IIT Madras

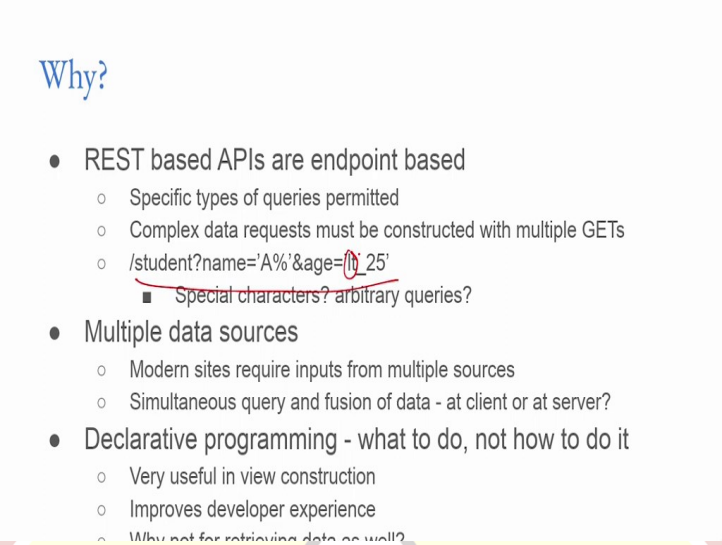
ONLINE DEGREE

Modern Application Development - II
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology Madras
Introduction to GraphQL

(Refer Slide Time: 0:14)

Hello, everyone, welcome to modern application development part 2. We are now going to look at the topic of GraphQL, the graph query language. Now, this is going to be a very high-level overview, in particular, I will not be looking at even the syntax of GraphQL, much less talking about examples or actually showing you a demonstration of it, this is more looking at why GraphQL? What are the problems it is trying to solve and how it approaches that problem?

(Refer Slide Time: 0:40)



So, the first question why, and we have already seen that REST based APIs are endpoint based, what that means is specific types of queries are permitted. Let us say that you want a list of students, you can probably just have an endpoint saying slash students. You want to get details of a student, there will probably be an endpoint slash student slash 123.

But, if you have a complex request, get all students above the age of 21, with names beginning with A and who reside in Andhra Pradesh? How do you do that with the REST API? There is no sort of straightforward way that you can construct such arbitrary queries. So more likely, what would happen is you would have to send the entire query across, or rather just get the list of students from the server and then figure out how to sort it out at the client end, or go and modify the server so that it can take such complicated queries?

Then, of course, the question becomes how do I specify such queries? I mean, can I construct a URL that looks something like this? Possibly? But now look at this, why am I using lt over here? Because I do not want to use the symbol less than because that is a special character for HTML.

So, how should I even construct this? Is this the right way? Are there better ways by which I can construct URLs? Should I be even using quotes? I guess not, I should probably encode them in some way. How do you make arbitrary queries? So, that is one big problem with regard to REST based APIs.

The other question is, very often in the design of modern websites, we are dealing with input from multiple sources? It could be as simple as saying that your user information is coming

from one source, let us say that I am using something like you know, google authentication, which means that the user database, well, you will probably have your own copy of a user database, but you might be getting it from one particular server, which is shared across multiple different applications.

I might, along with this, I might have something you know, maybe I am putting up a blog post. But suddenly, I have something in the blog that says the latest headlines of the day, or the weather in Chennai, at the moment? So, I need to be able to fetch headlines, I need to be able to fetch something from a weather API? In other words, a modern web app is very likely to have inputs from multiple different data sources.

