12: GraphQL vs REST

Note: I learnt this because my team at Atlassian was exploring using GraphQL as part of their tech stack. So it might be helpful for you, in case your team/organisation is exploring shifting from REST to GraphQL.

Note 2: I used CHATGPT to understand this one. Really loved the simplicity of the answers and found it just enough I needed to know as a PM.

What is GraphQL?

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. It provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

Where is it required?

GraphQL is typically used when building APIs for modern web and mobile applications. It can be used with any programming language and can be integrated with existing code and data. Some common use cases for GraphQL include:

- Providing a consistent and flexible way to fetch data from multiple sources
- Allowing clients to specify exactly what data they need, in a single request
- Enabling teams to evolve their APIs over time without breaking existing client applications
- Providing a powerful developer experience with tools like introspection and real-time debugging

In general, GraphQL is useful anytime you need to fetch and manipulate data in an application, especially when working with complex or evolving data structures.

How is it different from REST?

REST (Representational State Transfer) is an architectural style for building APIs, while GraphQL is a query language and runtime for APIs. In other words, REST is a set of rules for designing APIs, while GraphQL is a specific way of interacting with an API using those rules.

There are several key differences between the two approaches:

- In a REST API, the server exposes a set of URLs (endpoints) that accept specific HTTP methods (such as GET, POST, or DELETE). The client sends a request to one of these endpoints and the server responds with data in a specified format, such as JSON or XML.
- In a GraphQL API, the client sends a query to a single endpoint using the GraphQL query language. The server responds with data in the same format as the query, allowing the client to easily fetch and manipulate the data it needs.
- REST APIs often have a fixed structure, with a fixed set of endpoints that return fixed data structures. This can make it difficult to evolve the API over time without breaking existing client applications.
- In contrast, GraphQL APIs have a flexible and self-describing structure, allowing the client to ask for exactly the data it needs in a single request. This makes it easier to evolve the API without breaking existing clients.

Overall, while REST is a well-established and widely-used approach to building APIs, GraphQL offers several benefits that make it an attractive alternative, especially for modern web and mobile applications that need to work with complex and evolving data.

What is a simple data model?

A simple data model is one that is relatively straightforward and easy to understand. In the context of an API, a simple data model might consist of a small number of resources (such as users, posts, and comments) with a limited number of relationships between them. For example, a blog API with users, posts, and comments might have a simple data model because each resource is relatively independent and there are only a few relationships between them (e.g. a user can create multiple posts, and a post can have multiple comments).

In contrast, a more complex data model might have a larger number of resources with more complex relationships between them. For example, an e-commerce API with products, orders, payments, and shipments might have a complex data model because each resource is closely related to the others and there are many possible relationships between them (e.g. a product can belong to multiple categories, an order can contain multiple products, and payment can be associated with multiple orders).

The simplicity or complexity of a data model is often determined by the specific requirements and constraints of the project. A simple data model may be sufficient for a small, straightforward project, while a larger or more complex project may require a more complex data model.

In general, the more complex the relationships between your resources, the more likely it is that your data model will be considered complex. For example, if your API has many-to-many relationships (e.g. a user can belong to multiple groups, and a group can have multiple users), hierarchical relationships (e.g. a parent category can have multiple child categories), or other complex relationships, then your data model is likely to be considered complex.

Should we let go of REST?

No, not necessarily. Both REST and GraphQL have their own strengths and weaknesses, and the best choice for a particular project will depend on the specific requirements and constraints of that project.

REST is a well-established and widely-used approach to building APIs, and it has a number of benefits, including its simplicity and the fact that it is well-supported by many tools and libraries. In many cases, a REST API may be the best choice, especially if you need to integrate with existing systems or if you have a simple data model that doesn't require the flexibility of GraphQL.

On the other hand, GraphQL offers a number of benefits over REST, including the ability to fetch and manipulate data in a single request, the ability to evolve your API over time without breaking existing clients, and the powerful developer experience provided by tools like introspection and real-time debugging. If these benefits are important for your project, then using GraphQL may be the right choice.

In general, it's important to carefully consider the requirements and constraints of your project when deciding whether to use REST or GraphQL. Both approaches have their place, and the best choice will depend on your specific needs.

Which products use GraphQL?

- Facebook: The social media giant uses GraphQL in its mobile app, allowing it to efficiently fetch and manipulate data from its complex data model.
- GitHub: The popular code-hosting platform uses GraphQL for its public API, allowing developers to easily access and manipulate data related to repositories, users, and other entities.
- Pinterest: The visual discovery platform uses GraphQL to power its mobile app and web experiences, allowing users to easily search and discover new content.
- Coursera: The online education platform uses GraphQL to power its mobile and web experiences, allowing students to easily access and manipulate data related to courses, instructors, and other entities.