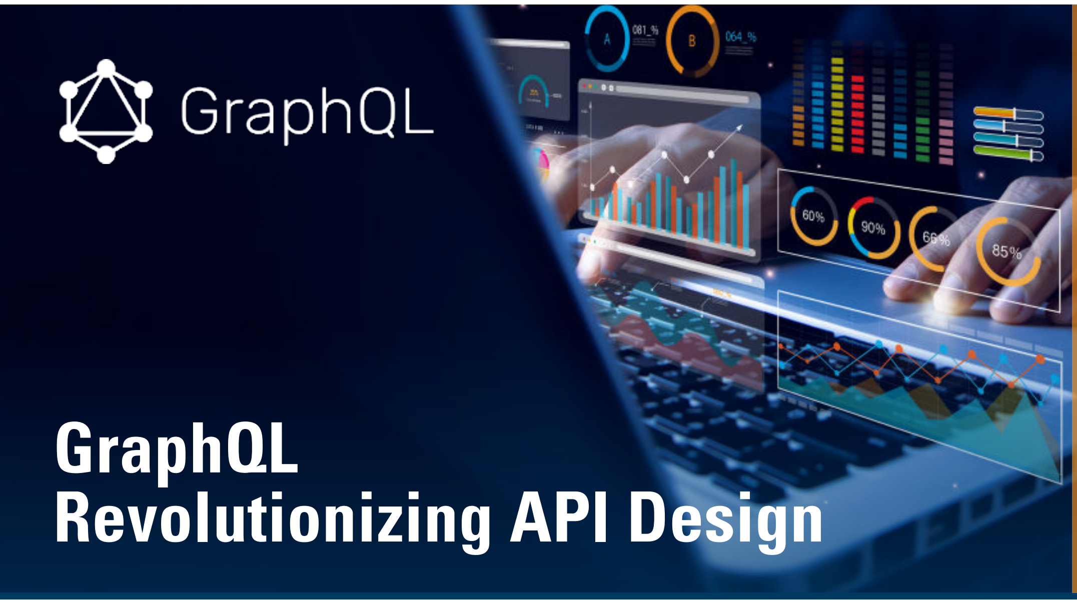

ST Engineering

GraphQL Revolutionizing API Design

In the rapidly evolving landscape of software development, choosing an appropriate technology can significantly impact the efficiency, flexibility, and overall success of an application. GraphQL has emerged as a game-changing solution that addresses numerous challenges associated with traditional API design. This paper explores the compelling advantages of utilizing GraphQL and highlights how it transforms the way we approach data querying, retrieval, and integration.

GraphQL is the leading open-source query language for APIs that streamlines the request process improving performance and response exchange.

In the era of the Internet of Things (IoT) and Big Data where an increasing amount of information is being collected and stored, there is need to revisit how to better optimize the speed and accuracy of retrieving and analyzing data to garner meaningful insights. As a result, GraphQL has risen in popularity particularly for companies managing larger, complex data sources. GraphQL allows a user to get the exact information needed in a single query. As an alternative to REST, GraphQL is also relatively easy to implement with the number of JSON API libraries that are available in many languages.

GraphQL was originally developed by Facebook in 2012 and became open source in 2015. The GraphQL Foundation was created in 2019 with the objective of ensuring that the GraphQL community continue to evolve the specification and reference implementations, or programs that implement all requirements from a corresponding specification. GraphQL has been implemented by large online services such as Shopify, Expedia, Instagram, X (formerly Twitter), and many others. Let's examine some of the key advantages of GraphQL.

Simplifying Customer Integrations

GraphQL puts the client in the driver's seat when it comes to data retrieval. Unlike conventional solutions, GraphQL allows clients to explicitly define the structure of the data they require. This precision ensures that only the necessary information is returned, preventing the unnecessary transfer of large datasets. For example, with REST API, a person's name, date of birth, address, phone number, and other information would be delivered in query from a person object when all you needed was the phone number. Additionally, if you need a person's phone number and the date of their last visit, that will require two object requests to be submitted. By eliminating over-fetching and under-fetching of data, GraphQL optimizes network utilization, leading to faster response times and improved overall performance.

