|  |  |
| --- | --- |
| /nffgs | Retrieve the set of all nffgs or create a set of new nffgs |
| /nffg | Used to create a new nffg |
| /nffg/{id} | Retrieve an nffg given its name |
| /policy | Used to create a single new policy or to verify a not stored policy |
| /policies | Retrieve the set of all policies or create a set of new policies |
| /policy/{id} | Retrieve the information of a single policy |
| /policy/verification | Used to verify a policy on the fly (without storing it on the system) |
|  |  |
|  | \*the way this operations are implemented are described below |
|  |  |
|  |  |

Preliminary notes:

In this solution I considered the possibility to create a set of Nffg. 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. In the case at least one nffg is not existing, no nffgs are created. The state of the system does not change.
* I want to guarantee the atomicity of the insertion in the following cases **\***:

1) The user is inserting a set of nffgs (already started the transaction) and

2) another user tries to insert an Nffg having the same name of one of the Nffg the second user is locked until all the nffgs of first user are created.

This way when the transaction of user1 is concluded the status is the system is correctly predictable.

**\***These features are applied to the ***loadAll*** client1 methods.  
  
Similar concepts are applied to the creation of a set of policies: no interleaved actions of different users can modify the result of a transaction. At the end of the transaction the status of the system is perfectly predictable. It can change only after the end of a transaction when another user overwrites existing policies. Hovewer the transaction behaviour is serializable and no undefined states can be reached.  
I do not considered the possibility of creating a whole set of nffgs and policies together as I considered them two different kinds of objects with different lifecycle.  
Concurrency issues:  
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. **This is possible because the Deletion of nffgs is not implemented! In fact the removal of an Nffg is connected to the lifecycle 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. Note that NOT ALL THE OPERATIONS on policies can be synchronized only on the HashMap of policies (see the code for further details).

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 atomicity/predictable result of operations.

CREATE A SINGLE NFFG GIVEN ITS NAME

|  |  |
| --- | --- |
| POST | /nffg |
| Content-Type | application/xml |
| Accept | application/xml |
| Body | 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 XML schema |
|  | 403 Forbidden – Resource already existing - 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 |

GET ALL AVAILABLE NFFGS

|  |  |
| --- | --- |
| GET | /nffgs |
| Accept | Application/xml |
| Response | 200 application/xml  Body type: xsd:Nffgs  Contains the set of all available nffgs in the system.   * Each Nffg has the associated attribute href containing the URI of the single resource |
|  |  |
|  |  |
| In case of service not available | 404 - Problems, service not available – wrong url ,WS not started |
|  | Typically no other errors can occur as no queries are executed, neither to Neo4J service |

GET A SINGLE NFFG GIVEN ITS NAME

|  |  |
| --- | --- |
| GET | /nffg/{name} |
| Content-Type | Application/xml |
|  |  |
| Response | 200 application/xml  Body type: xsd:XNffg |
|  | 404 Not Found – If the nffg with the requested name is not available in the system |
| 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) |
|  |  |

DELETE A SINGLE NFFG GIVEN ITS NAME – remove related policies

|  |  |
| --- | --- |
| DELETE | /nffg/{name}  Query parameter to remove or not related policies. |
| **To Remove also related Policies** | /nffg/{name}?delpolicy=y  Or  /nffg/{name}  Considering as default delpolicy=y |
| **Do not remove related policies** | /nffg/{name}?delpolicy=n |
|  | Query Parameter delpolicy does not refer to any element in xml schema. I assumed to use a string **y** or **n** as allowed values. |
| Response | 200 application/xml  Returns the information about the deleted Nffg. |
|  | 400 – BadRequest - Query Parameter Value delpolicy different from allowed y or n values |
|  | 404 – If the Nffg you want to delete is not available. Impossible to remove it |
|  | 405 – NotAllowed – **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 to delete the Nffg * Unexpected problems |
|  |  |
| In case of service not available | 404 - Problems, service not available – wrong URL, WS not started |

