Macagno Michele – matr.231705

Assignment3 - Report

RESOURCES	
/nffgs	Collection of Nffgs, can be used to retrieve all nffgs or to create a set of new Nffgs
/nffg	Single Nffg. Used to create a new nffg
/nffgs/{name}	Single Nffg. Used to retrieves a single Nffg given its name
/policies	Collection of Policies. Used to retrieve the set of all policies
/policies/{name}	Single Policy. Used to retrieve the information of a single policy, create a new one or update and existing one
/policy/verification	Used to verify a policy on the fly (without storing it on the system)
	*the way these operations are implemented are described below

Design and Implementation Choices

In this solution I considered the possibility to create a set of Nffgs with a single Post operation (POST /nffgs). The main motivations of this choice are:

- I want to reduce the number of interactions with the server
- I want to give the user the possibility to create a whole set of Nffgs with a predictable result:
  In case at least one nffg is already existing, no nffgs are created. Only if all nffgs can be created then they will be created.
  In this way, the loadAll method of client1 will not have unpredictable and intermediate results.
- I want to guarantee the atomicity of the creation of nffgs in the following case*:
  1) The first user is inserting a set of nffgs (he has already started the transaction) and another user tries to insert Nffgs (also with same name). No interleaved creations of nffgs coming from different users are possible
  These features are applied to the ***loadAll client1 methods.
  Note that the creation of a set of resources together is not the best choice in RESTful WS design, but it is the best solution I found to guaranty a predictable behaviour for loadAll method: If AlreadyLoadedException is thrown, no Nffgs have been created. Moreover this solution requires to maintain locked the resources for all the time necessary to create all Nffgs, including time to contact Neo4J

I didn't apply these considerations while creating policies because policies can be easily overwritten, i.e. at the end of the loadAll methods another client can create a policy overwriting the already existing one or can simply delete it. In this case I preferred to follow the standard guidelines of RESTful WS creating one resource at a time.

I managed **concurrency** at two levels, nffg and policy level. I exploited the synchronized blocks on two different **private static objects**: the map containing the nffgs and the map containing the policies.

- Actions involving exclusively actions on policies (that are not interfering with Nffgs) are synchronized only on the map of policies.
- Action on policies requiring access also to the set of nffgs are synchronized either on the HashMap of Nffgs and on the HashMap of Policies. The order is always:
  synchronized(mapXNffgs)
  synchronized(mapXPolicies)
  In this way the situation of **deadlock can never occur** and, at the same time, it is possible to execute in parallel actions on nffgs and policies that can not interfere. Not all actions on policies

can be synchronized only on map of policies: e.g. Creation of policies must be synchronized also on map of Nffgs, because it is necessary to verify the existence of nffgs and relatives nods.
- All the actions involving Nffgs are synchronized on the HashMap of Nffgs only. This is a trade-off to obtain the highest as possible concurrency level and to maintain a predictable result of operations. The deletion of Nffgs (not implemented) will be synchronized both on maps of nffgs and policies
- In case of future extension of the library (e.g. by adding deletion of Nffgs) the management of concurrency is eased.

Creation of Policies

To create or update a policy I exploited the method **PUT** on the url **/policies/{name}** because:

- Creation of a policy can be executed multiple times on the same URL, i.e. there is not e real distinction between creating or updating of policies.
  To distinguish if the policy was already existing or not a 201 is returned if the policy was not existing, 200 if it was already existing
- If a user can overwrite a policy while creating it, it means it is considered an idempotent action (PUT is used for idempotent actions)
- PUT can be used either to create a resource (if the identifier/url of resource is decided by the client) or to update a resource. The URL/ID of the policy is specified by the user.
- (I followed the idea explained in TelDirectory.java example)

Verification of Policies

To verify policies already stored in the server I used a POST with empty body (POST /policies/{name})
The usage of POST is necessary because verification action is NOT IDEMPOTENT, as it modify every time the verification information of the policies (at least the date-time)

Exceptions

When an error occurs server side it raises an exception. I always set up the Response.status(error_code) in the response, specifying the type as PLAIN TEXT. This way also a browser incurring in an exception is able to show the error message. Moreover, in this way, I can **hide all the details** about the **stack trace** of the exception, avoiding to share information about the server configuration (useful information for attackers). This improves the **security of the server.**

Link

After creating a resource, I return back the URI of the created resource. I also introduced the **href** property including the URI of the resource, this is important especially during the creation of a set of nffgs, where it is impossible to embed the URI of each resource in the request header. Everything to be more compliant with RESTful principles.

