Building the Legal Knowledge Graph for Smart Compliance Services in Multilingual Europe

D3.6 Extraction and Linking Services

PROJECT ACRONYM	Lynx
PROJECT TITLE	Building the Legal Knowledge Graph for Smart Compliance Services in Multilingual Europe
GRANT AGREEMENT	H2020-780602
FUNDING SCHEME	ICT-14-2017 - Innovation Action (IA)
STARTING DATE (DURATION)	01/12/2017 (36 months)
PROJECT WEBSITE	http://lynx-project.eu
COORDINATOR	Elena Montiel-Ponsoda (UPM)
RESPONSIBLE AUTHORS	Artem Revenko (SWC)
CONTRIBUTORS	Vova Dzhuranyuk (KD), Dorielle Lonke (KD)
REVIEWERS	Filippo Maganza (ALP), Christian Sageder (OLS), Stefanie Hegele (DFKI)
VERSION | STATUS	v1.0 | Final
NATURE	Other
DISSEMINATION LEVEL	Public
DOCUMENT DOI	10.5281/zenodo.3558282
DATE	29/11/2019

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780602

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	First draft	21/10/2019	Artem Revenko (SWC)
0.2	Reviews received	01/11/2019	Maria Navas (UPM), Filippo Maganza (ALP)
0.3	Updates according to remarks in reviews	04/11/2019	Artem Revenko (SWC)
0.4	Executive Summary, Extraction Services	05/11/2019	Artem Revenko (SWC)
0.5	WSID & EntEx	06/11/2019	Artem Revenko (SWC)
0.6	Dictionary Access	07/11/2019	Vova Dzhuranyuk (KD)
0.7	Reviews	14/11/2019	Filippo Maganza (ALP), Christian Sageder (OLS)
0.9	Final version	18/11/2019	Artem Revenko (SWC)
1.0	Extension of Appendix description	19/11/2019	Artem Revenko (SWC)

ACRONYMS LIST

DA:	Dictionary Access
EL:	Entity Linking
EntEx:	Entity Extraction
KG:	Knowledge Graph
LKG:	Legal Knowledge Graph
NER:	Named Entity Recognition
RelEx:	Relation Extraction
RNN:	Recurrent Neural Network
SEAR:	Search service
URI:	Universal Resource Identifier
WSID:	Word Sense Induction and Disambiguation

EXECUTIVE SUMMARY

The Lynx project aims at facilitating cross-border compliance for European enterprises. This requires the gathering of relevant regulations, case laws, best practices and standards, and their provision through accessible means, which enables end users (in the form of law firms, in-house lawyers, SMEs and the general public) to find answers to their regulatory related needs with ease. In order to provide such accessible means, the Lynx partners have brought together their technical expertise in the fields of Information Extraction, Semantic Web, Knowledge Management, and Document Management, to create an integrated solution.

An essential part of the solution provided by Lynx is the automatic analysis of documents in order to facilitate their discovery and retrieval by the end user. These analyses have three primary objectives:

- To extract information contained within the documents that is relevant to user queries.
- To put documents in context, in terms of relations they hold to other documents.
- To make documents accessible in different languages.

In order to realize these objectives, Work Package 3 serves to develop a series of services that extract various types of information from the documents, and store the information in a well-organized, standards-compliant knowledge graph, which we call the Legal Knowledge Graph (LKG). This LKG will contain not only information about these documents, but also information extracted from them, as well as general knowledge from the compliance domain, in the form of controlled vocabularies.

The services that are reported on in this deliverable are divided into Linking Services, which aim at contextualizing documents or parts of documents by finding references and commonalities with other known documents or entities in the LKG; and Extraction Services which aim at extracting information contained in the documents, both to annotate them and to enrich the LKG with them.

Being a final report, not all of the services have been fully implemented. However, large amounts of work have been done in the standardization of the service architecture, deployment procedure and data interchange format. Furthermore, demonstration implementations are available for most of the services listed here, which are being used to test interoperability among them. Finally, some of these services are the result of ongoing research, and thus their initial version is already an innovation success.

TABLE OF CONTENTS

TABLE OF TABLES

1 INTRODUCTION

1.1 PURPOSE OF THIS DOCUMENT

This document aims at describing the status of services related to linking and extraction functionalities. This description will fulfil two main objectives: the first is a reporting effort, in order to better assess the progress of the project; t he second is the documentation of the status of the services, conventions and functionalities, in order to serve as future reference within the project.