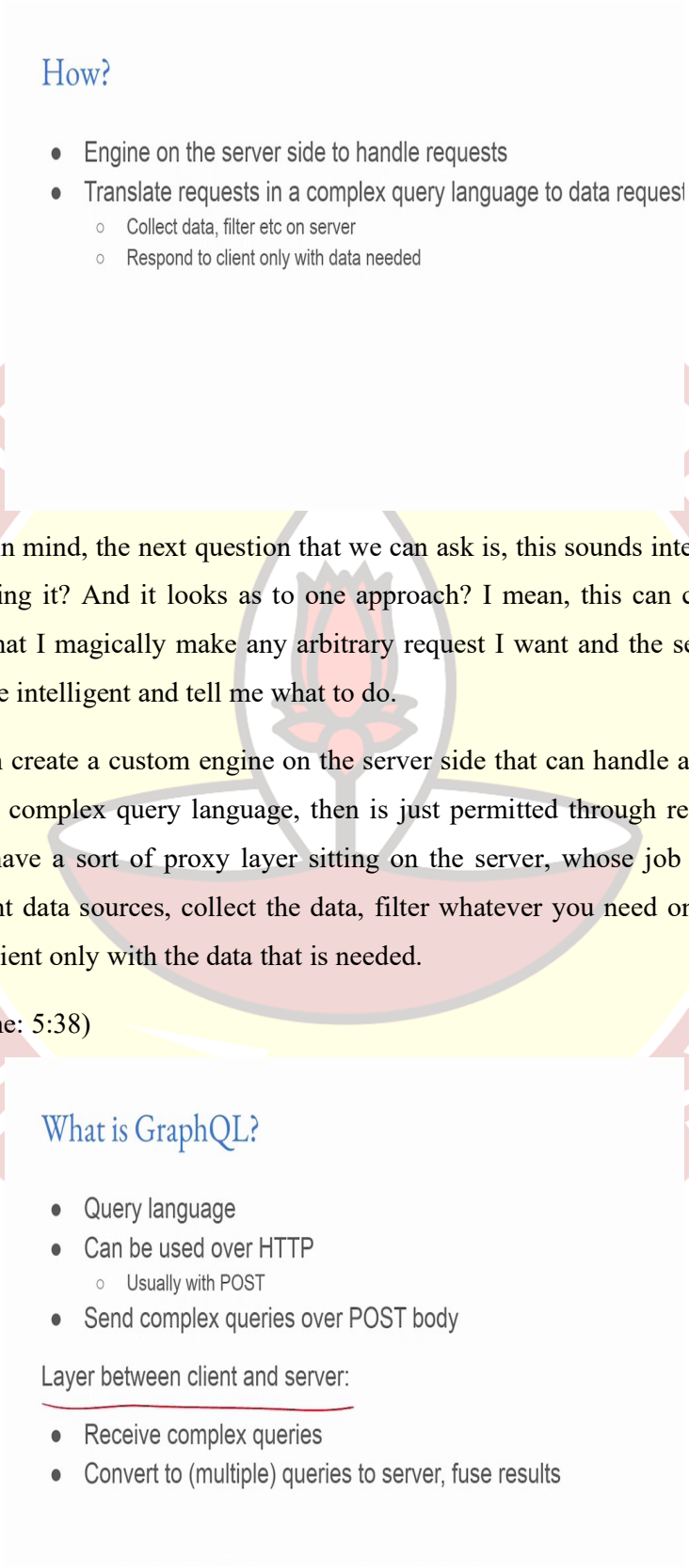
Of course, you could just issue independent queries to each one of them. But supposing you wanted to have one entity out there that basically, is trying to collate information from multiple such sources, and get you a common piece of information that needs to get displayed. Can you still do it?

You probably either need to have all the individual queries performed, and then handled at the frontend, maybe your JavaScript or view or the logic that is written in view, would take care of filtering out and showing only what is needed. So, it can be done. But the question is, can you actually tell the server in some way, and let the server sort of handle that complexity and give you only the information that you need, that is the ideal situation.

And the third sort of interesting point to keep in mind over here is we have already seen the value of declarative programming. So, in the context of things like view, where I could just specify what I wanted the page to look like, and the framework would figure out how to build it from me.

