**REST VS GraphQL**

 we will compare RESTful Architecture and GraphQL Architecture. GraphQL is new to the field and RESTful was performing well before the birth of GraphQL.  


**REST:** It is an acronym for Representational State Transfer. REST is an API design style architecture to create web services(APIs).  
**Features of RESTful architecture:**

* REST server implementation is simpler than its counterpart SOAP i.e. one of the reasons why it is popular among new developers.
* REST is stateless, the server doesn’t store the state of a client in form of sessions or anything else. Every request is a new request.
* REST is supported by the JSON and XML, supported by both helps the developer those who are comfortable with one XML or JSON.
* Error message occurrence is meaningful the developer can easily guess the fault.

**GraphQL:** GraphQL was initially developed by Facebook internally for their own use and was later released to people in 2015. It is a manipulation and data query language for APIs. It is a runtime for responding to incoming API requests.

**Features of GraphQL architecture:**

Advertisement

* GraphQL has only 1 endpoint, hence easy to manage.
* Precise fetching of data. As fields are passed along with the GraphQL operation, only the fields that are required are returned.
* Shipping different versions of API is as simple as editing the previous code.
* GraphQL APIs are client-driven. This means that the client decides the structure of API, in contrast to REST where the server decides the API implementation.

**Similarities between REST and GraphQL:**

* They have both the ability to return JSON formatted data.
* REST and GraphQL can fetch data with an HTTP GET method. However, the GraphQL API server usually has only one POST endpoint. Nevertheless, it is possible to use GET with GraphQL to fetch data.
* REST and GraphQL identify every entity as a resource. Every resource has an ID associated with it.

**Comparison between REST and GraphQL:**  
**API structure:**

* **REST:** In RESTful calls, we can use different types of HTTP methods like GET, POST, DELETE. The call also comprises of the endpoint which needs to be called.

GET /api/v1/users/geeksforgeeks

* **GraphQL:** In GraphQL call, only POST method is allowed. The endpoint also remains constant for an application. To service different types of requests, a ‘query’ is built and sent with the request.
* POST /graphql
* {query
* {users(id:"geeksforgeeks")
* {
* firstName
* lastName
* dateCreated

}}}

**Query sample:**

* **REST:** REST defines the ‘what’. The APIs written in REST focus on ‘what’ is to be the Response.

GET /users/:id

* **GraphQL:** In GraphQL call, only POST method is allowed. The endpoint also remains constant for an application. To service different types of requests, a ‘query’ is built and sent with the request.
* POST /graphql
* {query
* (name:"geeksforgeeks")
* {
* id
* dateCreated

}}

**Number of endpoints:**

* **REST:** In REST, an endpoint in a way binds with the particular method/resource. To access different resources, different endpoints are created.
* **Makes a new user:**
* POST /users/GFG
* **Get user with id:**

GET /users/id

* **GraphQL:** In GraphQL, only one endpoint is created which serves all kinds of resources.
* **Fetches datecreated and numberofarticles for a user with id gfg:**
* POST /graphql/{user(id:"gfg") {datecreated numberofarticles}}
* **Gets title of the blog with id 53289:**

POST /graphql/{blog(id:"53289") {title}}

**Documentation:**

* **REST:** In REST, documentation is maintained by developers of endpoints, the resources they are used to access, the parameters(if any) that they take, the type of value that each field takes, and other specifications.
* **GraphQL:** In GraphQL, the documentation is in-the-box in the sense that the information about what queries the GraphQL schema supports is provided or is gettable. This leads to less number of referring to documentation on the part of developer.

**Bandwidth:**

* **REST:** In REST, the selected resource is sent to the client and the filtering of the required information has to be done on the client-side. This consumes high bandwidth as information not required by the client is sent too.
* **GraphQL:** GraphQL provides a way to identify the required fields and only them to be sent to the client. This causes low bandwidth consumption.

**Data type:**

* **REST:** RESTful service can return weakly-typed data as well. The type of data is a crucial bit of information in the documentation. Not providing this information because it isn’t present can cause difficulty to a client when he calls the endpoint.
* **GraphQL:** While GraphQL is predominantly strongly-typed because of the schema that has to be created before the query can be called on it.

**Communication:**

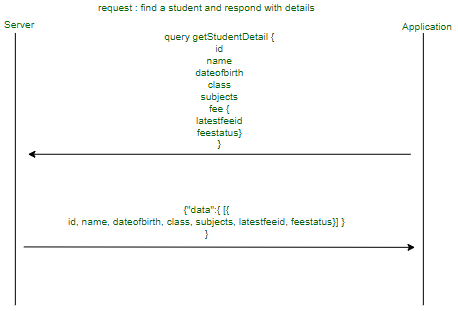
* **REST:** REST only communicates on HTTP/HTTPS protocol.
* **GraphQL:** The GraphQL server can communicate over HTTP, HTTPS, TCP, WebSockets, UDP, FTP.

**Error handling:**

* **REST:** In REST, the status of the response is indicated by the **HTTPStatusCode** in the HTTP Headers. These status codes are standardized and indicate the client about the type of error.
* **GraphQL:** GraphQL always returns 200 as response status over HTTP. The errors that occur while processing queries, if any, are sent as ‘error messages’ with the response.

**Request numbers:**

* **REST:** In REST, there can be multiple requests to resources due to the virtue of how they are structured. For example, to retrieve the details of a student, first **GET** the student with **id**, then get fee details of the students using the field **feeid**  
  Graphical user interface, text, application, email

  Description automatically generated
* **GraphQL:** In GraphQL, multiple resources can be accessed in one call only, thus reducing the number of requests that an application makes, and hence consumes less data.  
  

**Caching:**

* **REST:** In REST, HTTP caching is leveraged and implemented using HTTP. In HTTP headers, such control information is passed along with the response. The header is ‘Cache-Control’ and the server can decide how to use this.
* **GraphQL:** In GraphQL, no such caching mechanism is present. Hence, the responsibility lies on the clients to cache the resources on their end.

**Conclusion:** REST is an old stable approach performing very well till date. GraphQL is boosting up and gained a lot of momentum. If you think GraphQL is an option only for new products and new code then you are wrong. It can perform on old systems that use REST, you can still integrate GraphQL to fetch data from the old REST systems. This will help to improve the client layer. You can write queries to the new GraphQL server to communicate with the REST systems.