



# REST Architecture

We Showed Up -  
Matthew Iglesias, Ricardo Alvarez, Diego Rincon, Daniela Garcia, Jessica Redekop



# Table of Contents

Overview

Example Application

Problems to solve

Security Implications

Project objective

Citations

Constraints

Quiz



# Overview - What is REST?

REST is an architectural style that stands for Representational State Transfer. It is created for distributed hypermedia (non linear medium on information) systems [1].

## **Why is it important?**

As google developer Joe Gregorio states in his video “Intro to REST” REST is the “architecture of the web as it works today.”





# What problem does it solve?

REST solves the problem of the client and the server being dependent of each other. In REST the client and the server are implemented independently which means that the code on the client side can be changed without affecting the server.





# How does it solve the problem?

Because of the fact that both the client and the server are following the same format to send messages to each other both of them can be modular and independent, therefore reducing coupling.





# The 6 RESTful Architectural Constraints

1

Client-Server

4

Layered System

2

Stateless

5

Uniform Interface

3

Cacheable

6

Code on Demand (Optional)



# Constraints: Expanded



# Client-Server

## O1

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

Allows both components to evolve independently -- hence, *separation of concerns*.

The client should only know the resource URI's (Uniform Resource Identifier, to which the request applies, is almost like a HTTP URL (Uniform Resource Locator).

Client



Server





# Stateless

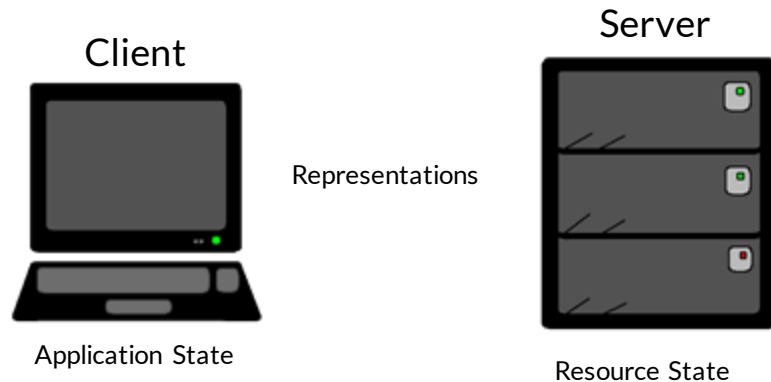
## 02

The server contains no client state:  
Instead, it only has an application state.  
While the client is a resource state.

Any session state is held on the client in the application state.

Therefore, each request of a resource is *self-descriptive*;  
it contains enough context to process the message.

Session state is therefore kept entirely on the client.



### Advantages:

1. Visibility
2. Reliability
3. Scalability

### Trade-Offs:

1. May decrease network performance
2. May reduce the server's control over consistent application behavior





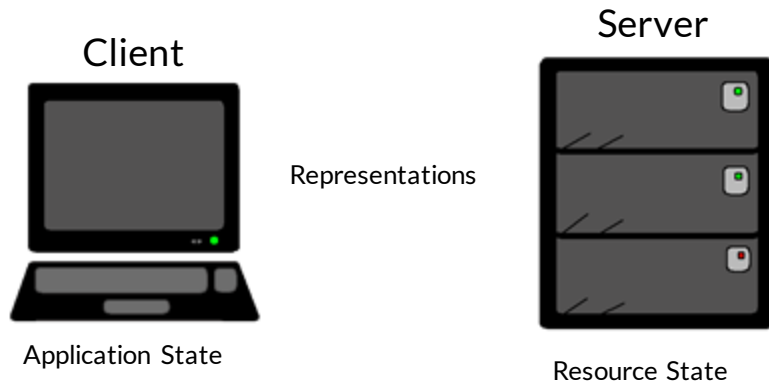
# Cacheable

## 03

The server responses are cacheable.

In REST, caching needs to be applied to all resources if it is possible, and the resources also must declare themselves as cacheable.

Caching can be implemented on the server or client side.



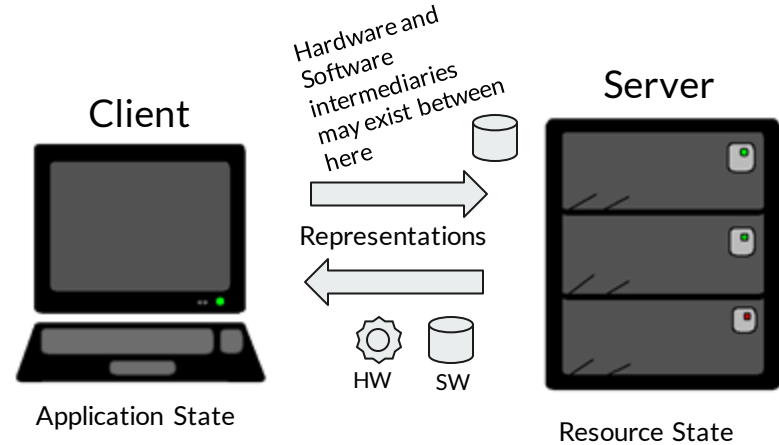
# Layered System

## 04

The client cannot assume direct connection to the server.

