Egill Anton Hlöðversson, egilla14, kt. 190193-3309
Edda Steinunn Rúnarsdóttir, eddasr15, kt. 241095-2909

Class assignment III - REST

1. What are the requirements for a web service to be RESTful?
The requirements for a web service to be RESTful are:

- Its resources identification can be easily accessed through the URI
- Uniform interface, which reduces coupling between the client and server
- Self-descriptive messages
- Stateful interactions through hyperlinks

2. What are resources?
Resources are representations of data as a stream of bits. Usually representing a document or a physical object. An example of this is if Reykjavík University kept an API that could retrieve a student as a resource by some student ID, f.x. by issuing a GET request to *https://api.hr.is/student/{id}*, one would get the information (id, name, etc.) on a Reykjavík University student intended to represent that physical student of Reykjavík University as a response (f.x. in JSON or XML format).

3. What is the difference between a resource and a resource mapping?
A resource is the stream of bits whereas resource mapping is the interface that maps a representation to an entity.

4. What is a representation of data? Provide real-world examples.
A representation of data (a resource) is data that represents some physical object or document. For example you can make a request for a book to an API which intuitivey will response with perhaps a web page containing information on the book requested. This means that you'll recieve information on the book you requested as a representation of it although you don't receive the physical book when you issue the request.

5. What is hypermedia?
A uniform interface constraint that aims at reducing client and server coupling by reducing the number of URLs the client needs to know about, even sometimes to only a single entry point URL to the API in question. Hypermedia achieves this by interconnecting resources and describing their capabilities in a machine-readable manner. It's the most important concept related to the REST concept because it essentially separates RESTful APIs from RPC APIs.

6. Name four examples of hypermedia formats and describe their usage scenario briefly.

1. **HAL** (**H**ypertext **A**pplication **L**anguage) is a format of hypermedia which is lightweight and uses the idea of resources and links to model a JSON response. It's most commonly presented as series of links within an attribute that can lead user on to other queries for a resource within the API that stores it. It's most useful to use HAL when dealing with JSON specific API that aims to abstract nested API queries from client, but provies hyperlinks to "deeper" information. An example of this would be constructing the following attribute _link and send it along with resource requested, such as follows:

```
{
        "id": "1",
        "name": "Edda",
        ...
        "_links" :
        {
                "self": { "href": "https://api.hr.is/students/1" },
                "app:courses": { "href": "https://api.hr.is/students/1/courses" },
                "app:grades": { "href": "https://api.hr.is/students/1/grades" },
                …
        }
}
```

   It's most useful to use HAL when one wants a lightweight hypermedia, when one wants to adopt the standard without making breaking changes to API and especially when one wants to design an API that can easily lead the client in both machine and human understandable manner on into nested API queries and information by specifying which actions client can take on a resource.

2. **JSON-LD** is also a lightweight JSON-based hypermedia format similar to HAL in some ways which also leads user deeper into API via adding attributes to a JSON response and is considered very human-readable while still being machine readable as well. It's especially useful for unstructured databases and search engine optimization. It conventionally uses the '@' identifier before any JSON-LD linked data attributes within the JSON object response. However the most important linked data JSON-LD provides is the 'context' which defines a set of terms that are scoped and valid within the representation. It often holds documentation about the meaning of a property An example of **JSON-LD** response could for example be:

```
{
        "id": "1",
        "name": "Edda",
        …
        "@type": "student",
        "@context": {
                “name”: “https://api.hr.is/name”
                “id”: “https://api.hr.is/id”
        }
}
```

It’s most useful to use JSON-LD when you want to adopt the standard without breaking your API with the changes, as the JSON-LD syntax does not interfere with already deployed systems and provide a smooth migration path from JSON to JSON with added semantics. It’s less useful to use JSON-LD when you want to get specifications on actions you can take on resources, as JSON-LD lacks support for that.

3. **Collection+JSON** is also a lightweight JSON-based hypermedia format but differs as it defines both the format and the semantics in a single media type and because that format is essentially geared to handling collections. The object in this format has to have a property called “collection”, which maps to another object, and that collection property contains a nested property “items” that maps to a list containing objects, and so on and so forth. This facilitates navigation of an API relatively easy for client and reduces the client’s need to fully understand the meaning of the API. An example of Collection+JSON format media is as follows (given that api.hr.is/students is a collection of students):

```
{
        "id": "1",
        "name": "Edda",
        ...
        "collection":
        {
            "version": "1.0",
            "href": "https://api.hr.is/students",
            "items":
            [
                {
                    "href": "https://api.hr.is/students/1",
                    "items":
                    [
                      { "name": "Edda", "id": "1", ... },
                      ...
                    ]
                }
            ]
        }
}
```

Collection+JSON format is most useful to use when API needs to handle collections (however it can handle most API responses by representing a single item as a collection of one element).