Moreover, GraphQL empowers clients to dictate workflows that best suit their needs. With GraphQL, clients can perform multiple actions in a single call, streamlining the interaction between client and server. This capability not only simplifies the development process but also reduces the number of network requests, minimizing latency and enhancing the user experience. The support for related actions within a single parent

Example of Terminal Provisioning via REST vs. GraphQL

Method	API URL	SEQ
POST	/api/1.0/config/element/terminal/provision	1
GET	/api/1.0/config/element/satelliterouter/<ID>	2
GET	api/1.0/config/element/virtualroot?obj_name=<Network Name>	3
. . .	. . .	. . .
POST	/api/1.0/config/element/<Terminal ID>/apply_changes	24

Method	API URL	SEQ
POST	/graphql	1
WebSocket	/graphql	2

action is another compelling facet of GraphQL. This allows clients to efficiently retrieve interconnected data, reducing the need for multiple round-trip requests. Consequently, GraphQL promotes a more intuitive and coherent approach to data access, fostering the development of applications that seamlessly navigate complex data structures.

Current REST API requires multiple actions (24 in this example) to commission a terminal. Any resulting individual failures on actions require further API calls to fix. GraphQL removes the complexity to the request body, simplify validation by providing client schema, and drastically reduce the calls necessary.

Reducing Logic

GraphQL can substantially reduce the amount of logic required on the client side. Traditional solutions often necessitate extensive client-side processing to aggregate, filter, and manipulate data retrieved from different endpoints. In contrast, GraphQL's ability to consolidate multiple actions and related actions simplifies the data-fetching process. This results in cleaner, more maintainable client-side code, as developers are freed from the burden of managing intricate data retrieval logic.

Future Proof

One of the most distinctive features of GraphQL is its inherent "future-proof" nature. This sets it apart from conventional solutions, which often require meticulous versioning to accommodate changes and potentially disrupt customer integrations. Being technology-agnostic, companies can implement GraphQL with programming languages they already use, such as Java, Python, Ruby, .NET or Javascript. GraphQL takes a granular approach, which enables clients to request only the precise data they need. As we introduce new features or enhancements, clients can seamlessly integrate these changes into their workflows without undergoing extensive modifications. This forward compatibility not only ensures a smooth transition but also cultivates an agile and adaptable ecosystem.

Drawbacks and Concerns:

While GraphQL presents numerous advantages in revolutionizing data interaction and integration, it's important to acknowledge some of its drawbacks. One such drawback of GraphQL is that queries tend to result in larger text payloads being sent compared to traditional REST requests. This can have implications for bandwidth consumption and response times. However, GraphQL allows query responses to be finely filtered, enabling clients to request only the necessary data. Additionally, GraphQL's architecture permits the assembly of query responses from multiple microservices, further optimizing the data returned and reducing the amount of unnecessary information.

The process of defining an initial schema in GraphQL might be seen as a disadvantage, as it requires more upfront work compared to some other approaches. However, GraphQL's schema-first approach allows for effective code generation, automatically generating types, queries, and mutations. This considerably reduces the total cost of ownership over time, as it streamlines development and enforces consistency.

While these concerns may give pause, it's crucial to recognize that the advantages of GraphQL far outweigh these disadvantages many cases. GraphQL's abilities significantly enhance application performance and user experience.

As the software industry continues to evolve, GraphQL stands as a versatile and powerful tool that significantly enriches the way we interact with and integrate data, ultimately paving the way for more efficient, flexible, and responsive applications.
along with reliable transport mediums to deliver comprehensive solutions in the digital marketplace.

GraphQL vs. REST

	GraphQL	REST
What is it?	A query language and method to create and manipulate APIs	A set of rules that defines structured data exchange between a client and a server
Architecture	Client-driven	Server-driven
Ideal for	Large, complex, and interrelated data sources	Well defined, simple data sources
Data access	Single URL endpoint	Multiple endpoints in the form of URLs
Data Fetching	Returns data in a flexible structure defined by the client	Returns data in a fixed structure defined by the server
Use Cases	Multiple microservices, mobile apps	Simple apps, resource-drive apps

At ST Engineering iDirect, we recognized the need for a more flexible and efficient structure data exchange as well as the benefits for more granular data to accommodate the growing number of users and API loads on our customers' networks. Our move towards microservices and cloud-based architectures also facilitates the adoption of GraphQL. However, we will continue to support REST for Standard APIs.