Improves scalability:

Hardware and Software intermediaries can exist between the client and server



### Advantages:

1. Encapsulation of legacy services to protect new services from legacy clients
2. Simplification of components by moving infrequently used functionality to a shared intermediary
3. Improved scalability by enabling load balancing of services across multiple networks and processors through the use of intermediaries

### Trade-Offs:

1. The layered systems adds overhead and latency to data processing, reducing performance



# Uniform Interface

## 05

Defines interface between client and server.

Example Uniform Interface:

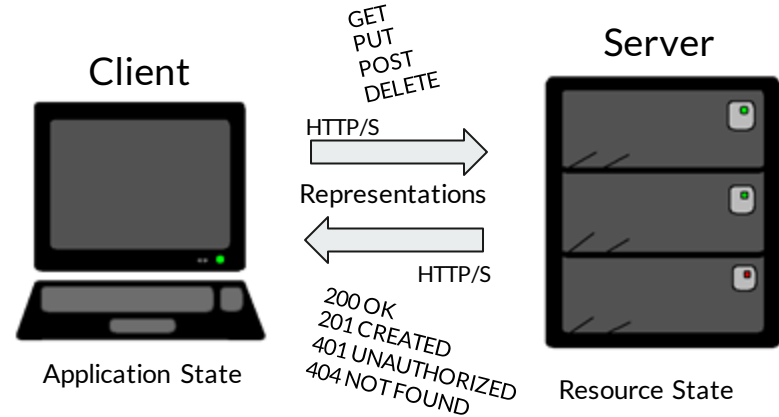
Req and resposes use HTTP or HTTPS as transport protocol.

HTTP Request (from client to server):

Uses verbs

HTTP Reponse (from server to client):

Uses status codes



### Advantages:

1. Simplifies
2. Decomposes

### Trade-Offs:

1. Degrades Efficiency





# Code on Demand (COD)

## o6 (optional)

Optional for this architecture

This property refers to when the client has access to a set of resources but doesn't know how to process them, the client requests the server some code that will be executed inside of the client itself to process those resources.

Provides:

- Can add functionality to a deployed client,
  - Extensible and configurable
- Better perceived performance by the user since the code runs locally (instead of long remote interactions)
- Scalability of the server is improved since some operations are performed by the clients

TradeOff:

- Might cause deployment issues if the client does not trust the server due to the lack of visibility
- Simplicity of the system itself might be reduced



# Example Applications



Due to its stateless nature, RESTful systems are desirable in various web applications. One of the most common is for applications that require extensive back-and-forth communication between the server and a client, i.e. in the use of mobile apps. The RESTful architecture is helpful to their functionality since the client's communication to the server is stateless it doesn't require to be in a continuous connection with it and the server doesn't need to store any context of the client's session.

In general, RESTful architecture can be observed in various web services such as the ones by Google, Facebook and Twitter due to the simplicity and familiarity of the HTTP, the simplicity of managing heterogeneous systems at the same time and not requiring any significant resources from the client's side.





# Security Implications

Like any development project and REST API's there are many ways to secure the REST API. Here are some ideal solutions:

1. REST should be stateless, that way no authorization is dependent on cookies or sessions, instead the request should contain some sort of authentication signature to be validated at every request.
2. Using HTTPS, which utilized SSL to allow authentication credentials to be given a randomized access token delivered to the username seeking request. You can send multiple requests over a single connection without the need of the TCP and SSL handshake.
3. Password Hashing, to avoid security breaching via password hacking. Some hashing algorithms are PBKDF2, bcrypt and scrypt algorithms.
4. Avoid sensitive data on URLs, any usernames, passwords, session tokens and API keys should be strictly prohibited from being visible on any URL; as they can be captured in web server logs which can expose the API key.





# Quiz

Q1: How does the implementation of cache increase speed in requests?

Q2: How does a stateless design decrease efficiency?

Q3: How does using REST reduce coupling?

Q4: What is the main disadvantage of using a layered system?


Q5: How is the code on demand principle useful to extend the system after deployment?





# Citations

- [1] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Ph.D dissertation, College of Eng. and SC. Univ. of California, Irvine, CA, 2000. [Accessed on: May 8, 2020]. [Online]. Available: [https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)
- [2] C. Hunsaker, “REST VS SOAP: When Is REST Better for Web Service Interfaces?: Stormpath,” *Stormpath User Identity API*, 01-Nov-2016. [Online]. Available: <https://stormpath.com/blog/rest-vs-soap>. [Accessed: 08-Apr-2020].



# Thank you.