1.2 STRUCTURE OF THIS DOCUMENT

The document starts with an introduction and then 2 major sections follow: linking services (section 2) and extraction services (section 3). After that, we include the references and, finally, an appendix containing reports on OpenAPI descriptions of the services.

2 LINKING SERVICES

2.1 ENTITY LINKING

In the previous deliverable of this task (Deliverable 3.1) this service was presented as two distinct services: EntEx and WSID. In the course of the project it was decided to merge these services for the convenience of the user. However, the functionalities are still served also independently, i.e. EntEx with disambiguation and WSID without extraction.

2.1.1 General Description of Method

Entity Extraction is the process that finds entities from a predefined set of entities in a document. Consider a document containing a string “Abdel and Betty are married since 2005”, and a Knowledge Graph (KG) containing the entity “Betty”. The Entity Extraction process would find the string “Betty” and return its URI in the KG.

The Entity Extraction service (EntEx) was developed using, as a starting point, the PoolParty Semantic Suite software of SWC. In agreement with the general microservice architecture used for Lynx, EntEx is a lightweight wrapper that converts inputs and outputs from the Lynx-defined formats and standards into PoolParty’s. The extraction itself is performed by the latter, by using the LKG developed in the project, in particular, the set of controlled vocabularies collected in Task 2.3. In the following, we give an overview of the extraction process, and the connections to the Lynx project.

The general setting is a KG whose nodes are entities, and each of which has a list of labels, at least one per language of the project (EN, ES, DE, NL). These labels are strings which denote possible ways in which the entity is referred to in natural language. For example, an entity denoting “Office” could have labels “Office@EN”, “Bureau@EN”, “Oficina@ES”, “Buró@ES”, “Despacho@ES”, “Büro”@DE, “Kanzlei”@DE, “Kontor”@DE, “Kantoor”@NL, “Bureau”@NL, “Officie”@NL, where the @XY denotes the label in language XY. Each entity is assigned a URI with which it is identified across the LKG.

Furthermore, in order to bring about the graph property of a KG, the relationships between said entities must be specified. The simplest of such relationships, which form the basis of the special type of KG known as Thesaurus (or Taxonomy) is the “narrower/broader” relationship which denotes hyponymy, one word having a narrower meaning than the other, and which is often spelled out as “is a”. In the case of the entity “Office”, one such relation could be “Office is a Working Place” or “Patent Office is an Office”. Relation types (such as hyponymy) are assigned a URI, and relation instances are encoded in triples of URIs of the form <subject> <predicate> <object>.

Given a KG defined as described above, the Entity Extraction service amounts to finding mentions of these entities in a given document. While in the simplest case this could amount to a string-matching process, modern NLP techniques call for various degrees of processing to be done on the text, the list, or both, in order to improve the matching process. The EntEx service, for example, is able to recognize plurals such as “offices”@EN, or “Kanzleien”@DE. Once the matching has been done, PoolParty returns a list of entities found in a document, as well as their positions. This list of entities and their position is then assembled into a NIF file.

The EntEx service uses two parts of the PoolParty Semantic Suite. First, the thesaurus management capabilities are used to import, curate and maintain taxonomies, as well as to create any deemed necessary. This is done in a user-friendly way, allowing for domain-expert users without detailed technical knowledge to manage these taxonomies. By following the Semantic Web standards, these taxonomies can be exchanged between different parties, can be integrated into the larger LKG, and can be published as linked data.

The WSID service tackles the issue of ambiguity in natural languages. Indeed, four out of ten words in English have more than one meaning [Durkin1989]. The domain dependent vocabularies or terminologies prepared in the Lynx pilots are supposed to be domain-specific and, therefore, are not expected to cover possible out-of-domain senses of labels. In order to enable the usage of these terminologies for automatic text annotations, we introduce a robust method for inducing and discriminating word senses using synonyms and hypernyms. The method makes use of a pre-trained language models to generate potential substitutes for the target (matched label). Consider "We came into a bar and ordered two grasshoppers." In this case potential substitutes for "grasshoppers" would include "cocktails", "drinks", "wines", "beers". For the induction process several different contexts with the same target are considered, i.e. several contexts featuring "grasshopper" in different senses. Next, substitutes for each context are collected and then clustered in order to identify different senses.

