Interface Technology

How to connect the different parts of the system together. Understand it as "connectives" between servers and clients (mainly in a context of web services).

We will address two main technologies:

- `REST` (Representational State Transfer)
- `GraphQL` (Graph Query Language)

REST

Origin

It is basically the `http` protocol but with a convention on top of it. Before REST, we had `SOAP` (Simple Object Access Protocol) which was based on `XML` and would contain all the information about the request and the response. The 'enveloppe' containing that information would be unpacked and process. The web services would be big, not flexible and complex.

A technology appeared which enabled lightweight requests. Android would historically support SOAP (which consumed a lot of energy). The iPhone was then a breakthrough because it was REST native (which consumed less energy, enabling the iPhone to have a longer battery life and faster response time). REST then became the standard for web services.

Principles

The idea is to translate the server schema to an API. You then access data whereas SOAP was focussed on operations. The convention grew organically and we therefore can hardly talk abotu a standard. `RESTful` is a term used to describe a web service that is compliant with the original REST principles.

The architecture builds on basic `http` and decouples the data-centric and the user-centric applications (driven by web services). It has multiple principles:

1. **Resource based**: The main idea is to think of the server as a collection of resources. The client can then request a resource and the server will return it. It is not based on actions. Related to that idea, nouns are used in URLs instead of verbs. The resources will be identified by a unique URL (URI - Uniform Resource Identifier). Different APIs can also be used to access the same resource which is fine (we call that aliasing).

2. **Representation**: The resource representation are independent of the system that is serving them.

3. **Uniform Interface**: Based on the CRUD, we have `GET`, `POST`, `PUT` and `DELETE` methods. The URL also has a standardised structure.

   > We will have an exercise to construct a restful API based on the information.

4. **Stateless**: The server does not store the state of the client. Each request is self-contained and the server does not need to know the client's history. Therefore, you should not do partial processes with a REST API.

5. **Cacheable**: the REST API can make temporary copies of the data instead of calling the server each time. The server can therefore retrieve the answers faster from the cache than from running the request again (and at no cost). On the server side, you can cache to avoid running the same request. On the client side, you can bookmark to avoid running the same request.

6. **Layered System**: there can be multiple layers between the client and the server. An API to the outside can be used 'like' a database.

Summary

REST APIs are a collection of resources. These can be accessed with URIs (URLs). And the URL will include https methods (GET, POST, PUT, DELETE) and response codes:

- 100: Continue
- 200: OK
- 300: Redirect
- 400: Client Error
- 500: Server Error

The requests can be refined with searching, sorting, and filtering.

GraphQL

This is an emergent technology. Whereas REST had fine-grained APIs and might requires a lot of requests. One solution is to make the response less fine-grained but then it is costly. `GraphQL` is a solution to query the response and to get exactly what you want based on the schema of the response.

The format of the query will also be the format of the answer you want to get back.