So, the fact that I want a binding between a particular variable and something which shows up on the screen, or if I want to have certain components that are constructed and you know, composed in a certain way, I just specify that composition and the actual construction and laying out of those things is handled by the compiler, essentially, something which manages the templating and generation of HTML. This improves the developer experience; it makes it much easier to use. So, can we apply something similar for retrieving data as well?

(Refer Slide Time: 4:48)



With all of this in mind, the next question that we can ask is, this sounds interesting. How do we go about doing it? And it looks as to one approach? I mean, this can come for free. It cannot just be that I magically make any arbitrary request I want and the server is going to suddenly become intelligent and tell me what to do.

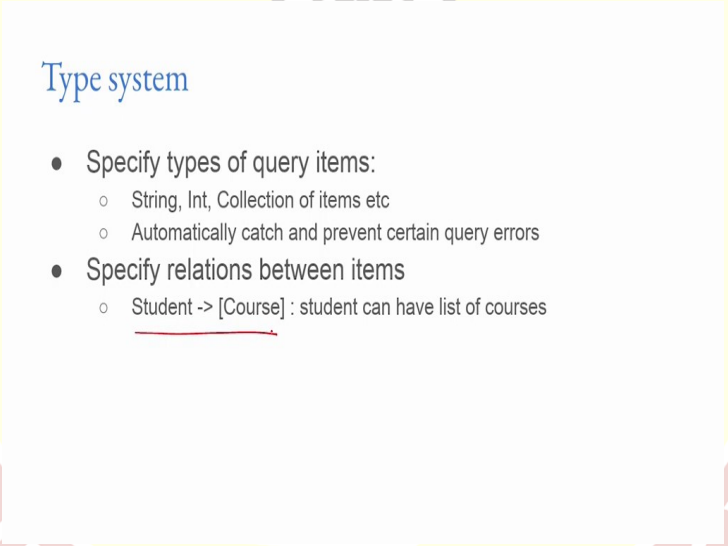
But maybe I can create a custom engine on the server side that can handle arbitrary requests in a much more complex query language, then is just permitted through regular URLs and APIs. So, you have a sort of proxy layer sitting on the server, whose job is to connect to multiple different data sources, collect the data, filter whatever you need on the server, and respond to the client only with the data that is needed.

(Refer Slide Time: 5:38)

That is essentially what GraphQL aims to be, so it is a query language. How does it function? Well, they once again, built on top of the web concept, and they said, look, let us use HTTP. But now I want to be able to send sort of more complex queries. So, rather than trying to encode everything into the URL, let me just use POST and use the POST body in order to contain the complex query that needs to be sent.

Which means that the GraphQL engine, so to say, is going to sit as a layer between the client and the server. It is probably sitting at the server end, which means that now as far as the client is concerned, it now connects to the GraphQL server instead of directly to the backend. The GraphQL server then retrieves or other receives all these complex queries. And it converts it possibly into multiple queries to either one or different servers, fuses the results together, filters, and returns only the data that is required.

(Refer Slide Time: 6:44)



Now, GraphQL has been, it was developed originally by Facebook, and was released as open source around 2015, I believe, and has really taken off in a big way. So, it is quite a powerful construct. Ultimately, what it is doing is, it is trying to bring the power of a query language, something like SQL to the end user directly, rather than saying that oh, you are restricted by these REST APIs, you now have the power of a query language, as part of the requests that you can make.