For the disambiguation step the different induced clusters are used. The target is substituted by words from different clusters. The cluster that yields higher scores (each substitute is scored using the same pretrained language model) is assigned as the correct sense.

For WSID currently BERT (https://github.com/google-research/bert) is used as the pretrained language model. The pretrained BERT is available in more than 100 languages, therefore the methodology is inherently multilingual. Formal Concept Analysis factoring technique is used as the clustering method [Belohlavek2010].

2.1.2 Description of Service within Lynx

In a way, the EntEx service takes knowledge from the LKG's vocabularies and analyses a document using it. Due to the linking quality of the LKG, the knowledge about entities enlarges the knowledge of documents with the information "Document X contains a mention of entity A and entity B".

The Entity Extraction service is a prerequisite for several other services within Lynx. The already mentioned RelEx service, for example, first finds known entities in a document, and then recognizes whether a given relationship is being expressed about them. The Semantic Similarity (SeSim) service also makes great use of extracted entities, as they serve to compute similarity between snippets of text using not only the information explicitly contained in them, but also the information about that information which is part of the LKG. Likewise, the QA service uses extracted entities to enhance the information about the query and the documents from which answers are to be retrieved.

The WSID service succeeds EntEx in workflows and can also operate on NIF. It checks the entity annotation provided by EntEx and looks up for available models for the annotated entities. In case there are saved models, the method uses them to disambiguate the correct sense. The models should be prepared in advance in the induction step.

2.1.3 Evaluation

The WSI evaluation is performed on SemEval 2013 Task 13 dataset [Jurgens2013]. In order to know more about the used metrics see the respective paper.

Method	Fuzzy-NMI	Fuzzy-Bcubed F score	Geometric mean
[Wang2015]	7.14	55.4	19.89
[Amrami2018]	11.26	57.49	25.43
[Amplayo2019]	7.97	61.7	22.17
Our method	10.92	53.3	24.13

Table 1. Results of unsupervised evaluation on SemEval dataset.

The WSD evaluation is performed on the same dataset using the information about senses from the WordNet[1].

Method	Jaccard Index	Positionally-Weighted Tau	Weighted NDCG Recall
AI-KU (remove5-add1000)	0.245	0.642	0.332
UoS (#WN senses)	0.192	0.596	0.315
Unimelb (5p)	0.218	0.614	0.365
Our method	0.196	0.606	0.372

Table 2. Results of supervised evaluation on SemEval dataset.

2.2 DICTIONARY ACCESS

2.2.1 General Description of Method

The Lexicala API of K Dictionaries (KD) is a RESTful API that provides lexical data originating from lexicographic resources for 50 languages: the multilingual Global series (25 languages), the semi-bilingual Password series (40+ languages), and Random House Webster's College Dictionary (English). The API returns data in JSON format as well as – still to be completed – RDF data in JSON-LD (and N-Quads).

The resources provided by Lexicala consist of a list of words and senses for each word, with rich linguistic information about each, as well as examples of use. Some of these resources are categorized into knowledge fields.

Overall, the access process to the Lexicala API can be described in the following three steps:

1. Listing of all the languages available for search.
2. Searching, using free text, for a particular word in a particular language, getting back a list of IDs.
3. Getting detailed information about an entry or one of its senses, using the ID. This returns information like phonetics, elaborate syntactic and semantic information, related multiword phrases, usage examples and the translations available into other languages.

2.2.2 Description of Service within Lynx

The use of general dictionaries in Lynx is twofold:

1. To enrich controlled vocabularies with synonyms and translations. This is necessary as not all controlled vocabularies are multilingual, especially those which are extracted automatically using the Terminology Extraction service (described in deliverable 3.2).
2. The WSID service (described in this document), will use the synonyms contained in the dictionary for better clustering of free-text terms.
3. The Cross-lingual Search (SEAR) service uses Dictionary Access (DA) to expand the search queries.

[1] http://wordnetweb.princeton.edu/perl/webwn

2.2.3 Evaluation

In the case of manually compiled dictionary data, the evaluation is constant and takes form in work practice, rather than machine protocol, which ensures the quality of the data and its validity. The dictionaries are compiled by experienced editors and lexicographers, and the work of verifying that the dictionary is up-to-date is assisted by advanced corpus-based analysis tools, which enable constantly adding neologisms and phrases. As is the case for human-curated data, it is not possible to automatically evaluate the data under any standards, for one, due to the ever-changing and fluid nature of human language, which requires human supervision, and also due to the shortcomings of automated tools for natural language, which do not encapsulate the intricacies of natural language data well enough to evaluate it.

