**TITLE :** Advantage of using GraphQL over REST API in client and server architecture

**ABSTRACT:**

REST endpoints give clients methods to interact with data. You can make GET, POST, PUT, or DELETE calls to each endpoint, which each represents a distinct entity. Clients must make several calls in order to populate a user interface (UI) that shows multiple entities because there are only one endpoint for each entity.Additionally, because endpoints deliver a predetermined data structure, clients frequently under- or over-fetch data, resulting in wasteful data transfer.

Finally, while trying to manipulate many pieces of data simultaneously, the entity structure is not adaptable enough. It's not unusual for customers to use a large number of API calls to address a single issue.We will use GraphQL to tackle this issue because it allows users to specify exactly what they need from the server and obtain that data in a predictable manner.

**INTRODUCTION: (*GraphQL****over* ***REST API)***

**PROBLEM DEFINITION:**

**Data Layer:**

All of the data repositories that the service layer has access to are included in the data layer. The architecture of the service layer will determine the specifics of the data layer.

**Service Layer:**

One or more APIs are made available to the UI layer by the service layer. Although it can take on a variety of forms, for the time being we'll focus on three typical architectures:

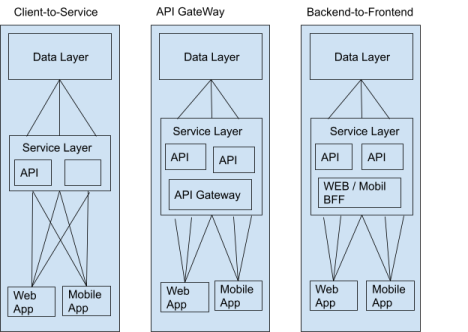
**Client-to-service:**

A direct client-to-service link is established by the service layer's exposure of both internal and external APIs to the UI layer.

**Api Gateway:**

A single API makes up the service layer and is placed in front of other underlying services through an API gateway. Only the API Gateway is used by clients. **Backend-for-frontend:**

The service layer is made up of several more compact APIs that are in charge of one or more customers that are placed in front of other supporting services.



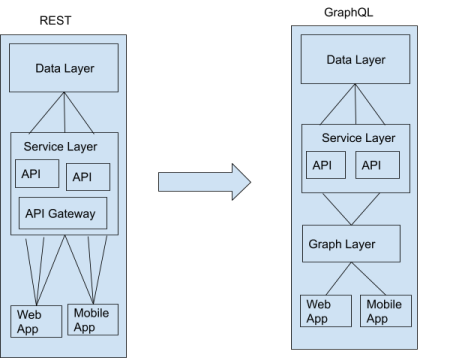
These architectures are functional, but they fall short of exposing more features to the UI layer without burdening the clients with added complexity. It is challenging to manage communication with the service layer due to the steadily growing number of endpoints. The client's capacity to iterate declines as the number of endpoints rises.

To prevent the clients from handling numerous endpoints, the service layer frequently develops into designs like the API gateway and BFF patterns. However, even with APIs specifically designed for a small number of clients, or even just one, they still face the issue of an expanding number of endpoints as the API surface expands.

This issue is resolved by include GraphQL in the stack.Let's explore what this entails and how GraphQL modifies the stack by adding a new layer known as the graph.

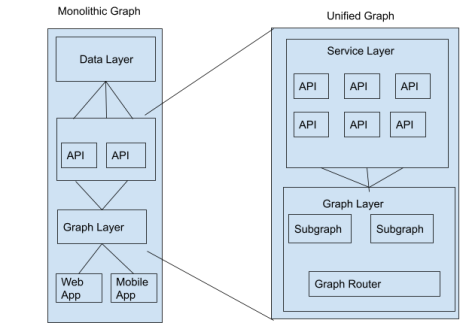
**SOLUTION/APPROACH:**

The entrance to the graph layer is provided by GraphQL. Between the UI layer and the service layer is the graph layer. It unifies all of a company's information and offerings into a single, unified, safe, and discoverable interface that anyone can use as a single endpoint for access.



No matter how big the service layer and graph layer may get, the graph layer will always only expose a single endpoint through GraphQL, which decouples the UI from changes made higher up the stack.

Instead of developing the infrastructure required to manage all the endpoints from the service layer, clients can now concentrate on developing the product. On the other hand, the service layer is free to develop without concern about how it will affect clients.



**Conclusion:**

To answer the initial question, where does GraphQL fit in the stack, let's go back. We have discovered that adding the graph layer to the stack when using GraphQL alters the application design. Therefore, GraphQL creates the basis for a brand-new layer in the stack rather than simply integrating into an existing one.

Clients can obtain what they require through a single endpoint thanks to the graph layer. Additionally, the smaller surface area enables autonomous development on both the client and server sides, resulting in a more adaptable architecture. Check out this GraphQL vs. REST post if you're interested in learning more about the advantages of utilizing GraphQL instead of REST.

**Discussion/Analysis:**

While GraphQL enjoys numerous upper hands over customary REST APIs, there are a few critical drawbacks too.

One weakness is that questions generally return a HTTP status code of 200, whether or not or not that inquiry was effective. On the off chance that your question is fruitless, your reaction JSON will have a high level mistakes key with related blunder messages and stacktrace. This can make it considerably more hard to do blunder taking care of and can prompt extra intricacy for things like checking.

Another weakness is the absence of underlying storing support. Since REST APIs have various endpoints, they can use local HTTP storing to keep away from refetching assets. With GraphQL, you should arrangement your own storing support which means depending on another library, or setting up something like all around the globally interesting IDs for your backend.

This leads us to the last weakness: intricacy. In the event that you have a straightforward REST Programming interface and manage information that is somewhat reliable over the long run, you would be in an ideal situation staying with your REST Programming interface. For organizations that arrangement with quickly evolving information, and have the designing assets to dedicate to rearchitecting their Programming interface stages, GraphQL can address a significant number of the trouble spots experienced with REST APIs.

GraphQL gives a fascinating answer for normal obstacles confronted while utilizing REST APIs. While utilizing GraphQL has a few disadvantages, in the event that you end up working with quickly changing information at scale, it very well may be an extraordinary answer for your business. To more deeply study, look at the docs link shared in the reference. Furthermore, in the event that you're searching for an all the more completely fledged GraphQL arrangement, Apollo gives a simple method for getting everything rolling on both the client and server-side.

**Reference:**

https://graphql.org/learn/

https://graphql.org/learn/caching/