Now, one of the things that that query language gives us is also provides something called a type system? For those of you who are programmed in C, you would know that there are types like int for integers, char for eight-bit characters, a string, which can hold multiple

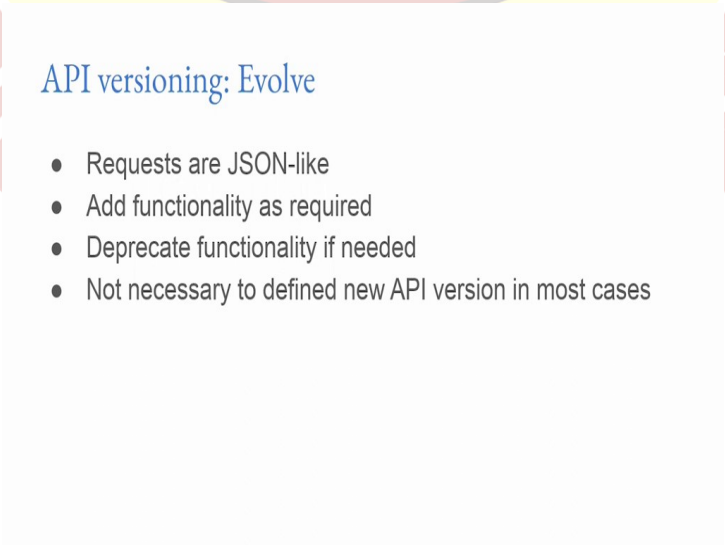
characters together. There is also bool, if you are working with C++? But Python, for example, does not enforce the use of types.

It still has types. So, in Python, for example, there is a distinction between a integer, a floating-point number, a character, a list, a dictionary. Each of those is a different type of object. But it does not enforce the use of types. So, in other words, if I just say A is equal to 5, I could then later on use A is equal to Hello. And it will now be changed from a number to a string.

And when I say A is equal to 5, do I mean an int or a float? Once again, that is sort of determined by the interpreter at runtime. So, Python is an example of what is called a dynamically typed language, it does not specify the types of variables upfront. GraphQL, sort of tells you that using types is a good thing? And it specifies that each query element each entity, what kind of type can it have? Is it a string? Is it an integer? Is it a collection of items, because a large part of what we are doing with regard to transferring data back and forth is actually dealing with collections of items?

It also allows you to specify relationships between items as part of this type specification. So, you can say that a student is associated with an array of courses that a student can have a list of courses that they are enrolled in. So, all of this is part of what GraphQL provides to you it, the query language allows you to specify these types, all that is done at the server end, which means that the server can now validate any queries that are coming in from client.

(Refer Slide Time: 9:17)



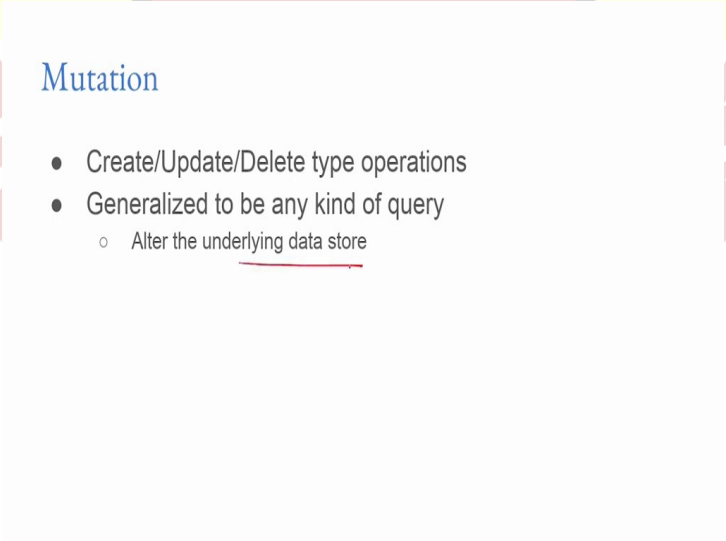
It also means that magically right, the queries that you have can sort of evolve, because these queries are now being passed in purely as JSON objects. I am not giving specific examples over here, because I feel that it is beyond the scope of what can be done in a limited lecture, the, it is definitely worth going into, but it would take a lot more time to explain the details of how GraphQL queries are constructed.

