(N)ERD service manual

Authors: Patrice Lopez
Last update: 02.10.2015

1.   Overview

(N)ERD - (Named-)Entity Recognition and Disambiguation - includes a RESTful service implementation for consuming the entity recognition and disambiguation processes.

2.   Build and set up environment for local deployment

To build the (N)ERD service for local deployment, you just have to go to the root of the project and run the following command:

```
> cd nerd;
> mvn clean install
```

Then deploy the generated war to the server. The artifact is under:

```
nerd/target/fr.inria.nerd-<version>.war
```

As an alternative, it is also possible to quick-start and test the service with jetty:

```
> mvn -Dmaven.test.skip=true jetty:run-war
```

3.   Build and set up environment for remote deployment

To build (N)ERD for remote deployment, you have to go to the root of the project and run the following command:

```
mvn clean install -PgenericBuild
```

It will generate 2 artifacts, 1 for the text mining data (*nerd-data-<version>.zip*) and the deployable war (*fr.inria.nerd-<version>.war*):

```
nerd/target/nerd-data-<version>.zip
nerd/target/fr.inria.nerd-<version>.war
```

Copy these 2 artifacts to your remote server.

nerd-data-<version>.zip contains the needed native libraries, the models, lexicons, gazetteers and a config directory that contains 2 properties files *nerd.properties* and *nerd_service.properties*.

You have to unzip nerd-data wherever you want on your server.

```
unzip nerd-data-<version>.zip
```

In *fr.inria.nerd-<version>.war*, the file *web.xml* has 3 parameters to set before starting the server:

- fr.inria.nerd.property: path to nerd.property
- fr.inria.nerd.property.service: path to nerd_service.properties
- fr.inria.nerd.home: path to nerd_home

These properties are filled by the following variables:

_NERD_PROPERTY, _NERD_SERVICE_PROPERTY, and _NERD_HOME, so that it is possible to fill these values with a script given the environment. It is also possible to set manually these variables.

4.   Use of (N)ERD console

Welcome page is available at http://<server instance name>/<root context name> (i.e: for local tomcat http://localhost:8080/<name of the war deploy in webapp>).

Type of request	URL	Requesting type	Parameter name	MIME Type		Description
				Request input type	Response outputtype	
General	/nerd	GET	N/A	N/A	text/html	Gives a very brief description about the (N)ERD service.

This "welcome resource" is accessed by the "About" section (Fig. 4.1). From there, it is possible to access the service interface "Service" (Fig. 4.2) and the administration section (Fig 4.3):
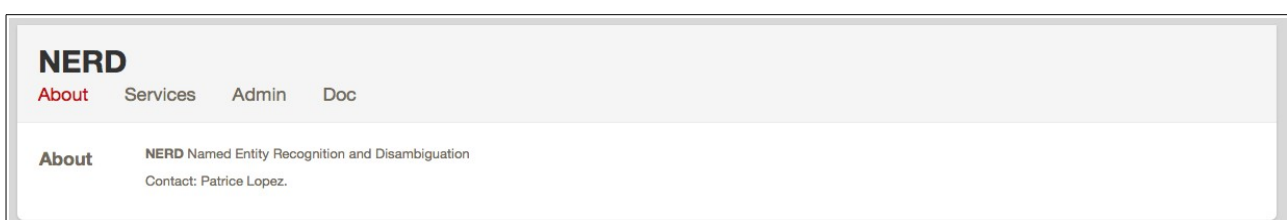