Instead, data is constantly revised and post-edited; tests are created to search for unique or problematic cases that require special attention, and periodic updates are applied to the headword lists, in order to reflect the language at its most recent form, flagging old-fashioned or obsolete entries along the way, and adding new meanings to existing ones.

2.3 DOCUMENT CLASSIFICATION

This service was abandoned as it was not required by any of the pilots.

2.4 QUESTION ANSWERING

This service was moved to task 3.4 and its description is to be found in Deliverable 3.9.

3 EXTRACTION SERVICES

3.1 RELATION EXTRACTION

3.1.1 General Description of Method

The RelEx service aims at extracting relations between entities in plain text. For example, in the string "Michael Jackson was born in the USA." a relation between "Mickael Jackson" and "USA" is expressed, we could call this relation "Entity-Origin".

The relations could be expressed in various ways in the text. Usually a verb is used, however one easily finds multiple examples where a relation is expressed without using verbs. Therefore, methods based on curated rules and linguistic parsing might not give satisfactory results. We have decided for a methodology based on pre-trained deep neural networks, like ELMO (https://github.com/HIT-SCIR/ELMoForManyLangs), BERT (https://github.com/google-research/bert) or OpenAI GPT (https://github.com/openai/gpt-2). The advantage of this methodology is that the pre-trained models not only vectorize the input sequence (tokens, sub-tokens or even characters), but also take the preceding and the following context into account, therefore these classifiers are able to deal efficiently with negations, synonyms, paraphrasing, etc.

We are currently using the OpenAI GPT2 model fine-tuned on SemEval 2010 Task 8 [HendrickxEtAl2009] dataset (https://github.com/DFKI-NLP/TRE). The model is able to recognize 8 directed relations:

- Cause-Effect (CE). An event or object leads to an effect. Example: *those **cancers** were caused by **radiation exposures***
- Instrument-Agency (IA). An agent uses an instrument. Example: ***phone operator***
- Product-Producer (PP). A producer causes a product to exist. Example: *a **factory** manufactures **suits***
- Content-Container (CC). An object is physically stored in a delineated area of space. Example: *a **bottle** full of **honey** was weighed*
- Entity-Origin (EO). An entity is coming or is derived from an origin (e.g., position or material). Example: ***letters** from foreign **countries***
- Entity-Destination (ED). An entity is moving towards a destination. Example: *the **boy** went to **bed***
- Component-Whole (CW). An object is a component of a larger whole. Example: *my **apartment** has a large **kitchen***
- Member-Collection (MC). A member forms a nonfunctional part of a collection. Example: *there are many **trees** in the **forest***
- Message-Topic (MT). A message, written or spoken, is about a topic. Example: *the **lecture** was about **semantics***
- If none of these relations is found the system output the relation "Other".

The system is also capable of recognizing the directionality of relation, i.e. the response contains the order of the entities.

3.1.2 Description of Service within Lynx

We provide an HTTP API endpoints to communicate with the service. In the course of the pilot work we aim at training new models that are specific for the data used in pilots. Therefore, in the call parameters the user is able to choose the model to be used for extracting the relations.

The entities are expected to be recognized before the input is sent to the RelEx service, therefore EntEx service precedes the RelEx service in the workflows.

A report on available HTTP endpoints with their descriptions is available in the appendix.

3.1.3 Evaluation

The model is evaluated on SemEval 2010 Task 8 and reaches the F1 score of 87.1.

3.2 MONOLINGUAL TERMINOLOGY EXTRACTION

This service has been moved to Task 3.1 and its description can be found in Deliverable 3.7.

3.3 DOCUMENT STRUCTURE EXTRACTION

The implementation of this service was evaluated and turned out to be infeasible in frames of the Lynx project scope. The service proved to be very complex on its own as it should be able to recognize the structure of .pdf documents automatically. Since such a task is not in the core of Lynx project it was decided to do the document parsing with pre-specified parser that are specific to each pilot as we only have a limited number of document types per pilot.

3.4 ENTITY EXTRACTION

This service has been merged with the WSID service. The description is to be found in this document in Section 2.1.

4 CONCLUSION

This deliverable features 3 services that are developed in frames of Task 3.3: RelEx, DA, EL. All the services have an OpenAPI interface. RelEx and EL take NIF documents as input and output also NIF documents. The services are deployed and ready for integration.

