****

**ENGLISH COMPONENT – CYCLE 3 SESSION 12**

WORKSHEET

**REST and its principles**

**Worksheet 3.12**

**Videos:**

1. <https://www.youtube.com/watch?v=JD6VNRdGl98>
2. <https://www.youtube.com/watch?v=8JDLM4G_bbw>

**Vocabulary**

<https://www.cram.com/flashcards/rest-and-its-principles-s12-12963442>

applet

timestamp

[media type](https://www.iana.org/assignments/media-types/media-types.xhtml)

addressable

affordance

client

streamline

decoupled

metadata

portability

**Reading comprehension: Questioning**

<https://www.youtube.com/watch?v=blbTZKQoxdQ>

**What is questioning?**

**Why questioning?**

**Describe the questioning strategy.**

**What are the different ways we can use questioning?**

**Read the following text.**

**What is REST?**

REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000 in his famous [dissertation](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm).

Like any other architectural style, REST also does have it’s own [6 guiding constraints](https://restfulapi.net/rest-architectural-constraints/) which must be satisfied if an interface needs to be referred to as **RESTful**. These principles are listed below.

**Guiding Principles of REST**

1. **Client–server** – 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.
2. **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
3. **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
4. **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
5. **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.
6. **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

**Resource**

The key abstraction of information in REST is a **resource**. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on. REST uses a **resource identifier** to identify the particular resource involved in an interaction between components.

The state of the resource at any particular timestamp is known as **resource representation**. A representation consists of data, metadata describing the data and **hypermedia** links which can help the clients in transition to the next desired state.

The data format of a representation is known as a [media type](https://www.iana.org/assignments/media-types/media-types.xhtml). The media type identifies a specification that defines how a representation is to be processed. **A truly RESTful API looks like *hypertext***. Every addressable unit of information carries an address, either explicitly (e.g., link and id attributes) or implicitly (e.g., derived from the media type definition and representation structure).

According to Roy Fielding:

Hypertext (or hypermedia) means the **simultaneous presentation of information and controls** such that the information becomes the affordance through which the user (or automaton) obtains choices and selects actions. Remember that hypertext does not need to be HTML (or XML or JSON) on a browser. Machines can follow links when they understand the data format and relationship types.

Further,**resource representations shall be self-descriptive**: the client does not need to know if a resource is employee or device. It should act on the basis of media-type associated with the resource. So in practice, you will end up creating lots of **custom media-types** – normally one media-type associated with one resource.

**REST and HTTP are not same !!**

A lot of people prefer to compare HTTP with REST. **REST and HTTP are not same.**

**REST != HTTP**

Though, because REST also intends to make the web (internet) more streamline and standard, he advocates using REST principles more strictly. And that’s from where people try to start comparing REST with web (HTTP). Roy Fielding, in his dissertation, nowhere mentioned any implementation directive – including any protocol preference and HTTP. As long as you are honoring the 6 guiding principles of REST, you can call your interface RESTful.

In simplest words, in the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs). The resources are acted upon by using a set of simple, well-defined operations. The clients and servers exchange representations of resources by using a standardized interface and protocol – typically HTTP.

Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control. And most importantly, every interaction with a resource is stateless.