Fig 4.1: About

Fig 4.2: Test Rest Interface
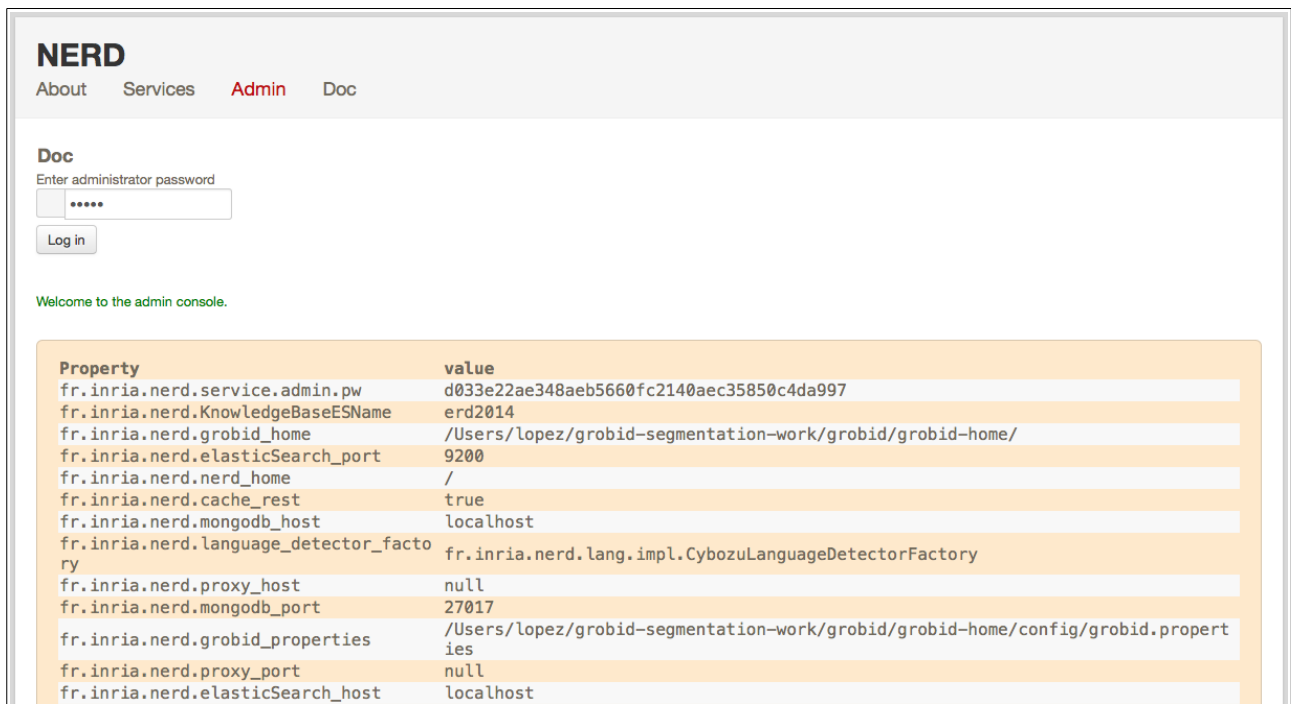
Fig 4.3: Service administration

The web page "Service" (Fig. 4.2) allows to test the different REST requests quickly. In the source code, the class NerdRestService is the entry point for each rest service of the (N)ERD services.

The administration section "Admin" (Fig 4.3) allows to manage dynamically the properties contained in *nerd.properties* and *nerd_service.properties*.

Finally the "Doc" section gives access to the present manual.

5.   (N)ERD REST API

5.1.   Administration services

These services make possible the modification the property values dynamically while the server is deployed. For instance it is possible to modify the nerd_home path or the ElasticSearch instance when new models and resources are available/indexed. This is a standard production requirement.

The table below shows the provided resources corresponding to the HTTP verbs to use the administration services. All url described bellow are relative path, the root url is *http://<server instance name>/<root context>/service/*.

The console web application calls these services and can be used to administrate the (N)ERD deployment or can be exploited as javascript reference client implementation.