REFERENCES

[HendrickxEtAl2009] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2009. SemEval-2010 task 8: multi-way classification of semantic relations between pairs of nominals. In Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions (SEW '09). Association for Computational Linguistics, Stroudsburg, PA, USA, 94-99.

[Durkin1989] Durkin, K., & Manning, J. (1989). Polysemy and the subjective lexicon: Semantic relatedness and the salience of intraword senses. Journal of Psycholinguistic Research, 18(6), 577-612.

[Belohlavek2010] Radim Belohlavek, Vilem Vychodil, Discovery of optimal factors in binary data via a novel method of matrix decomposition, Journal of Computer and System Sciences, Volume 76, Issue 1, 2010, Pages 3-20, ISSN 0022-0000, https://doi.org/10.1016/j.jcss.2009.05.002.

[Jurgens2013] Jurgens D, Klapaftis I. Semeval-2013 task 13: Word sense induction for graded and non-graded senses. InSecond Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013) 2013 Jun (pp. 290-299).

[Wang2015] Wang, J., Bansal, M., Gimpel, K., Ziebart, B.D., & Yu, C.T. (2015). A Sense-Topic Model for Word Sense Induction with Unsupervised Data Enrichment. Transactions of the Association for Computational Linguistics, 3, 59-71.

[Amplayo2019] Amplayo, R. K., Hwang, S., & Song, M. (2019). AutoSense Model for Word Sense Induction. Proceedings of the AAAI Conference on Artificial Intelligence, 33, 6212–6219. https://doi.org/10.1609/aaai.v33i01.33016212

[Amrami2018] Amrami, A., & Goldberg, Y. (2018). Word Sense Induction with Neural biLM and Symmetric Patterns. Proceedings Of the 2018 Conference on Empirical Methods in Natural Language Processing, 4860–4867. https://doi.org/10.18653/v1/d18-1523

ANNEX 1 – IMPLEMENTATION DETAILS (CODE)

In the appendix one finds a report on OpenAPI description of the services' HTTP endpoints. The tool https://mrin9.github.io/RapiPdf/ was used to generate these reports.

API Reference

WSID API

API Version: 0.1

Word Sense Induction and Disambiguation service

This service has 2 main functionalities:

1. Inducing the senses of words in a given corpus;
2. Disambiguates word senses in a given previously unseen context.

The current implementation factorizes substitutes generated by a pre-trained language model (currently [BERT] (https://github.com/google-research/bert) is used).

How to use as a standalone web service

Navigate to `<host>/` and you will be redirected to the swagger interface of the webservice.

Induce call Provide several contexts (at least 3 for each meaning you would like to induce). For each context specify the start index and the end index of the target. Optionally you can provide a target to check which contexts are representative for the induced senses. Please, check the Swagger interface for the exact `json` structure. The output is an array of induced senses; each sense contains generating contexts and 5 descriptive words for this sense.

Disambiguate call Provide a context, start and end indices, and an array of senses. Each sense is in its turn an array of words (sense descriptors) that describe this sense. The service returns the aggregated score of the sense descriptors for each sense. The higher the score the more likely that this sense is used. The descriptors might be either induced (see Induce call) or taken from elsewhere.

Example workflow

The following describes the full workflow. If you already have induced and/or send sense inventory you may skip the corresponding steps.

We consider the case that the user has a certain *target word*.

```
curl -X POST "http://wsid-88-dev-int.cloud.itandtel.at/wsd/api/lm/induce/?
top_n_predictions=55&n_sense_indicators=5" -H "accept: application/json" -H "Content-
Type: application/json" -H "X-CSRFToken:
WCrrUzvdvbA4uahbunqIJGxTpyAwFuIGgIm9O91EfeiQwH3TnUUsnF2cdXkHXi94" -d "[ {\"context\":
\"Find a qualified lawyer in Milwaukee. Finding your way through the legal system is
difficult — we can help. For more than 160 years the Milwaukee Bar Association has
```

