**Discussion 2.1 – REST**

Patrick Cuauro

Bellevue University

WEB 420-307O RESTFul APIs

Professor Krasso

May 30, 2023

# REST

REST is an acronym for **RE**presentational **S**tate **T**ransfer and an architectural style for **distributed hypermedia systems**. Roy Fielding first presented it in 2000 in his famous dissertation.

REST does not enforce any rule regarding how it should be implemented at the lower level, it just put high-level design guidelines and leaves us to think of our own implementation.

Like other architectural styles, REST has its guiding principles and constraints. These principles must be satisfied if a service interface needs to be referred to as RESTful.

## Guiding Principles of REST

The six guiding principles or constraints of the RESTful architecture are:

## Uniform Interface

By applying the principle of generality to the components interface, we can simplify the overall system architecture and improve the visibility of interactions.

Multiple architectural constraints help in obtaining a uniform interface and guiding the behavior of components.

The following four constraints can achieve a uniform REST interface:

* Identification of resources – The interface must uniquely identify each resource involved in the interaction between the client and the server.
* Manipulation of resources through representations – The resources should have uniform representations in the server response. API consumers should use these representations to modify the resources state in the server.
* Self-descriptive messages – Each resource representation should carry enough information to describe how to process the message. It should also provide information on the additional actions that the client can perform on the resource.
* Hypermedia as the engine of application state – The client should have only the initial URI of the application. The client application should dynamically drive all other resources and interactions with the use of hyperlinks.

## Client-Server

The client-server design pattern enforces the separation of concerns, which helps the client and the server components evolve independently.

By separating the user interface concerns (client) from the data storage concerns (server), we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.

While the client and the server evolve, we must make sure that the interface/contract between the client and the server does not break.

## Stateless

Statelessness mandates that each request from the client to the server must contain all the information necessary to understand and complete the request. The server cannot take advantage of any previously stored context information on the server.

For this reason, the client application must entirely keep the session state.

## Cacheable

The cacheable constraint requires that a response should implicitly or explicitly label itself as cacheable or non-cacheable.

If the response is cacheable, the client application gets the right to reuse the response data later for equivalent requests and a specified period.

## Layered System

The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior.

For example, in a layered system, each component cannot see beyond the immediate layer they are interacting with.

## Code on Demand (Optional)

REST also allows client functionality to extend by downloading and executing code in the form of applets or scripts.

The downloaded code simplifies clients by reducing the number of features required to be pre-implemented. Servers can provide part of features delivered to the client in the form of code, and the client only needs to execute the code.

Sources: https://restfulapi.net/