(/)

Curriculum
SE Foundations
Average: **108.76%**

REST API

REST API is a software architectural style for Backend.

REST = "REpresentational State Transfer". API = Application Programming Interface

Its purpose is to induce performance, scalability, simplicity, modifiability, visibility, portability, and reliability.

REST API is **Resource-based**, a resource is an object and can be access by a URI. An object is "displayed"/transferred via a **representation** (typically JSON). HTTP methods will be actions on a resource.

Example:

- Resource: `Person` (John)
- Service: contact information (`GET`)
- Representation:
  - `first_name` , `last_name` , `date_of_birth`
  - JSON format

There are 6 constraints:

1. Uniform Interface

- Define the interface between client-server
- Simple and can be split in small parts

HTTP verbs

- `GET` :
  - Read representation of a resource or a list of resources
- `POST` :
  - Create a new resource
- `PUT` :
  - Update an existing resource

- `DELETE` :
(/)
    - Remove an existing resource

URIs - resource name

A resource representation is accessible by a URI:

- `GET /users` : path for listing all user resources
- `GET /users/12` : path for the user `id = 12`
- `GET /users/12/addresses` : path for listing all addresses of the user `id = 12`
- `POST /users` : path for creating a user resource
- `PUT /users/12` : path for updating the user `id = 12`
- `DELETE /users/12/addresses/2` : path for deleting the address `id = 2` of the user `id = 12`

HTTP Response

In the HTTP Response, the client should verify the information of two things:

- status code: result of the action
- body: JSON or XML representation of resources

Some important status code:

- `200` : OK
- `201` : created => after a `POST` request
- `204` : no content => can be return after a `DELETE` request
- `400` : bad request => the server doesn't understand the request
- `401` : unauthorized => client user can't be identified
- `403` : forbidden => client user is identified but not allowed to access a resource
- `404` : not found => resource doesn't exist
- `500` : internal server error

2. Stateless

The server is independent of the client. The server doesn't store user client information/state. Each request contains enough context to process it (HTTP Headers, etc.)

Some authentication systems like OAuth have to store information on the server side but they do it with REST API design.

3. Cacheable

All server responses (resource representation) are cacheable:

- Explicit

- Implicit
- Negotiated

Caches are here to improve performances. In a REST API, clients don't care about the caching strategy, if the resource representation comes from a cache or from a database…

4. Client-Server

REST API is designed to separate Client from the Server. The server doesn't know who is talking to it. Clients are not concerned with data storage => the portability of client code is improved. Servers are not concerned with the user interface or user state so that servers can be simpler and more scalable

5. Layered System

Client can't assume direct connection to server. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.

6. Code on Demand (optional)

Server can temporarily:

- Transfer logic to client
- Allow client to execute logic
- Example: JavaScript