Validation

I validated the content of xml body sent by client against the xml schema implementing a `MessageBodyReader<JAXBElement<?>>` in class `NffgPolicyValidationProvider`

CREATE A SINGLE NFFG GIVEN ITS NAME

NOTE: xsd type indicates the type of element you can find on the XML schema.

POST	/nffg
Request body	application/xml xsd type: XNffg
Response body	application/xml xsd type: XNffg
Response	201 – Created - application /xml type: XNffg - Return an element of type XNffg, with updated "lastUpdate" field, containing date-time of server at creation time. - The Nffg contains the attribute href containing the information on the URI of the newly created resource. URI is also embedded in the response header
	400 Bad Request – The request body does not respect the XML schema
	403 Forbidden – Nffg already existing
	500 - Internal Server Error - Includes all problems concerning contacting neo4j - Unexpected Problems
In case of service not available	404 - Problems, service not available – e.g. wrong url, WS not started

CREATE A SET OF NFFGS

POST	/nffgs
Request body	application/xml xsd type: XNffgs
Response body	application/xml xsd type: XNffgs
Response	201 Created - Return an element of type XNffgs, with updated "lastUpdate" field for each Nffg, containing date-time of server at creation time. - Each Nffg contains the attribute href containing the information on the URI of the newly created resource.
	400 Bad Request - The request body does not respect the XML schema
	403 Forbidden – At least a Nffg was already existing
	500 - Internal Server Error - Includes all problems concerning contacting neo4j - Unexpected Problems
In case of service not available	404 - Problems, service not available – e.g. wrong url, WS not started

Note: If status code 403 is returned, no nffgs have been created.

GET ALL AVAILABLE NFFGS

GET	/nffgs
Response Body	application/xml

	xsd: XNffgs
Response	200 OK Contains the set of all available Nffgs in the system. - Each Nffg has the associated attribute href containing the URI of the single resource
	500 – InternalServerError - Unexpected Errors
In case of service not available	404 - Problems, service not available – wrong url, WS not started
	Note: no other errors can occur as no queries are executed, neither to Neo4J service

GET A SINGLE NFFG GIVEN ITS NAME

GET	/nffgs/{name}
Response Body	application/xml xsd: XNffg
Response	200 OK
	404 NotFound – If the Nffg with the requested name is not available in the Server
In case of service not available	404 - Problems, service not available – wrong URL, WS not started
	Note: an error 404 is automatically sent back if the name of the requested Nffg does not respect the regular expression on the name (as explained in the intro.pdf of assignment 1 – name composed by letters and numbers and must start with a letter)

DELETE A SINGLE NFFG GIVEN ITS NAME – removing or not related policies

DELETE	/nffgs/{name} Query parameter to remove or not related policies.
Response Body	application/xml xsd: XNffg
Query Parameter	delpolicy accepted values: y,n Query Parameter delpolicy does not refer to any element in xml schema. I assumed to use a string **y** or **n** as accepted values.
To Remove also related Policies	/nffgs/{name}?delpolicy=y or /nffgs/{name} Considering as default delpolicy=y
Do not remove related policies	/nffgs/{name}?delpolicy=n
Response	200 OK Returns the information about the deleted Nffg.
	400 – BadRequest - Query Parameter Value delpolicy different from a y or n values
	404 – NotFound - If the Nffg you want to delete is not available in the server.