Type of request	URL	Parameter name	Requesting type	MIME Type		Description
				Request input type	Response output type	
Administration	/admin	sha1	POST	application/x-www-form-urlencoded	text/html	Request to get parameters of nerd.properties and nerd_service.properties formatted in html table.
	/admin?sha1=<pwd>		GET	String		
	/sha1	sha1	POST	application/x-www-form-urlencoded	text/html	Request to get an input string hashed using sha1.
	/sha1?sha1=<input string>		GET	String		

	/allProperties	sha1	POST	application/x-www-form-urlencoded	text/xml	Request to get all properties key/value/type as xml. Sent xml follow the following schema:
						`<properties>`
						`    <property>`
						`        <key>key</key>`
						`        <value>value</value>`
						`        <type>type</type>`
						`    </property>`
						`    <property>...</property>`
						`</properties>`

	/allProperties? sha1=<password>		GET	String		

	/changePropertyValue	xml	POST	application/x-www-form-urlencoded	text/xml	Change the property value from the property key passed in the xml input. Xml input has to follow the following schema:
						`<changeProperty>`
						`    <password>pwd</password>`
						`    <property>`
						`        <key>key</key>`
						`        <value>value</value>`
						`        <type>type</type>`
						`    </property>`
						`</changeProperty>`

	/changePropertyValue? xml=<some xml>		GET	String		

5.2. Language identification

For preliminary check, a language identification service is available.

Type of request	URL	Requesting type	Parameter name	MIME Type		Description
				Request input type	Response output type	
Language identification	/processLIdText	GET, POST	text	multipart/form-data	application/json	Identify the language of a fragment of text. Answer: JSON with identified language (ISO 639-1) and confidence score: {"lang":"en","conf":1}

```
> curl -X POST http://localhost:8090/service/processLIdText?text=Bonjour
> {"lang":"fr","conf":0.7142846023142064}
```

5.3. Sentence segmentation

This service segments a text into sentences. It is useful in particular for the interactive mode for indicating that only certain sentences need to processed for a given query. Beginning and end of each sentence are indicated with offset positions with respect to the input text.

Type of request	URL	Requesting type	Parameter name	MIME Type		Description
				Request input type	Response output type	

Sentence segmentation	/processSentenceSegmentation	GET, POST	text	String	application/json	Segment a text into sentences.
						
						Answer: JSON with offsets for each sentences:
						`{ "sentences" : [`
						`{ "offsetStart" : 0, "offsetEnd" : 6 },`
						`{ "offsetStart" : 6, "offsetEnd" : 21 }`
						`]`
						`}`

```
> curl -X POST 'http://localhost:8090/service/processSentenceSegmentation?text=I+eat.+Then+I+spleep.'
> { "sentences" : [ { "offsetStart" : 0, "offsetEnd" : 6 }, { "offsetStart" : 6, "offsetEnd" : 21 } ] }
```

5.4.   NERD text processing

Two services performing named entity recognition and disambiguation are available:

– **processNERDText** for processing raw text fragment, resulting in a structured response containing the different annotations,

– **processNERDQuery,** dedicated to interactive applications, which consumes as input a structured JSON query following the format of the (N)ERD annotated text and which produces a re-processed structured response.

These services are restricted to a set of 26 classes of names entities (see https://github.com/kermitt2/grobid-ner/wiki/Grobid-NER-classes-and-senses). If covered by Wikipedia/FreeBase, the service will try to disambiguate the recognized named entities. Entities not covered by these knowledge bases will be characterized by an entity class, a word sense estimation and a confidence score.

In addition, two similar services are provided performing free disambiguation. Without entity class restriction, the service tries to disambiguate all the entities covered by Wikipedia/FreeBase. However, in this case the entities not present in the knowledge base will not be recognized and characterized at all:

– **processERDText** for processing raw text fragment, resulting in a structured response containing the different annotations,

– **processERDQuery,** dedicated to interactive applications, which consumes as input a structured JSON query following the format of the ERD annotated text and which produces a re-processed structured response.

5.4.1.   processNERDText and processERDText

This is the base NERD/ERD service to be used on raw text fragment.

