REST & RESTful WEB SERVICES

In a Nutshell

- REST is about resources and how to represent resources in different ways.
- REST is about client-server communication.
- REST is about how to manipulate resources.
- REST offers a simple, interoperable and flexible way of writing web services that can be very different from other techniques.
- Comes from Roy Fielding's Thesis study.

REST is NOT!

- A protocol.
- A standard.
- A replacement for SOAP.

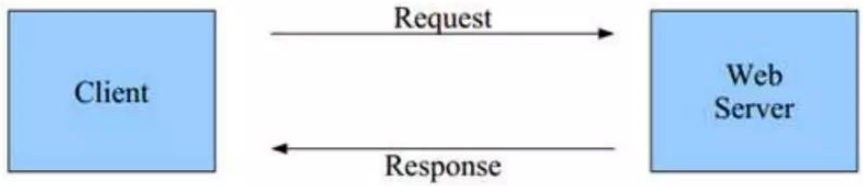
REST

- **Re**presentational **S**tate **T**ransfer
- Architectural style (technically not a standard)
- Idea: a network of web pages where the client progresses through an application by selecting links
- When client traverses link, accesses new resource (i.e., transfers state)
- Uses existing standards, e.g., HTTP
- REST is an architecture all about the Client-Server communication.

An Architectural Style

- REST is the architecture of the Web as it works today and, so it is already used in the **web**!
- It is an software architectural model which is used to describe distributed systems like **WWW** (**W**orld **W**ide **W**eb).
- It has been developed in parallel with **HTTP** protocol.

THE WEB



REST

- Client **request**s a specific **resource** from the server.
- The server **respond**s to that request by delivering the requested resource.
- Server does not have any information about any client.
- So, there is no difference between the two requests of the same client.
- A model which the representations of the resources are transferred between the client and the server.
- The Web as we know is already in this form!

Resources

- Resources are just consistent mappings from an identifier [such as a URL path] to some set of views on server-side state.
- Every resource must be uniquely addressable via a URI.
- “If one view doesn’t suit your needs, then feel free to create a different resource that provides a better view. ”
- “These views need not have anything to do with how the information is stored on the server … They just need to be understandable (and actionable) by the recipient.”
  Roy T. Fielding

Requests & Responses

- REQUEST

  GET /news/ HTTP/1.1

  Host: example.org

  Accept-Encoding: compress, gzip

  User-Agent: Python-httplib2

Here is a **GET** request to «http://example.org/news/»

Method = **GET**

Requests & Responses

- And here is the response…
- RESPONSE

HTTP/1.1 200 Ok

Date: Thu, 07 Aug 2008 15:06:24 GMT

Server: Apache

ETag: "85a1b765e8c01dbf872651d7a5"

Content-Type: text/html

Cache-Control: max-age=3600

<!DOCTYPE HTML>

...

Requests & Responses

- The request is to a resource identified by a **URI** (**URI** = **U**nified **R**esource **I**dentifier).
- In this case, the resource is «http://example.org/news/»
- Resources, or addressability is very important.
- Every resource is URL-addressable.
- To change system state, simply change a resource.

URI Examples

- http://localhost:9999/restapi/books
  - GET – get all books
  - POST – add a new book
- http://localhost:9999/restapi/books/{id}
  - GET – get the book whose id is provided
  - PUT – update the book whose id is provided
  - DELETE – delete the book whose id is provided

REST Characteristics

- Resources: Application state and functionality are abstracted into resources.
  - *URI*: Every resource is **uniquely addressable** using URIs.
  - *Uniform Interface*: All resources share a uniform

  interface for the transfer of state between client and resource, consisting of
    - Methods: Use only HTTP methods such as **GET**, **PUT**, **POST**, **DELETE**, **HEAD**
    - Representation
- Protocol (The constraints and the principles)
  - *Client-Server*
  - *Stateless*
  - *Cacheable*
  - *Layered*

HTTP Methods

- GET – *safe, idempotent, cacheable*
- PUT - *idempotent*
- POST
- DELETE - *idempotent*
- HEAD
- OPTIONS

CRUD Operations Mapped to HTTP Methods in RESTful Web Services

OPERATION	HTTP METHOD
Create	POST
Read	GET
Update	PUT or POST
Delete	DELETE

Status Codes

HTTP status codes returned in the response header:

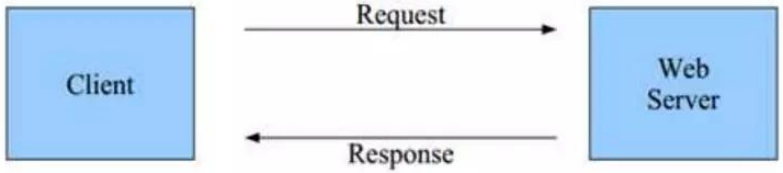
- **200 OK** The resource was read, updated, or deleted.
- **201 Created** The resource was created.
- **400 Bad Request** The data sent in the request was bad.
- **403 Not Authorized** The Principal named in the request was not authorized to perform this action.
- **404 Not Found** The resource does not exist.
- **409 Conflict** A duplicate resource could not be created.
- **500 Internal Server Error** A service error occurred.

RESTful Web Services

- Platform independent.
- Language independent.
- Work on HTTP protocol.
- Flexible and easily extendible.
- They also have some constraints or principles.
  - *Client-Server*
  - *Stateless*
  - *Cacheable*
  - *Uniform Interface*
  - *Layered System*
  - *Code on Demand*

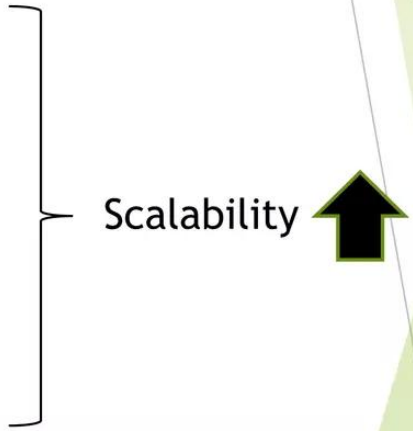
Client-Server

- **Seperation of concerns.**
- Client and server are independent from eachother.
- Client doesn’t know anything about the resource which is kept in the server.
- Server responds as long as the right requests come in.
- **Goal**: Platform independency and to improve scalability.



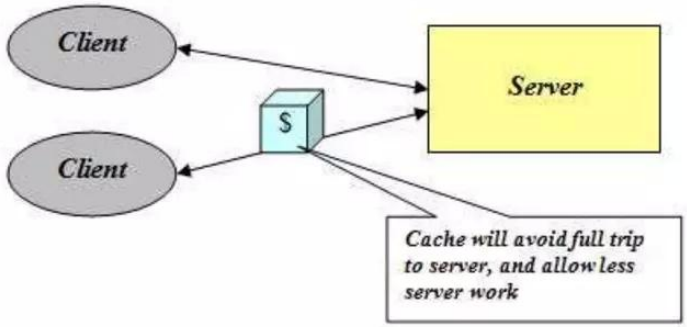
Stateless

- Each request is independent from other requests.
- No client session data or any context stored on the server.
- Every request from client stores the required information, so that the server can respond.
- If there are needs for session-specific data, it should be held and maintained by the client and transferred to the server with each request as needed.
- A service layer which doesn't have to maintain client sessions is much easier to scale.
- Of course there may be cumbersome situations:
  - The client must load the required information to every request. And this increases the network traffic.
  - Server might be loaded with heavy work of «validation» of requests.



Cacheable

- HTTP responses must be cacheable by the clients.
- Important for performance.
- If a new request for the resources comes within a while, then the cached response will be returned.



Uniform Interface

- All resources are accessed with a generic interface (HTTP-based).
- This makes it easier to manage the communication.
- By the help of a uniform interface, client and server evolve independently from eachother.
- E.g. LEGO; eventhough there are different shaped pieces, there are only a few ways to pair up these pieces.

Layered System

- There can be many intermediaries between you and the server you are connecting to.
- Actually, a client does not know if it is connected to the last server or an intermediary server.
- Intermediaries may improve performance by caching and message passing.
- Intermediary servers can increase scalability by load-balancing and can force clients to form some sort of security policies.
- This structure can be used when encapsulation is needed.

Code on Demand

- Servers can send some kind of **executable scripts** to the client-side in order to increase or change the functionality on the client side.
- This may cause **low visibility**, so it is the only **optional** constraint.
- *EXAMPLE*

```
...
<head>
    <script src="utility.js"
        type="text/javascript">
    </script>
...
```

more

- If a service does not include all constraints out of «Code on Demand», it is not a RESTful web service.
- The most epic constraint is «**Stateless**».