REST

Representational State Transfer (REST) is a style of software architecture for distributed systems

Key goals

- Scalability of component interactions
- Generality of interfaces
- Independent deployment of components

Constraints

The REST architectural style describes constraints applied to the architecture, while leaving the implementation of the individual components free to design.

Constraint: client-server

The separation of concerns is the core theme for the Web's client-server constraints.

This means that, for example, clients are not concerned with data storage, which remains internal to each server, so that 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.

Constrain: stateless

The client–server communication is further constrained by no client context being stored on the server between requests. Each request from any client contains all of the information necessary to service the request, and any session state is held in the client.

Constraint: cacheable

As on the World Wide Web, clients can cache responses. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

Constraint: uniform interface

The uniform interface between clients and servers, simplifies and decouples the architecture, which enables each part to evolve independently.

The main guiding principle is **identification of resources**: each distinct Web-based concept is known as a *resource* and may be addressed by a unique identifier, such as a URI.

For example, particular URI, like
https://api.tealogic.com/vertical/reports/{guid}
uniquely identifies a specific report resource.

RESTful web services

A RESTful web service is a web service implemented using HTTP and the principles of REST. It is a collection of resources.

The following table shows how the HTTP methods are typically used to implement a web service.

Resource	GET	PUT	POST	DELETE
Collection URI, such as http://example.com/resources/	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.	**Delete** the entire collection.
Element URI, such as http://example.com/resources/item17	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it doesn't exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.	**Delete** the addressed member of the collection.

Concept

Consider the following few examples of a RESTful HTTP web service vs a SOAP web service

RESTful (HTTP request)	SOAP (method call via client/proxy)	Typical response/result	
		HTTP	SOAP
`GET /reports`	`Client.GetReports()`	`All reports in JSON`	`List of report objects`
`GET /reports/{guid}`	`Client.GetReportByGuid(guid)`	`Specified report in JSON`	
`POST /dashboard`	`Client.SaveDashboard(dashboard)`	`The saved dashboard in JSON`	`void, maybe true or false`

* also each HTTP response has a status code (*200 OK*, *202 Accepted*, and so on)

References

1. Wikipedia
2. RESTful Web Services by Leonard Richardson and Sam Ruby
3. REST API Design Rulebook by Mark Masse
4. How I Explained REST to My by Ryan Tomayko