Note that the language identifier will be applied as a starting point of the process to ensure that the language of the text is supported by the system. In the current version, only **English, French and German** are supported by the ERD services, and **only English** is supported by the NERD services. If the language of the text is not supported, an http error 406 (request not acceptable) is sent back. The different possible parameters are described bellow.

The two services processNERDText and processERDText use exactly the same arguments/parameters and differ only with respect to the explanations given in 5.4.

Type of request	URL	Requesting type	Parameter name	MIME Type		Description
				Request input type	Response output type	
Named Entity Recognition and Disambiguation	/processNERDText /processERDText	GET, POST	text onlyNER nbest sentence format customisation	String boolean boolean boolean String String	application/json	Perform a Named Entity Recognition and Disambiguation on a text: identify entities and classify them in term of NER types and, when possible, subtypes. Entity resolution against Wikipedia & FreeBase.

The request is an HTTP GET or POST with the following query parameters:

text Text to be processed in UTF-8
Mandatory - Default: none

onlyNER Boolean indicating if the process should be limited to Named Entity Recognition, without disambiguation and resolution against Wikipedia and Freebase. Performing only NER results in much faster processing time.
Optional – Default: false

nbest Boolean indicating if only the best disambiguation results should be returned for an identified entity or several best hypotheses.
Optional – Default: false

sentence Boolean indicating if a sentence segmentation should be present in the returned result.
Optional – Default: false

format String indicating if the response format is JSON only or JSON+TEI.
Optional – Default: JSON

customisation Indicate the name of a domain customisation, or *generic* if no particular customisation is used.
Optional – Default: generic

```
> curl -X POST 'http://localhost:8090/service/processNERDText?text=John+Smith&onlyNER=true'

> {"text": "John Smith", "runtime": 3, "language": {"lang":"de", "conf": 0.5714286566545531}, "entities":
[{ "rawName" : "John Smith", "type" : "PERSON", "offsetStart" : 0, "offsetEnd" : 10, "conf" : "0.8", "prob" : "1.0" }]}
```

5.4.2. processNERDQuery and processERDQuery

This NERD service supports interactive usage, where a user can manually pre-annotate or correct some annotations and the client application might sends the corrected annotated text to the service several time. There is only a unique parameter which is a JSON string representing a query. This query is similar to the base response of the NERD/ERD service.

When annotations are present in the query, the NERD system will consider them certain and:
- ensure that the user annotations will be present in the output response without inconsistencies with other annotations,
- exploit the user annotations to improve the context for identifying and disambiguating the other possible entities.

The client must respect the JSON format of the NERD/ERD response as new query, as described bellow.

Type of request	URL	Requesting type	Parameter name	MIME Type		Description
				Request input type	Response output type	
Named Entity Recognition and Disambiguation	/processNERDQuery /processERDQuery	GET, POST	query	application/json	application/json	Perform a Named Entity Recognition and Disambiguation on a text: identify entities and classify them in term of NER types and, when possible, subtypes. Entity resolution against Wikipedia & FreeBase.

1) The JSON format for the query parameter to be sent to the service is identical to a response of the service. The parameters of the processNERDText are attribute of the JSON, which typically follow a template like the following one:

```
{    "text": "The text to be processed.",
     "language": {
         "lang": "en"
     },
     "entities": [],
     "resultLanguages" : ["fr","de"]
     "onlyNER": false,
     "nbest": 0,
     "sentence": false,
     "format": "JSON",
     "customisation": "generic"
}
```

An additional available parameter is *resultLanguages*. This parameter is a list of language codes and permits to get the wikipedia pages in additional languages if they exist (currently only English, German and French wikipedia are supported). In this example the processNERDText service optional parameters (*onlyNER, nbest, sentence, format, customisation*) are set to their default values (they remain optionals). The '*entities*' attribute is here empty, which means that there is no pre-defined annotation. Apart from the source language which is pre-set (consequently the language is considered certain, and the language identifier will not be used) and additional target languages of results, this example is similar to a processNERDText request but in a JSON format.