**Note: if delpolicy assumes values different from y/n an error:**

**405 – Method Not Allowed is returned. Please provide the query parameter: delpolicy=y or delpolicy=n. Alternatively do not specify the parameter delpolicy (default actions y is assumed)**

CREATE A NEW POLICY – Overwriting existing policy

|  |  |
| --- | --- |
| POST | /policy |
| Content-Type | application/xml |
| Accept | application/xml |
| Body | xsd:XPolicy |
| Response | 201 – Created – application/xml 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 |
|  | 400 – BadRequest the request body does not respect the XML schema |
|  | 403 – Forbidden – The policy contains node not available in the Nffg |
|  | 404 – NotFound – The Policy refers to a Nffg that is not existing. |
|  | 500 – Internal Server Error – Unexpected problems |
| In case of service not available | 404 - Problems, service not available – wrong URL, WS not started |
|  |  |

**NB IS IT A PROBLEM TO HAVE THE SAME STATUS CODE FOR DIFFERENT PROBLEMS?!**

GET ALL AVAILABLE POLICIES

|  |  |
| --- | --- |
| GET | **/policies** |
| Content-Type | application/xml |
|  |  |
| Response | 200 application/xml  xsd:XPolicies  Contains the set of all policies stored in the service with verification results. Each Policy has the href attribute containing the URI of single resource |
| In case of service not available | 404 - Problems, service not available – wrong URL, WS not started |
|  |  |

CREATE A SET OF POLICIES

|  |  |
| --- | --- |
| POST | **/policies** |
| Content-Type | application/xml |
| Accept | Application/xml |
| Response | 201 - Created application/xml  xsd:XPolicies  Contains the set of all policies just created |
|  | 400 – BadRequest the request body does not respect the XML schema |
|  | 404 – NotFound – At least one policy refers to a Nffg that is not existing. |
|  | 500 – Internal Server Error – Unexpected problems |
| In case of service not available | 404 - Problems, service not available – wrong URL, WS not started |
|  |  |

GET A SINGLE POLICY GIVEN ITS NAME

|  |  |
| --- | --- |
| GET | **/policy/{name}** |
| Content-Type | application/xml |
| Response | 200 application/xml  xsd:XPolicy  Contains all the information of the given Policy |
|  | 200 / Body: xsd type: Policy |
|  | 404 – NotFound – In case the policy with the requested name is not stored in the database |
|  | 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 | **/policy/{name}** |
| Content-Type | application/xml |
| Body |  |
| Response | 200 application/xml  xsd:XPolicy  Contains the information of the deleted policy. |
|  | 404 – NotFound – The policy you want to delete is not stored in the database |
|  | 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 | 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 |

UPDATE A SINGLE POLICY

|  |  |
| --- | --- |
| PUT | **/policy/{name}** |
| Content-Type | application/xml |
| Accept | application/xml  xsd:XPolicy |
| Response | 200 application/xml  xsd:XPolicy  Contains the information of the updated policy |
|  | 403 – Forbidden – The Nffg the policy refers to does not exist. |
|  | 404 – NotFound – The policy you want to update does not exists. |
|  | 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 DOES NOT MODIFY THE NAME of the policy.  
To modify the name of the policy it is necessary to delete the existing one and create a new one with the preferred name.

VERIFICATION OF A POLICY ALREADY STORED IN THE SERVER

|  |  |
| --- | --- |
| POST | **/policy/{name}** |
| Content-Type | application/xml |
| Response | 200 application/xml  xsd:XPolicy  Contains the information of the verified policy, including all verification informations |
|  | 404 – NotFound – The policy you want to verify does not exists. |
|  | 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** |
| Accept | Application/xml  xsd:XPolicy Contains the information of the policy to be verified without storing it in the system |
| Content-Type | application/xml |
| Response | 200 application/xml  xsd:XPolicy  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. |
|  | 404 – NotFound – The nffg or the nodes the policy requires to be verified are not stored in the system, impossible to satisfy the request |
|  | 500 – InternalServerError   * Problems while contacting Neo4j * 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.**