The queries themselves are in a form of representation that is very similar to what JSON would look like. So, it would, in other words, it would primarily look like a JavaScript object. Which also, interestingly enough, looks very much like a Python dictionary. So, Python dictionaries, JavaScript objects, they look very similar when represented. And the good part about it is because in either a dictionary or in an object, you can add or remove keys very easily, you can just specify a key and assign it a value and it is present as part of the object.

Which means that if I want to add functionality to the API over time, it is, I do not need to declare a new API version number, I can just pretty much add it. And whoever was using it earlier, would receive only the older data, whereas someone who wants to use the new version of the API can start making requests for new information.

Similarly, I can also mark certain keys as deprecated, which means that the server that once again requires a little bit more of work, the server will now sort of probably send back a warning along with the response saying do not use this kind of a query in future.

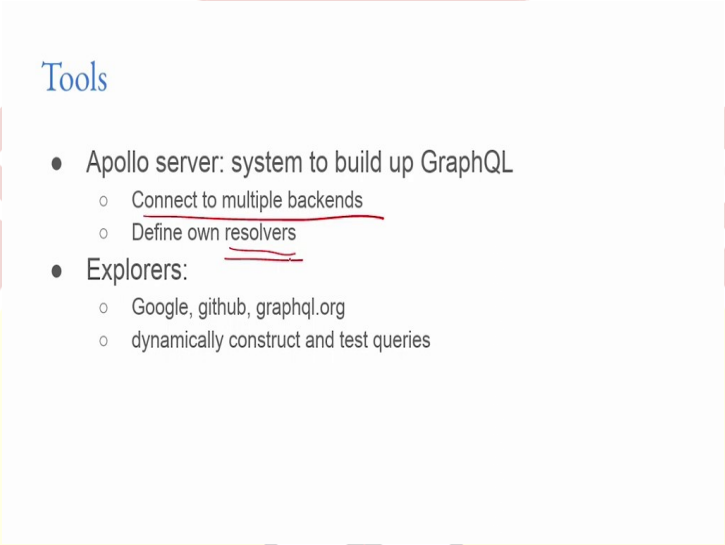
(Refer Slide Time: 11:04)



Now, these queries are not just for retrieving data, you can also use them to mutate. Similar to the concept of mutations in something like you know, view state management. Ultimately,

what a mutation means is something that changes data. In this case, the mutations actually update information at a database backend. So, all the CRUD, or rather, at least the C, U, D, the Create Update, Delete part of CRUD, can now be done through mutations in GraphQL. And the point is, this can now be pretty much generalized to any kind of query, which just alters the underlying data store.

(Refer Slide Time: 11:42)



So, there are a number of different tools that you might want to explore a little bit in order to understand a bit more about how this works. One of the most popular today is what is called Apollo server, which is set of JavaScript libraries that is used to build up GraphQL. And this is where one of the real strengths of GraphQL comes up, even in their fundamental tutorial, the demonstration, one of the things they specify or there is you can connect to multiple backends.

So, a query that you generate, might be pulling in data from multiple different backends, not just from one database. Normally, what we are used to is we have a database, and SQL Lite, or MySQL or PostgreSQL, where we store all the information for our system. And what GraphQL is saying is, look, that is not how real large systems today work, I need to be able to systematically get data from multiple sources.

So for example, can you get me the list of all students who scored more than 85 percent on days when it was raining in Chennai? There is no way I am going to store the information of whether it was raining in Chennai, in my database. But can I get the information about whether it was raining in Chennai? Of course, there is another weather API, which has a history information which can allow me to extract that information.

A GraphQL server could actually connect to both. Why would I want such information? That is a different question. But there are more realistic examples that you can construct it where you might want to connect through different APIs and fuse them together, in order to get more useful information then as possible from any one of them alone.

