IN FOCUS

GRAPHQL FOR NDC

JUNE 2019

Before reading this InFocus, you should already be familiar with the following InFocus document:"How APIs can help airlines", also available on iata.org/ndc.

Why is this important?

The NDC (and ONE Order) XML standards consist of schemas describing messages. These messages work in pairs (request/response), for specific actions, e.g.:

- Shopping (AirShoppingRQ/AirShoppingRS)
- Booking (OrderCreateRQ/OrderViewRS)

The use of a full-featured language such as XML in a process-driven world makes it possible to enforce complex business requirements.

However, this comes at a price. In some situations, developers may prefer to work with simpler solutions, even if the associated functions are limited. In that regard, GraphQL can be an option.

GraphQL is a specification of a query and manipulation language for APIs, initially developed by Facebook for internal use in 2012, and then made public in 2015. GitHub API v4 is another example of a major GraphQL API.

The aim of GraphQL is to provide a simple, efficient, and straightforward interface, simplifying the development work for both the API provider and the API consumer.

It can positively impact the API performance as well as the developer experience. This is even more relevant for newcomers trying to use the API for the first time, for instance in hackathons (programming and business ideas contests).

Zoom into the topic

The response's structure is in the request

The main concept of GraphQL is that the client, in the request, provides the structure of the expected response. This solves two issues at once:

- All required information is retrieved in a single message (no under-fetching, so no need for several messages in a row)
- No unnecessary information is included in the response (no over-fetching, so no superfluous data transmitted)

Compared to other APIs architectures, the number of messages and/or their size can be reduced.

An example of that is shown in the screenshot below: the response (right) looks exactly like the request (top-left); which simplifies the implementation for the client.

How a GraphQL API is designed

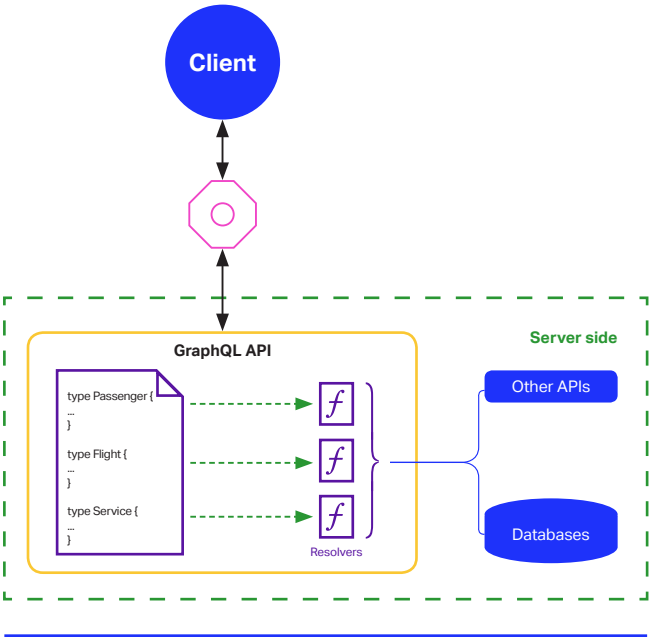
A GraphQL API (on the server side) has two main components:

- A schema describing the possible queries (using the Schema Definition Language)
- Resolvers, i.e. functions that process the client queries, relying on internal databases and/or other APIs.

This structure helps decorrelating the front-end and the back-end development.

For the client, the schema and the native introspection functionality (i.e. the ability to directly explore this schema) facilitates the creation of calls to the GraphQL API. Available requests (queries – fetching data – or mutations – modifying data) are entirely described, as is the structure of the response that can be requested. Because the client describes the expected response in the request, the actual response is fully predictable.

Architecture of a GraphQL API



Industry state of play

The GraphQL Schema Definition Language is not descriptive enough to represent the full complexity of XSDs (XML schemas) currently used for NDC and ONE Order. However, it is possible to design and implement GraphQL APIs providing a subset of the NDC/ONE Order functionalities.

A study of GraphQL (and other paradigms and data formats for APIs) is available on airtechzone.iata.org. A GraphQL API prototype has also been designed and implemented: it is an additional layer on top of the IATA NDC sandboxes, providing basic shopping and booking services. The source code is available on github.com/airtechzone/ndc-graphql-api.

GraphQL cannot be a complete solution for NDC and ONE Order as they are today, because of the complex business requirements that need to be enforced. However, a majority of APIs with a simpler logic can use this technology. GraphQL can facilitate vAPI development and its usage, as well as contribute to their efficiency.

Example from the NDC GraphQL API prototype

```
query retrieveOrder($RQ: OrderRetrieval!) {
  retrieveOrder(request: $RQ) {
    OrderID
    Owner
    TotalPrice
  }
}
```

QUERY VARIABLES

```
{
  "RQ": {
    "OrderID": "B12K25"
  }
}
```

```
{
  "data": {
    "retrieveOrder": {
      "OrderID": "B12K25",
      "Owner": "C9",
      "TotalPrice": 1278
    }
  }
}
```