helped people just like you.\", \"start_ind\": 146, \"end_ind\": 149, \"title\": \"legal 1\"}, {\"context\": \"The only formal education I had before I began the California State Bar Law Office Study Program, was some foreign language classes, a stint in the United States Marines, and some useless typing lessons at a community college\", \"start_ind\": 68, \"end_ind\": 71, \"title\": \"legal 2\"}, {\"context\": \"I sat for the California Bar Examination 36 months or so later. But this was only after dedicated training under several skilled attorneys in various types of practices.\", \"start_ind\": 26, \"end_ind\": 29, \"title\": \"legal 3\"}, {\"context\": \"No special requirements are needed to become an attorney without law school in Virginia or Washington as far as I could determine. All of the above information came from the 'Comprehensive Guide to Bar Admission Requirements 2004.'\", \"start_ind\": 198, \"end_ind\": 201, \"title\": \"legal 4\"}, {\"context\": \"Importance of the Bar Exam Itself – ABA Resistance to Law Office Study.\", \"start_ind\": 18, \"end_ind\": 21, \"title\": \"legal 5\"}, {\"context\": \"\Next, the Bar Candidate must take and pass the First Year Law Students Exam (FYLSX) within the first three (3) attempts.\", \"start_ind\": 10, \"end_ind\": 13, \"title\": \"legal 6\"}, {\"context\": \"Law schools teach general principles and are often comprised of Socratic types of classroom discussions that would do little to help a person pass the Bar Exam, in my opinion.\", \"start_ind\": 151, \"end_ind\": 154, \"title\": \"legal 7\"}, {\"context\": \"And when I was there, I could not understand why the essays weren't structured more like Bar Exam questions.\", \"start_ind\": 89, \"end_ind\": 92, \"title\": \"legal 8\"}, {\"context\": \"THE TEMPLE BAR WAS FOUNDED IN 1840 AS A WINE AND SPIRIT MERCHANT SHOP AND IS STEEPED IN THE TRADITION OF HOSPITALITY.\", \"start_ind\": 11, \"end_ind\": 14, \"title\": \"common 1\"}, {\"context\": \"TOM CLEARY, OWNER OF THE TEMPLE BAR, IS PASSIONATE ABOUT WHISKEY AND HIS FAMILY HAVE BEEN BLENDING AND BOTTLING WHISKEY FOR GENERATIONS.\", \"start_ind\": 32, \"end_ind\": 35, \"title\": \"common 2\"}, {\"context\": \"Well, show me the way To the next whisky bar Oh, don't ask why\", \"start_ind\": 41, \"end_ind\": 44, \"title\": \"common 3\"}, {\"context\": \"Vodka is the most mixable and useful liquor in the bar.\", \"start_ind\": 51, \"end_ind\": 54, \"title\": \"common 4\"}, {\"context\": \"Popped in here because you can't come to krakow and not go here! Cute little bar. Obviously had to get the vodka shot flights!\", \"start_ind\": 77, \"end_ind\": 80, \"title\": \"common 5\"}, {\"context\": \"Great little bar with wodka aplenty\", \"start_ind\": 13, \"end_ind\": 16, \"title\": \"common 6\"}, {\"context\": \"Beer Bar is a love letter to the beer of the world.\", \"start_ind\": 5, \"end_ind\": 8, \"title\": \"common 7\"}]"

1. Collect a corpus of short contexts where the *target word* occurs. One sentence with more than 7 words is OK as a context. It is desirable that every sense is present in at least 3 contexts. And the more balanced the distribution of sense is, the better results one can expect.
2. For an example of the context corpus see `django_wsd/test_data/bar.json` - a corpus containing 15 contexts: 7 with "bar" as a place and 8 with "bar" as a legal certification.
3. Do the sense induction, i.e. call `induce` endpoint. Example:

The actual response depends on several factors, but it should look similar to


```
[{"uri": "",
  "extent": ["legal 5", "legal 7", "legal 6"],
  "sense_descriptors": ["school", "law", "college", "university", "student"],
  "lemma": ""},
 {"uri": "",
  "extent": ["common 1", "common 2", "common 4"],
  "sense_descriptors": ["industry", "village", "district", "market", "family"],
  "lemma": ""}]
```


```
curl -X POST "http://wsid-88-dev-int.cloud.itandtel.at/wsd/api/lm/sense_info/" -H
"accept: application/json" -H "Content-Type: application/json" -H "X-CSRFToken:
WCrrUzvdvbA4uahbunqIJGxTpyAwFuIGgIm9O91EfeiQwH3TnUUsnF2cdXkHXi94" -d "[{\"uri\":
\"http://lynx-project.com/resource/bar_legal\", \"extent\": [\"legal 5\", \"legal
7\", \"legal 6\"], \"sense_descriptors\": [\"school\", \"law\", \"college\",
\"university\", \"student\"], \"lemma\": \"bar\"}, {\"uri\": \"\", \"extent\":
[\"common 1\", \"common 2\", \"common 4\"], \"sense_descriptors\": [\"industry\",
```

```
\"village\", \"district\", \"market\", \"family\"], \"lemma\": \"bar\"}]"
```