2) In the following example, a pre-defined entity (typically pre-annotated by a user) is present in the '*entities*' attribute:

```
{    "text": "Austria invaded and fought the Serbian army at the Battle of Cer and Battle of Kolubara
             beginning on 12 August.",
    "language": {
        "lang": "en"
    },
    "entities": [ {
            "rawName": "Austria",
            "type": "LOCATION",
            "offsetStart": 0,
            "offsetEnd": 7,
            "wikipedia": "26964606"
    }] }
```

In a typical interactive scenario, an application client first sends a text to be processed via the processNERDText service, and receives a JSON response with some entities. The annotated text is displayed to a user which might correct some invalid annotations. The client updates the modified annotations in the first JSON response and can send it back to the service now as new query via the processNERDQuery. The corrected annotations will then be exploited by the (N)ERD system to possibly improve the other annotations and disambiguations.

3) To support addition of text by a user (e.g. note taking environment where the (N)ERD service is called continuously in background), it is possible to indicate the (N)ERD system to process only certain sentences of the input text. The entity disambiguation and resolution will still consider the entire text and the previous annotations when processing only the indicated sentences.

For this purpose, an additional attribute *processSentence* is possible when calling the service *processNERDQuery.* The parameter/attribute provides the list of sentences to be processed:

```
{
    "text": "The army, led by general Paul von Hindenburg defeated Russia in a series of battles
             collectively known as the First Battle of Tannenberg. But the failed Russian invasion,
             causing the fresh German troops to move to the east, allowed the tactical Allied victory at
             the First Battle of the Marne.",
    "processSentence" : [1]
}
```

When *processSentence* is set, a sentence segmentation always occur, whatever the value of the attribute *sentence*. In this example only the second sentence will be the object of the NERD processing. It is possible to express sentences as interval, e.g. [2, 5-10].

```
    {
        "text": "The army, led by general Paul von Hindenburg defeated Russia in a series of battles
                collectively known as the First Battle of Tannenberg. But the failed Russian invasion,
                causing the fresh German troops to move to the east, allowed the tactical Allied victory
                at the First Battle of the Marne.",
        "processSentence" : [1],
        "sentences": [
            { "offsetStart": 0,
                  "offsetEnd": 163 },
              { "offsetStart": 163,
                  "offsetEnd": 319 } ],
        "entities": [ { "rawName": "Russian",
                        "type": "NATIONAL",
                        "offsetStart": 179,
                        "offsetEnd": 186,
        etc,...
    }
```

> curl -X POST 'http://localhost:8090/service/processNERDQuery' -d '{ "text" : "John Smith", "onlyNER" : true}'

> {"text": "John Smith", "runtime": 3, "language": {"lang":"de", "conf": 0.5714286566545531}, "entities":
[{ "rawName" : "John Smith", "type" : "PERSON", "offsetStart" : 0, "offsetEnd" : 10, "conf" : "0.8", "prob" : "1.0" }]}

5.5. ERD term vector processing

5.5.1. processERDQueryTerms

The (N)ERD service can process a weighted vector of terms. Each term will be disambiguated - when possible - in the context of the complete vector.

The client must respect a JSON format encoding the weighted term vector as query, as described bellow.

Type of request	URL	Requesting type	Parameter name	MIME Type		Description
				Request input type	Response output type	
Disambiguation of a weighted term vector	/processERDQueryTerms	GET, POST	query	application/json	application/json	Perform a disambiguation on a weighted term vector: entity resolution against Wikipedia & FreeBase for each term in the global context of the vector.