4. **SIREN** represents generic entities for a resource along with specifications on actions for modifying those entities and links for client navigation. That is the most important part of SIREN, although it can also provide read-only links with an additional link property for a response object just like with HAL format. Most commonly to achieve this functionality SIREN format adds a property action to specify which actions client can take on a resource. An example format of SIREN would be:

```
{       "id": "1",
        "name": "Edda",
        …
        "actions": [
           {
                "class": "graduate",
                "href": "https://api.hr.is/students/1",
                "method": "DELETE",
           }
        ...
        ],
}
```

SIREN format provides most useful when API wants to reduce the need for client to fully understand how to modify, delete or perform an action on some resource.

7. Explain what HATEOAS does. Provide a real-world example.
HATEOS stands for “**H**ypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate”, and refers to a type of hypermedia which is a general term for things like HTML links and forms the techniques a server uses to explain to a client what it can do next (thouroughly defined in section five). A real life example would be a link for business news on the main page navigation bar on www.cnn.com that will redirect the user to https://money.cnn.com/international. This way the client user didn't have to remember that specific URL.

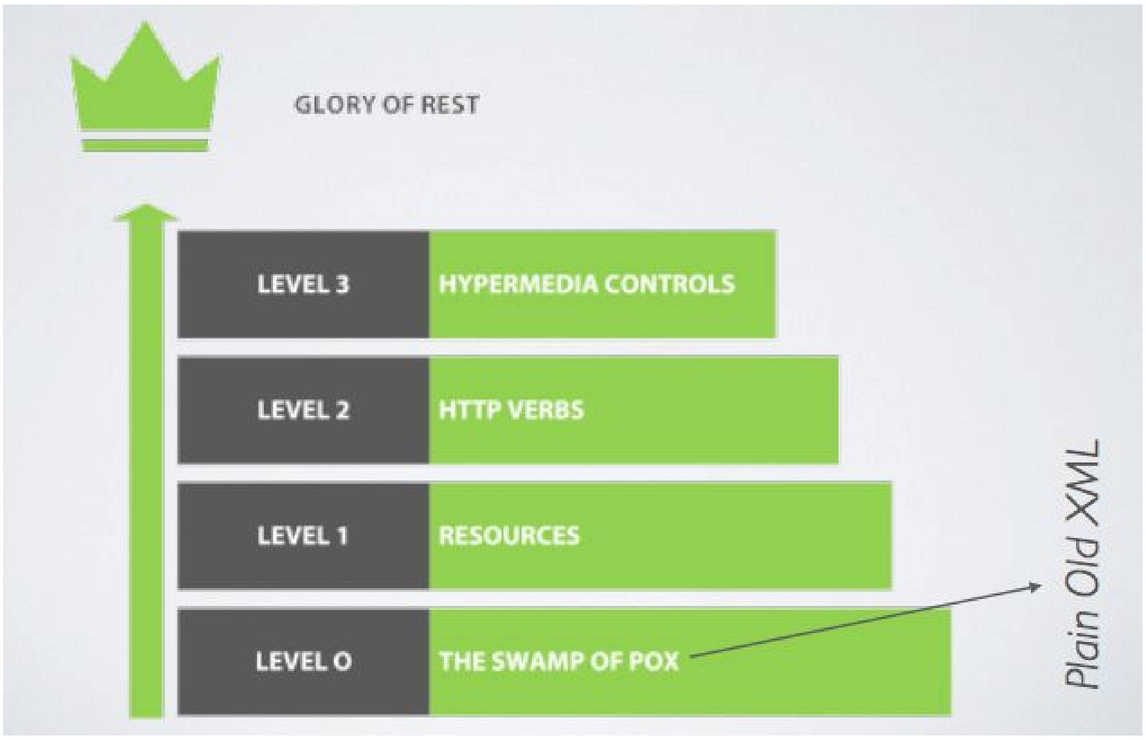
8. What does statelessness mean and how does it compensate the addressability of a web Service?
The term “statelessness” is getting at the fact that the server does not “care” what state the client is in, meaning that there’s no keeping track of state or variables for requests, connections or clients. HTTP is an example of a stateless protocol which means HTTP requests happen in complete isolation, meaning no matter how many identical requests client sends to the same API, it always gets the exact same response back as the API does not store information on client or request and always responds to requests in same ways.