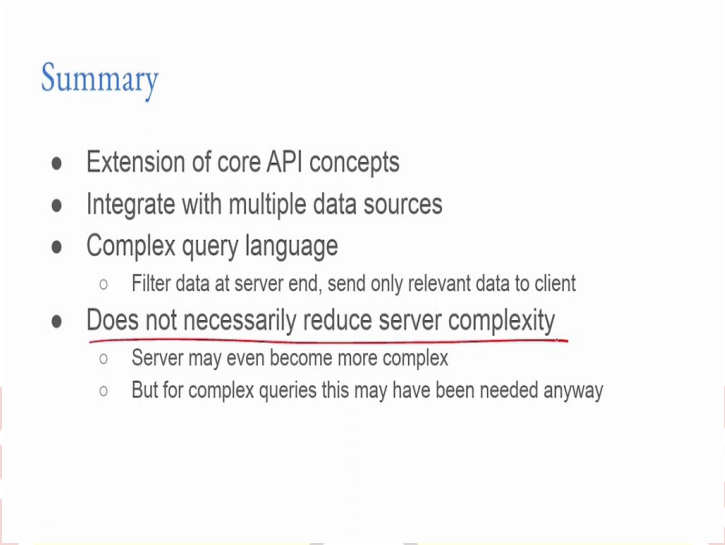
Now, what this means is that everything works around this idea of what is called a resolver. Because after all, it is not that a GraphQL engine, you just put a polo server in your code, and everything is magically taken care of. It still needs to know how to connect to the backend databases, it needs to know how to make the queries. In fact, it needs to know, if you make some arbitrary query, what is permitted? And how should I convert that into something that can actually be answered?

So, the programmer still needs to write code, they still need to do or rather the person who is implementing this GraphQL system, the engine still needs to know that they have to create something called a resolver that takes these requests, and then sort of knows where to send those requests, how to get the data back, how to filter views, etcetera and give back only the information that is required to the client.

So, Apollo server is not a magic bullet user, you do not just put it in there and it takes care of, you know, GraphQL enabled, now you are done. No, you still have to write a fair piece of code in order to actually get it working, and how to make it connect to your different database. Backends, and then fuse the information together in the way that you want.

There are some tools. GitHub, for example, has something called a GraphQL Explorer, which allows you to dynamically construct and test queries. And you can sort of type them in and literally on the fly, see that requests are made and that gives you updated information about what would be the response for a given query. So, in terms of constructing query, debugging them, seeing whether your GraphQL structure is working properly all of these are nice tools to have.

(Refer Slide Time: 15:08)



So, to summarize, the why was GraphQL needed? It is an extension of the core API concepts. But it allows you to do a lot more, it basically allows you to integrate with multiple data sources. It is also a complex query language, which means that you can filter the data and restrict what you send to the client to only the relevant data. That reduces the network traffic, and sort of makes things better.

The important thing to understand this, it does not necessarily reduce server complexity. In fact, it may even become more complex. So, it is not as though by reducing the amount of data sent back to the client, we have reduced the complexity of the system in some way no, that data still had to be fetched, and we needed to know what to send.

So, in some ways, maybe we are increasing load on the server, which means that once again, GraphQL is not just something that you throw in and say, now it is GraphQL enabled, everything is magically taken care of and reduce the amount of data sent to the client. No, what has happened is that your queries are now being filtered and processed on the server. And if you have done a poor job of that, you might be overloading your server.

So, you still need to design it carefully. But if that is done, you can construct a system where you balance it out, rather than sending a whole lot of information over the network and performing the filtering on the client. You take care of it on the server, where presumably first of all, it is a more powerful system.

But secondly, it might also be something that can be distributed across multiple servers. And finally, you might even be able to cache some of these results? Caching is difficult,

fundamentally in GraphQL, because at least at the URL level, you cannot do caching, because it is just a post to a URL that has been used. But if the server by itself knows what the queries are, it might be able to do some other level of caching internally, which is not part of the HTTP caching process, but allows you to get better performance out of the server.