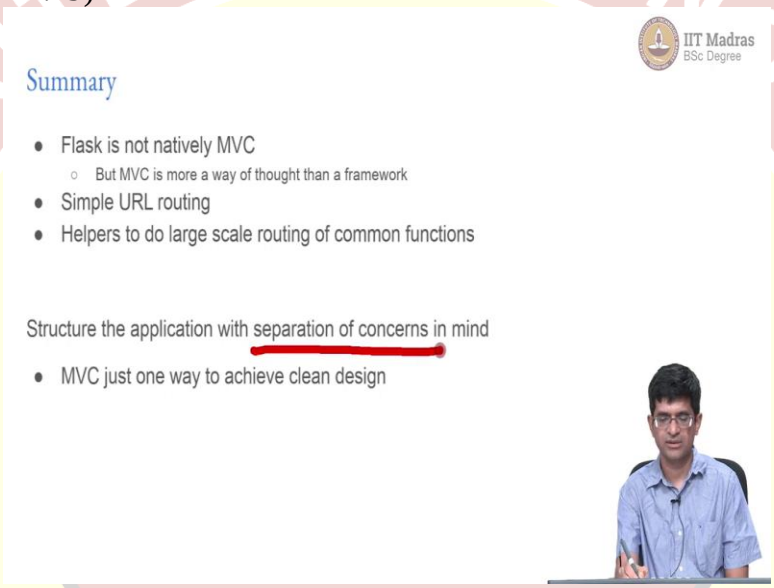
As you can imagine, this would just show the object, this would delete it from the database. Now, obviously, you do not want something trivial like that to happen just because I typed in the wrong verb at some point. So, in most cases, whenever there are destructive operations, things that can change the database, either update information or delete information.

There are ways by which you can request either, a require either authentication, or some extra, some tokens or some key something which basically authenticates that the user actually wanted this to happen. And it is not just, somebody on the net, finding this URL, and just, calling delete

on everything that they can see. You need to do that. And that is something very important to keep in mind when you are actually designing an application.

Because if you do not have that, you create a web application, somebody just calls a URL, slash (/) delete, and everything is gone. So, that is not something that you want happening, there are ways to get around it. And those are all part of good practice, rather than being fundamentally part of how flask implements anything.

(Refer Slide Time: 17:29)



So, to summarise, flask, as a framework, is not natively MVC, model view controller. There are, I mean, if you search around online, you will find either on Stack Overflow, or like various tutorials in different places, which say, how to implement MVC using flask. And under each and every one of those links, you will find like at least a dozen people complaining about it saying flask is not MVC, what are you trying to do?

So, do not get too hung up about it. You have to think of MVC more as a way of thought, separation of concerns. We are, at the end of the day, this concept separation of concerns is fundamental. Why is it fundamental? Because, it is just an aspect of clean design, by keeping the data model separate from the data view, and by ensuring that communication between the two happens only through explicitly defined controllers, you can prevent sort of unnecessary or bad modifications from happening to the data.

And that ultimately, is what we are trying to achieve. So, that they are always able to keep our data model and our views in a valid state. And thereby make it easier for us to, take care of various other things such as consistency, and the fact that, the server operates in a stateless mode. All of those things become easier when you keep this kind of a clean design process in mind. And separation of concerns is one of the ways of achieving it.

So, MVC is more of a way of thought, rather than a specific framework or a specific way of doing things. In that sense, flask can be used in order to achieve most of the goals of MVC. How is that? Because, it has a very simple URL routing structure. And it also has certain helpers, which allow you to sort of do overall controllers, logically related groups of functions can be on mass mapped to certain kinds of functionality.

So, without having to write out each and every one of those lines of app.route, app.route, you might be able to do it in one shot by using certain kinds of template functionality, that is present within flask. And that is basically where it, the power of the lang, the framework comes in. It is simple enough that it stays out of your way most of the time, but it is also powerful enough that it allows you to do certain things provided that you have thought it through, and your design process is clear.