3. Save the desired senses and POST to `sense_info`:
4. For this purpose you take the previous output and populate `uri`s and `lemma`s.
5. Notice that we leave the `uri` of the second sense empty. The `extent` might be kept as is, it is ignored.

Optional you can GET `sense_info` to check that the senses are saved.

4. Disambiguate. We show an example of disambiguating NIF document.

The document itself:

```
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix nif:   <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#> .

<http://dkt.dfki.de/documents#offset_51_54>
    a                    nif:RFC5147String , nif:String , nif:Structure ;
    nif:anchorOf         "bar" ;
    nif:beginIndex       "51"^^xsd:nonNegativeInteger ;
    nif:endIndex         "54"^^xsd:nonNegativeInteger ;
    nif:referenceContext  <http://dkt.dfki.de/documents#offset_0_55> .

<http://dkt.dfki.de/documents#offset_0_55>
    a               nif:RFC5147String , nif:String , nif:Context ;
    nif:beginIndex  "0"^^xsd:nonNegativeInteger ;
    nif:endIndex    "55"^^xsd:nonNegativeInteger ;
    nif:isString    "Vodka is the most mixable and useful liquor in the bar." .
```

The call:

```
curl -X POST "http://wsid-88-dev-int.cloud.itandtel.at/wsd/api/lm/disambiguate_nif/"
-H "accept: text/turtle" -H "Content-Type: text/turtle" -H "X-CSRFToken:
WCrrUzvdvbA4uahbunqIJGxTpyAwFuIGgIm9O91EfeiQwH3TnUUsnF2cdXkHXi94" -d "@prefix xsd:
<http://www.w3.org/2001/XMLSchema#> .@prefix nif: <http://persistence.uni-
leipzig.org/nlp2rdf/ontologies/nif-core#> .<http://dkt.dfki.de/
documents#offset_51_54> a nif:RFC5147String , nif:String , nif:Structure ;
nif:anchorOf \"bar\" ; nif:beginIndex \"51\"^^xsd:nonNegativeInteger ; nif:endIndex
\"54\"^^xsd:nonNegativeInteger ; nif:referenceContext <http://dkt.dfki.de/
documents#offset_0_55> .<http://dkt.dfki.de/documents#offset_0_55> a
nif:RFC5147String , nif:String , nif:Context ; nif:beginIndex
\"0\"^^xsd:nonNegativeInteger ; nif:endIndex \"55\"^^xsd:nonNegativeInteger ;
nif:isString \"Vodka is the most mixable and useful liquor in the bar.\" ."
```

And the output should look like:

```
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix nif:   <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#> .

<http://dkt.dfki.de/documents#offset_0_55>
    a               nif:RFC5147String , nif:String , nif:Context ;
    nif:beginIndex  "0"^^xsd:nonNegativeInteger ;
    nif:endIndex    "55"^^xsd:nonNegativeInteger ;
    nif:isString    "Vodka is the most mixable and useful liquor in the bar." .
```

As the correct "pub" sense of "bar" was saved without any `uri` it is considered as undesired sense, hence the respective structure is removed.
If the `uri` would not be empty, the value of that `uri` and the confidence value would have been added to the annotation.

5. Extract and disambiguate.

First we do the extract call to check the pure extraction results:

```
curl -X POST "http://wsid-88-dev-int.cloud.itandtel.at/wsd/api/ext/extract/?
server_url=https%3A%2F%2Flynx.poolparty.biz&project_id=1E0347CA-8950-0001-EDEF-
C630138A45F0&username={your PP user}&password={youe PP password}&language=en" -H
"accept: text/turtle" -H "Content-Type: text/plain" -H "X-CSRFToken:
WCrrUzvdvbA4uahbunqIJGxTpyAwFuIGgIm9O91EfeiQwH3TnUUsnF2cdXkHXi94" -d "Vodka is useful
in a bar."
```