The JSON format for the query parameter to be sent to the service must follow the following template:

```
{    "termVector": [{"term" : "computer science", "score" : 0.3},{"term" : "engine", "score" : 0.1}],
    "language": {
        "lang": "en"
    },
    "resultLanguages" : ["de"]
    "nbest": 0,
    "format": "JSON",
    "customisation": "generic"

}
```

The fields *nbest, language, format, customisation* are optional and defined similarly in section 5.4.2 and are set to their default values. The *termVector* field is required for having a well-formed query. *resultLanguages* can be set to get wikipedia pages for languages in addition to the language of the input terms.

```
> curl -X POST 'http://localhost:8090/service/processERDQueryTerms' -d '{"termVector":[{"term":"computer
science", "score":0.3},{"term":"engine","score":0.1}],"nbest":0}'

>{"runtime":33,"nbest":  false,"termVector":[{"term":"computer  science","score":0.3,"entities":[{"rawName":
"computer    science","preferredTerm":"Computer    science","nerd_score":"0.20149253731343286","prob":"1.0",
"wikipediaExternalRef":"5323","freeBaseExternalRef":"/m/01mkq","definitions":[{"definition":"'''Computer
science''' or '''computing science''' (abbreviated '''CS''') is the study of the theoretical foundations of
[[information]] and [[computation]] and of practical techniques for their implementation and application in
[[computer]] systems. Computer scientists invent [[algorithm]]ic processes that create, describe, and
transform information and formulate suitable [[abstraction (computer science)|abstraction]]s to model complex
systems.","source":"wikipedia-en","lang":"en"}]},{"term":"engine","score":0.1,"entities":
[{"rawName":"engine","preferredTerm":"Engine","nerd_score":"0.001837127251406647","prob":"1.0","wikipediaExte
rnalRef":"9640","freeBaseExternalRef":"/m/02mk9","definitions":[{"definition":"An '''engine''' or '''motor'''
is a [[machine]] designed to convert energy into useful [[Motion (physics)|mechanical motion]]. Devices
converting heat energy into motion are referred to as ''engines'', which come in many types. A common type is
a [[heat engine]] such as an [[internal combustion engine]] which typically burns a fuel with air and uses
the hot gases for generating power. [[External combustion engine]]s such as [[steam engine]]s use heat to
generate motion via a separate working fluid.","source":"wikipedia-en","lang":"en"}]}]}], "language":
{"lang":"fr","conf":0.857140003447354}}
```

5.6. ERD search query processing

5.6.1. processERDSearchQuery

The (N)ERD service can disambiguate a search query expressed as a short text. Search query disambiguation uses a special model optimized for a small number of terms and trained with search queries. The difference between standard text and short text is similar to the one of the ERD 2014 challenge (http://web-ngram.research.microsoft.com/erd2014/Docs/Detail%20Rules.pdf).

Type of request	URL	Requesting type	Parameter name	MIME Type		Description
				Request input type	Response output type	
Disambiguation of a search query	/processERDSearchQuery	GET, POST	query	application/json	application/json	Perform a disambiguation on a search query: entity resolution against Wikipedia & FreeBase of the search terms in the global context of the search query.

The JSON format for the query parameter to be sent to the service must follow the following template:

```
{    "text": "concrete pump sensor",
     "language": {
         "lang": "en"
     },
     "nbest": 0,
     "format": "JSON",
     "customisation": "generic"
}
```

The fields *nbest, language, format, customisation, resultLanguages* are optional and defined similarly in section 5.4.2 and are set to their default values.

```
> curl -X POST 'http://localhost:8090/service/processERDSearchQuery' -d '{"text":"concrete pump sensor","language": {"lang":"en", "conf": 1.0},"nbest":0}'
```