	403 – Forbidden - **Possible only with query parameter delpolicy=n** It occurs when at least one policy referring the the Nffg is still available.
	500 – Internal Server Error - Includes all problems concerning contacting neo4j (if implemented the deletion of nodes from Neo4J) - Unexpected problems
In case of service not available	404 - Problems, service not available – wrong URL, WS not started

CREATE A NEW POLICY – eventually overwriting an existing policy

UPDATE AN EXISTING POLICY

PUT	/policies/{name}
Request Body	application/xml xsd: XPolicy
Response Body	application/xml xsd: XPolicy
Response	201 – Created - Policy created, returns the information of the created policy URI of created policy is embedded in the header and also in href attribute of Policy
	200 – OK The policy with given name/url was already existing and it was replaced
	400 – BadRequest - the request body does not respect the XML schema - **The name specified in the <name> field of the policy is different from the name provided in the URI**
	403 – Forbidden - The policy referes to a Nffg not existing in the server - The policy contains nodes not available in the Nffg
	500 – Internal Server Error – Unexpected problems
In case of service not available	404 - Problems, service not available – wrong URL, WS not started

NOTE: This method **cannot be used to modify the name** of the policy.
To modify the name of the policy it is necessary to delete the existing policy and to create a new one with the preferred name.

GET ALL AVAILABLE POLICIES

GET	**/policies**
Response Body	application/xml xsd: XPolicies
Response	200 OK Retrieves the set of policies stored in the service, including verification results and attribute href specifying the URI of each single policy

In case of service not available	404 – NotFound Problems, service not available – wrong URL, WS not started

GET A SINGLE POLICY GIVEN ITS NAME

GET	**/policies/{name}**
Response Body	application/xml xsd: XPolicy
Response	200 OK Retrieves the information of the policy, including verification result and attribute href specifying the URI of the policy
	404 – NotFound – In case the policy with the requested name is not stored in the server
	500 – Internal Server Error – Unexpected problems
In case of service not available	404 - Problems, service not available – wrong URL, WS not started

DELETE A SINGLE POLICY GIVEN ITS NAME

DELETE	**/policies/{name}**
Response Body	application/xml xsd: XPolicy
Response	200 OK Retrieves the information of the deleted policy.
	404 – NotFound – The policy you want to delete is not stored in the server
	500 – Internal Server Error – Unexpected problems
In case of service not available	404 - Problems, service not available – wrong URL, WS not started

DELETE all stored policies

DELETE	**/policies**
Response Body	Empty
Response	204 - No content
	500 – Internal Server Error – Unexpected problems
In case of service not available	404 - Problems, service not available – wrong URL, WS not started

VERIFICATION OF A POLICY ALREADY STORED IN THE SERVER

POST	**/policies/{name}**
Request Body	Empty
Response Body	application/xml xsd: XPolicy
Response	200 OK Contains the information of the verified policy, including all verification information

	404 – NotFound – The policy you want to verify is not stored in the server.
	500 – InternalServerError - Problems while contacting Neo4j to verify the policy - Unexpected problems - **The nodes the policy refers to are not existing in the corresponding nffg**
In case of service not available	404 - Problems, service not available – wrong URL, WS not started

NOTE: to verify more than one policy, it is required to exploit multiple requests to the system.

VERIFICATION OF POLICY NOT STORED IN THE SERVER

POST	**/policy/verification**
Request Body	application/xml xsd:XPolicy Contains the information of the policy to be verified without storing it in the server
Response Body	application/xml xsd: XPolicy
Response	200 OK Contains the information of the verified policy, including all verification information. **Does not contain the URI** of the policy because it is not stored in the system.
	403 – Forbidden – The Nffg the policy refers to, or the nodes the policy requires to verify are not stored in the server, impossible to satisfy the request
	500 – InternalServerError - Problems while contacting Neo4j - Unexpected Problems
In case of service not available	404 - Problems, service not available – wrong URL, WS not started

NOTE: to verify more than one policy, it is required to exploit multiple requests to the system.