Statelessness can be beneficial; for example, caching is easier as there’s no state that needs to be cached and load-balancing is easier as well as it is unimportant which server to route a given request to as client making the request is independant from (unknown to) all servers and the server all resolve requests in the same manner no matter where they come from

9. Describe the REST Maturity Model.

The model specifies three level for a web service; if all levels are acieved a web service can be acknowledged as a proper REST web service (level zero being f.x. “plain old XML”, that is with no enhancement or fulfilling no requirements towards RESTful APIs). The levels are:

1. The first level of the model is when web service introduces resources, so the first step towards being RESTUL is not making requests to a singular service endpoint but to start “talking” to individual resources by **implementing the URL pattern to access resources**.
2. The second step is to **improve the HTTP verbs used** for responses and requests, this includes actually using HTTP methods such as PUT and POST correctly instead of disregarding HTTP method completely despite using HTTP as some other types of web services such as SOAP web services do.
3. The third level is when a web service has **proper hypermedia convention** for its services and therefore the last step for a web service towards being RESTful. This allows allows the client to remember one URL and then can navigate to all the subparts. Once a web service has reached level three and met requirements of level 1 and 2 it is considered RESTful.



10. Describe the usage of HTTP verb GET in RESTful web services. What is an appropriate status code to return?
The client sends a GET request to ask for a resource (representation of a resource), which is identified by a URL. The most common response code to a GET request is 200 (OK). Redirect codes like 301 (Moved Permanently) are also common.

11. Describe the usage of HTTP verb PUT in RESTful web services. What is an appropriate status code to return?
A PUT request is issued to modify a web service's resource. The client takes the representation of a resource (which client acquires f.x. via GET request), modifies the resource's values or attributes, then sends it as the payload of a PUT request to server containing the resource. The appropriate status code from server on success for a PUT request is most commonly either 200 (OK) or 204 (No Content).
Note: PUT requests are not to be confused with PATCH requests, but PATCH requests differ in the sense that they only modify some values or states for a resource, not the resource as whole whereas *PUT requests modify a resource in it's entirety*

12. Describe the usage of HTTP verb DELETE in RESTful web services. What is an appropriate status code to return?
The client sends a DELETE request when it wants a resource to be removed. The client wants the server to destroy the resource and not to refer to it again (either destroy it completely or mark it as deleted). Of course, the server is not obliged to delete something it doesn't want to. Appropriate status code would be 204 (No Content, i.e., "This resouce is not available or deleted, and I have nothing more to say to that"), 200 (OK, i.e., "Here's explicit notification that you've successfully deleted this with your request"); and 202 (Accepted, i.e., "I'll delete it … later").

13. Describe the usage of HTTP verb POST in RESTful web services. What is an appropriate status code to return?
POST is used when a client wants either create a new resource and append it to the resources POST-to-Append, or it is used to convey any kind of change (Bad habit. It's PUT, DELETE, PATCH, LINK, and UNLINK all rolled into one.)

The appropriate status code would be 201 (Created). It lets the client know that a new resource was created. The Location header lets the client know the URL to this new resource. Another common response code is 202 (Accepted), which means that the server intends to create a new resource based on the given representation, but hasn't actually created it yet.

14. Describe the usage of HTTP verb PATCH in RESTful web services. What is an appropriate status code to return?
PATCH is used when a client wants to modify a part of the state of this resource based on the given representation. If some bit of resource state is not mentioned in the given representation, the server leaves that bit unchanged. The appropriate status code would be 200 (OK) if the server wants to send data (such as an updated representation of the resource) along with its response, and 204 (No Content) if the server just wants to indicate success.
Note: PATCH requests are not to be confused with PUT requests, but PUT requests differ in the sense that they modify a resource in it's entirety whereas *PATCH requests only modify some values or states for a resource, not the resource as whole.*

15. What is the difference between an idempotent operation and NOT?
Idempotent operation is an operation that has the same effect no matter how many times applied. DELETE and PUT are considered idempotent because no matter how often you send DELETE request to delete a resouce, the end result will always be the same, i.e. resource is deleted, and no matter how often you send a PUT request to modify a resource (modify it in the same way) the end result will always be the same, i.e. resource will be modified like you requested.
it If the state of the server is modified after a request has been received and the state is changed, it is considered not idempotent, for example using POST or PATCH is either modifying data or adding data to the existing server.

16. Name three ways to implement to versioning of your web service and describe them:
There are three common strategies used to version your WS

1. Partitioning the URL space
    Example:
        www.mywebpage.com/api/books/v1
        www.mywebpage.com/api/books/v2
        www.mywebpage.com/api/books/v3
    But this is highly inefficient where now you need to support backwards capability.

2. Versioning the media type such as:
    www.mywebpage.com/api/books?version=3
This is seldom a good idea, because they make the client more dependant on domain-specific knowledge.

3. Versioning the profile
The profile neatly isolates the parts of the application that will break clients when they change. This is considered the best approach where the user see no difference in the domain namespace.