> {"runtime": 84, "onlyNER": false, "nbest": false, "text": "concrete pump sensor", "language": {"lang":"en", "conf": 1.0}, "entities": [{ "rawName" : "concrete", "preferredTerm" : "Concrete", "offsetStart" : 0, "offsetEnd" : 8, "nerd_score" : "0.988974165793762", "ner_conf" : "0.8", "prob" : "1.0", "wikipediaExternalRef" : "5371", "freeBaseExternalRef" : "/m/01mxf", "definitions" : [{ "definition" : "'''Concrete''' is a [[composite material|composite]] construction material, composed of cement (commonly [[Portland cement]]) and other cementitious materials such as [[fly ash]] and [[slag cement]], [[construction aggregate|aggregate]] (generally a coarse aggregate made of gravel or crushed rocks such as [[limestone]], or [[granite]], plus a fine aggregate such as [[sand]]), [[water (properties)|water]] and [[Chemistry|chemical]] admixtures.", "source" : "wikipedia-en", "lang" : "en" }] , "domains" : ["Materials", "Engineering", "Aviation", "Architecture"] , "categories" : [{"source" : "wikipedia-en", "category" : "Pavements", "page_id" : 1297865}, {"source" : "wikipedia-en", "category" : "Concrete", "page_id" : 2238414}, {"source" : "wikipedia-en", "category" : "Building materials", "page_id" : 3962842}, {"source" : "wikipedia-en", "category" : "Masonry", "page_id" : 3975663}, {"source" : "wikipedia-en", "category" : "Sculpture materials", "page_id" : 10308784}] }, { "rawName" : "concrete pump", "preferredTerm" : "Concrete pump", "offsetStart" : 0, "offsetEnd" : 13, ...

Fig 5: Search query disambiguation console

5.7. Customisation API

It is possible to use a customisation to specialise the entity recognition, disambiguation and resolution for a particular domain. This API allows to manage customisations for the (N)ERD instance which can then be used a parameter by the (N)ERD services.

Type of request	URL	Requesting type	Parameter name	MIME Type		Description
				Request input type	Response output type	
List the existing customisations	/NERDCustomisations	GET	N/A	N/A	application/json	Return the list of existing customisations as a JSON array of customisation names.
Get the information of a customisation	/NERDCustomisation/ {name}	GET	N/A	N/A	application/json	Return the JSON profile of an existing customisation identified by its name as path parameter.
Create a customisation	/createNERDCustomisation/ {name}	POST,PUT	profile	application/json	application/json	Create a customisation as defined in the input JSON, named following the path parameter. The JSON profile specifies a context via the combination of a list of Wikipedia article IDs, FreeBase entity mid and text fragments. A text describing informally the customisation can be added optionally.
Extend a customisation	/ extendNERDCustomisation/ {name}	POST,PUT	profile	application/json	application/json	Extend the definition of a customisation named following the path parameter by additional context information. The additional context is given by the combination of a list of Wikipedia article IDs, FreeBase entity mid and text fragments, which will be merged with the existing one. A text describing informally the customisation can be added optionally
Delete a customisation	/NERDCustomisation/ {name}	DELETE	name	String	application/json	Delete an existing customisation identified by its name provided as path parameter.

The JSON profile of a customisation to be sent to the server for creation and extension has the following structure:

```
{
        "wikipedia" : [4764461, 51499, 1014346],
        "freebase": ["/m/0cm2xh", "/m/0dl4z", "/m/02kxg_", "/m/06v9th"],
        "texts": ["World War I (WWI or WW1 or World War One), also known as the First World War or the
                    Great War, was a global war centred in Europe that began on 28 July 1914 and lasted
                    until 11 November 1918.", "The war drew in all the world's economic great powers, which
                    were assembled in two opposing alliances: the Allies (based on the Triple Entente of the
                    United Kingdom, France and the Russian Empire) and the Central Powers of Germany and
                    Austria-Hungary."],
        "description" : "Customisation for World War 1 domain"
}
```

Name is the identifier of the customisation. The context will be build based on Wikipedia articles, FreeBase entities and raw texts, which are all optional. Wikipedia articles are expressed as an array of Wikipedia page ID. FreeBase entities are given as an array of mid (the FreeBase Machine IDs). Finally, texts are represented as an array of raw text segments.

