#ASLI ENGINEERING

Everything you need to know about REST

SWIPE

BY

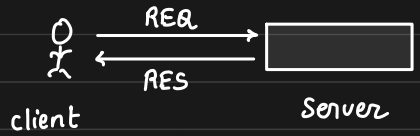
ARPIT BHAYANI

Everything you need to know about REST

REST - Representational State Transfer

⇓

Representation of the entities is central to this idea



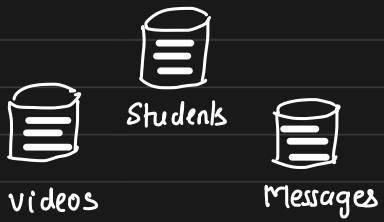
Client demands, server serves

Everything in REST is a Resource

Entity in your application (service) → resource

↓

Student, Customer, Message, Video

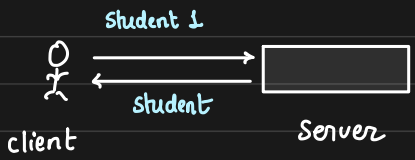


All the data of the application belongs to some entity type. (external)

eg: all students are stored in one table
all messages are stored in some database

Storage representation does not matter!!

The client asks for some data of some entity type in some representation, and server has to respond



What about representation?

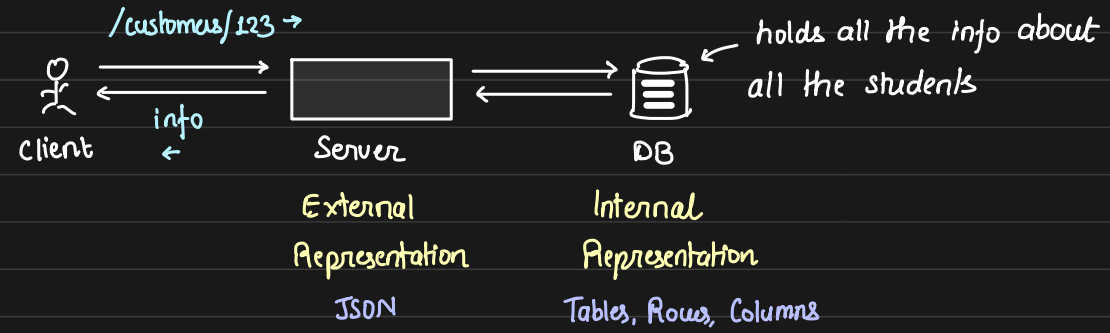
Client demands a particular representation of the entity

↓

JSON, Text, XML, etc

* Practically we all only care about JSON but REST does not restrict us.

REST empowers clients to demand resource in one of the format the server supports



Once the client has one "representation" it can request to update it

The idea is: everything happens on the data/entity sent by the REST server. ↓ Resource

1. creat a resource of type ...
2. updak a resource
3. delete a resource

REST and underlying protocol

REST does not enforce a certain protocal, but it is most commonly implemented over HTTP

REST is just a specification

REST and HTTP

REST goes very well with HTTP

HTTP verbs : GET, PUT, POST, DELETE has well defined meanings

So, by seeing a particular verb we could anticipak its purpose

DELETE /users/1

↓ ↑

delete the resource of type 'user' identified by '1'

With HTTP verbs we can multiplex

eg: get a student's details GET /students/1

instead of having an endpoint like /getStudent

update a student's details POST /students/1

instead of having an endpoint like /updateStudent

HTTP and tooling

Because entire internet works on HTTP, we already have a large efficient set of tooling that would work as is for REST

- HTTP Clients : cURL, postman, requests, etc
- Web caches : nginx cache, varnish, ha proxy
- HTTP monitoring tools : tracing, packet sniffing
- load balancers : distribute load uniformly
- Security control : SSL
- Compression

Downsides of doing REST over HTTP

- consumption is not easy

  not as simple as stubs in RPC

  we would need an HTTP client to make REQ, get response in say JSON, convert it to native objects and then consume

- consumption is repetitive

  Everyone who consumes / adopts REST is writing the same stuff again eg: serialization / deserialization to native objects, failures, timeouts, retries, compression, etc.

  A company may have an internal standardization but most would have to either repeat or creak a shared internal library.

- Some webservers may not support all HTTP verbs

  it is upto the webserver to provide support for HTTP verbs and some may choose to give support only for GET and POST

  if you adopt such servers, you are limiting your REST potential

- HTTP payloads are HUGE eg: JSON

  may not suit well for low-latency requirement

- You cannot switch protocols easily TCP → UDP