REST Architecture

In this lesson, we will learn about the REST architecture and its guiding principles.

We'll cover the following

Guiding principles of REST

A RESTful service needs to comply with the following 6 guiding constraints:

- `Client-Server` – the main principle behind this is the separation of concerns, i.e, separating the user interface concern from the data storage concerns. By following this principle, we can improve the:
  - portability of the user interface across platforms
  - scalability by simplifying the server component
- `Stateless` – each request from the client must be sent to the server with all the necessary information to understand the request and the request cannot take advantage of any stored context on the server. The session state is therefore maintained entirely on the client.
- `Cacheable` – cache constraints require that the data within a response to a request be implicitly or explicitly denoted as cacheable or non-cacheable to prevent clients from giving stale information. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests. This allows us to minimize the network calls made to the server.
- `Uniform Interface` – this is the fundamental principle of the RESTful system. It simplifies and decouples the architecture of the system, allows us to independently scale the components, and improve the interactions between components and other systems. The four architectural constraints that RESTful system should follow in order to a uniform interface are:

    - resource identification
    - manipulation of resources through representations
    - self-descriptive messages
    - hypermedia as the engine of application state (HATEOS)

- `Layered System` – the layered system style allows an architecture to be divided into a number of hierarchical layers or tiers by constraining each of the layer's behavior such that each layer cannot access beyond the immediate layer with which they are interacting with. This allows us to independently scale the layers.

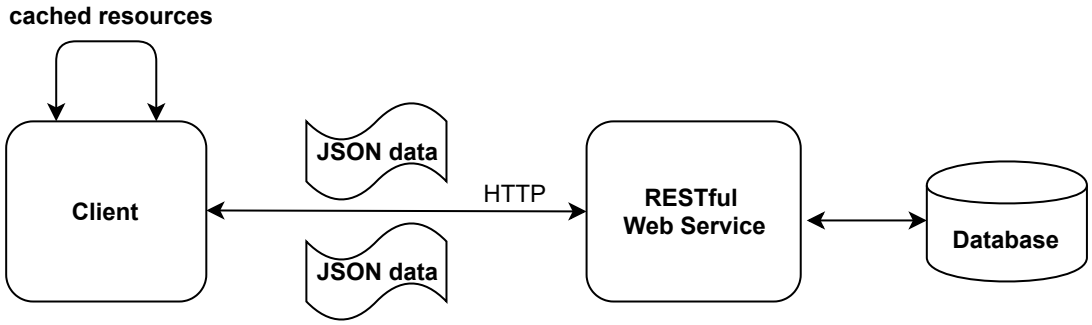
- `Code on-demand` – this principle is an optional one. REST allows client functionality to be extended by downloading executable scripts that can be executed on client-side like Java applets or Javascript.

An extract of the information is taken from https://restfulapi.net.

REST architecture

The diagram below depicts the interaction between client and server that hosts a multi-layered RESTful web service, over HTTP sending JSON messages to each other. When the client requests a cached resource, instead of requesting server for the same, the resource is returned from the client cache itself.

Each of the layers of the RESTful application can independently scaled to provide high efficiency.



Now that we learned about REST and its architecture, let's take a quick quiz to understand how much you have learned.