Response status codes

HTTP Status Code	Reason
200	Successful operation.
400	Wrong request.
404	Indicates that the customisation resource was not found.
500	Indicate an internal service error.

In case of error status, the service returns a JSON object with an error message.

```
> curl -X POST 'http://localhost:8090/service/createNERDCustomisation/ww1' -d '{"wikipedia" : [4764461]}'
> curl -X GET http://localhost:8090/service/NERDCustomisations
> ["ww1"]
> curl -X GET http://localhost:8090/service/getNERDCustomisation/ww1
> { "name" : "ww1", "wikipedia" : [ 4764461] }
```

6.   REST API response examples

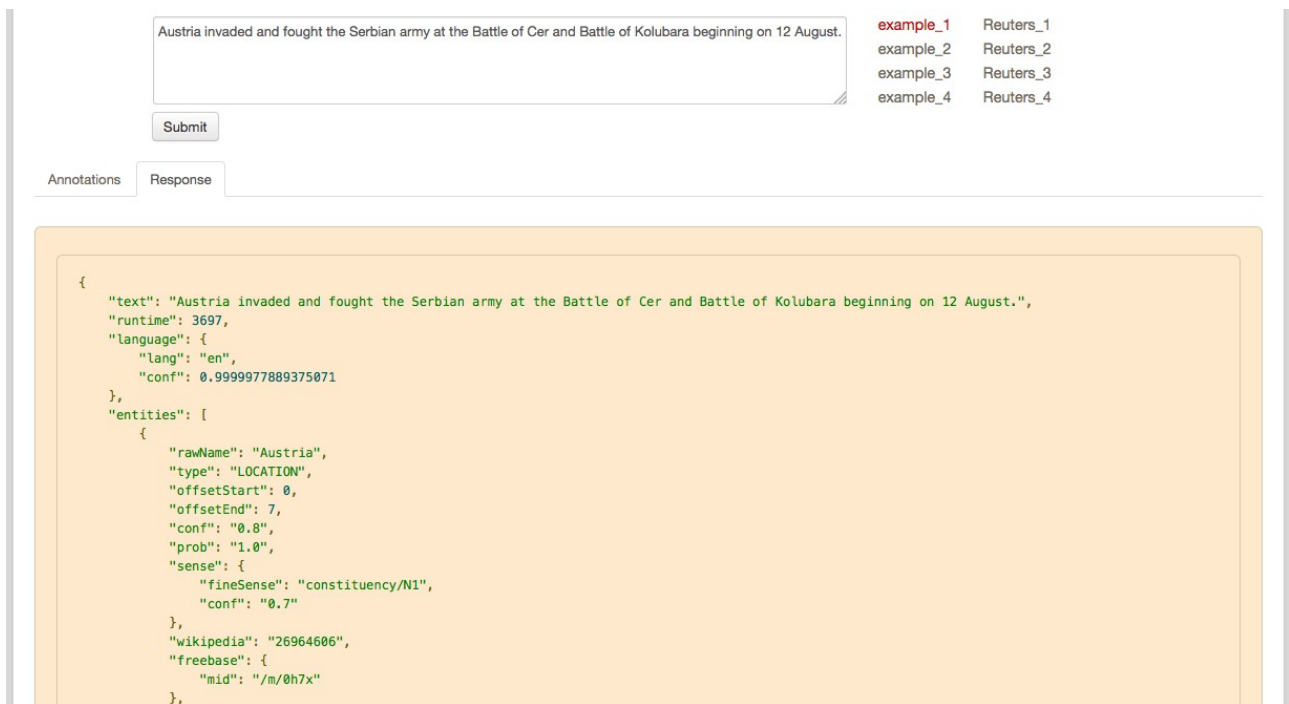It is possible to view the service response with the Web Service console as shown by Figure 6.1.

Fig 6.1: Viewing service response

Named Entity Recognition service:

Input:

Output:

...