**GraphQL Vs Rest API**

So, I have these two colleagues let's call them "R", and then we'll call the other one "G". Just using initials here to protect our identities.

Now, R is rather talkative. They tell you everything they know whether you want to hear it or not. G, on the other hand, is much more reserved.

They share the information that's absolutely necessary, but no more. But they're always willing to share more if I do ask. And look, this is one way that we can think of the differences between REST, that's the "R", and GraphQL, that's the "G". REST can be a bit of a know-it-all.

You must be specific in your requests and sift through all the data that's returned. With GraphQL, you can be precise and only retrieve the data you need. So GraphQL prioritizes, giving clients exactly the data they request and no more. But I'm getting ahead of myself here.

What clients? What do these technologies do?

Well, both REST and GraphQL are used to build something called APIs, or Application Programing Interfaces. And those allow different applications to communicate with each other over the Internet.

So an API takes requests from a from a client over here, and a client can be like a web application or a mobile app or something like that, and then it retrieves the necessary data from a server over here.

Now REST, or "REpresentational State Transfer" and "Graph Query Language" are two different approaches to building APIs that have their own strengths and weaknesses. So REST is an architectural style that relies on HTTP requests to interact with resources, while GraphQL is a query language that allows clients to retrieve data from multiple data sources in a single API call. So let's define some terminology and we're going to start with GraphQL. In GraphQL, we have something called a "schema". And this is the blueprint that defines all the possible data that clients can query through a service.

Did I say "query"? Yeah, query. That's the next one. So, a query is a request for data that follows the structure defined in the schema. Then we have something called a resolver, and the resolver is called to retrieve the data requested in the query. And this may involve fetching data from multiple data sources and assembling it into a response that matches the query structure. And then lastly, we have mutations. And mutations, their job is to modify data on the server. So, if we think of this in terms of the "Create, Read, Update and Delete" or the CRUD model, a query that would be equivalent to the "read" in the CRUD, and then all the others, the create, the update, and delete, they're handled by the mutations. So that's GraphQL. And if we take a look at REST, we consider "resources" in REST. So resources are really the fundamental concept in REST, and each resource has a unique identifier called a "URI" that identifies it - so "Uniform Resource Identifier". And a client can request a response using an HTTP method. So the methods that we can use are Get, Put, Post and Delete - and that gets us access to those resources. And then the server responds with a representation of the resource in a format like JSON or XML. Now, REST also allows clients to filter, sort and paginate the data using query parameters.

So how are GraphQL and REST similar? Well, both are used to, as I've mentioned, build APIs that allow different applications to communicate with each other over the internet. Both also have a commonality in that they use frameworks and libraries to handle all of that pesky network details so we don't have to worry about them. And they also both operate over HTTP. Although it is worth pointing out that GraphQL is protocol agnostic. Also, both can handle requests and responses using JSON or JavaScript object notation. But there are several key differences between the two technologies that make them better suited for different use cases. When requesting data from a specific end point, a REST API returns the full dataset for that resource. Whereas in contrast, GraphQL is a query language specification. Instead of tools that allow clients to interact with a single endpoint. REST APIs often require multiple requests to fetch related data, while GraphQL can fetch all the data in a single request using a complex query that follows the schema. So clients get what they request with no over-fetching.

Now REST APIs are already very familiar to many developers, so people know how to use them. GraphQL may represent a bit of a learning curve for some. REST APIs are well-suited for applications that require simple CRUD-like operations. So for example, an e-commerce website might use a REST API to allow customers to browse products, add items to their cart and complete orders. In that case, the API would use the HTTP methods we talked about - get, put, post and delete - to manipulate the data, like the products, the orders and the customer information. Now, on the other hand, GraphQL is better suited for applications that require, well, a bit more complex data requests. And what I mean by that is where we have examples of many sort of nested fields or multiple data sources that we need to pull from. So if we think about a company that provides a suite of financial planning tools for its clients, those tools might require data from multiple sources,. So they might need to go to bank transactions, investment portfolios and credit scores. And with GraphQL, the company can build a single API endpoint that allows clients to query all of the data in a single request. So clients simply specify exactly the data that they need, and the server would use a set of resolvers to fetch the necessary data from each source and assemble it into a response that matches the query structure.

Now, REST and GraphQL can also work together because GraphQL doesn't dictate a specific application architecture. It can be introduced on top of an existing REST API and can work with existing API management tools. So like my work buddies R and G, both REST and GraphQL have their unique quirks and strengths. Both have their place in the world of APIs, and by understanding their similarities and differences, you can choose the right tool for the job.