The result:

```
@prefix ns1: <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#> .
@prefix ns2: <https://www.w3.org/2005/11/its/rdf-content/its-rdf.html#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<https://lynx.poolparty.biz#offset_0_5> a ns1:Annotation,
        ns1:OffsetBasedString ;
    ns1:anchorOf "vodka" ;
    ns1:annotationUnit [ ns2:taIdentRef <http://vocabulary.semantic-web.at/
cocktails/22df810d-e509-4da8-979c-0758e94de3e6> ] ;
    ns1:beginIndex 0 ;
    ns1:endIndex 5 ;
    ns1:referenceContext <https://lynx.poolparty.biz#offset_0_25> .

<https://lynx.poolparty.biz#offset_21_24> a ns1:Annotation,
        ns1:OffsetBasedString ;
    ns1:anchorOf "bar" ;
    ns1:annotationUnit [ ns2:taIdentRef <https://lynx.poolparty.biz/Cocktails/1> ] ;
    ns1:beginIndex 21 ;
    ns1:endIndex 24 ;
    ns1:referenceContext <https://lynx.poolparty.biz#offset_0_25> .

<https://lynx.poolparty.biz#offset_0_25> a ns1:Context,
        ns1:OffsetBasedString ;
    ns1:beginIndex 0 ;
    ns1:endIndex 25 ;
    ns1:isString "Vodka is useful in a bar." .
```

The extract & disambiguate call

```
curl -X POST "http://wsid-88-dev-int.cloud.itandtel.at/wsd/api/ext/
extract_disambiguate/?server_url=https%3A%2F
%2Flynx.poolparty.biz&project_id=1E0347CA-8950-0001-EDEF-C630138A45F0&username={your
PP user}&password={your PP password}&language=en" -H "accept: text/turtle" -H
"Content-Type: text/plain" -H "X-CSRFToken:
WCrrUzvdvbA4uahbunqIJGxTpyAwFuIGgIm9O91EfeiQwH3TnUUsnF2cdXkHXi94" -d "Vodka is useful
in a bar."
```

And the expected output is:

```
@prefix ns1: <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#> .
@prefix ns2: <https://www.w3.org/2005/11/its/rdf-content/its-rdf.html#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<https://lynx.poolparty.biz#offset_0_5> a ns1:Annotation,
        ns1:OffsetBasedString ;
    ns1:anchorOf "vodka" ;
    ns1:annotationUnit [ ns2:taIdentRef <http://vocabulary.semantic-web.at/
```

```
cocktails/22df810d-e509-4da8-979c-0758e94de3e6> ] ;
    ns1:beginIndex 0 ;
    ns1:endIndex 5 ;
    ns1:referenceContext <https://lynx.poolparty.biz#offset_0_25> .

<https://lynx.poolparty.biz#offset_0_25> a ns1:Context,
        ns1:OffsetBasedString ;
    ns1:beginIndex 0 ;
    ns1:endIndex 25 ;
    ns1:isString "Vodka is useful in a bar." .
```

Note that "bar" is not annotated, though it is in the project and returned by PoolParty. This is the result of disambiguation.

Hot to run as a standalone web service

```
apt install docker.io docker-compose
```

1. Install [Docker](https://www.docker.com/) and [`docker-compose`](https://docs.docker.com/compose/)
2. Download your favorite BERT model (see https://github.com/huggingface/pytorch-pretrained-BERT/blob/master/pytorch_pretrained_bert/modeling.py) to `<some folder>` and
3. change `LOCAL_BERT_PATH` in `.env` to point to `<some folder>`.

Dev

```
export $(cat wsid_web_service/settings/.env | grep -v "#" | xargs)
docker-compose -f ./docker-compose-dev.yml up -d
```

3. Load env vars into your environment
4. Compose your docker

Optional:

- Load your fixtures If you wish to load any data using fixtures, dump it to `fixtures/latest.json`.

How to install as a django app

You might install this app into your project.

```
url(r'^wsd/', include('django_wsd.urls', namespace="wsd"))
```

1. Add the app `'django_wsd.apps.DjangoWSDConfig'` to your `INSTALLED_APPS` in settings.
2. Add urls to your `urls.py`:

TODO

- Add examples using EntEx + WSID

Notes

- `docker-compose-prod.yml` is out-of-date. Is it needed now?
- `/config` might be also out-of-date.

CONTACT

NAME: LYNX Project API team
EMAIL: apiteam@lynx-project.eu
URL: https://lynx-project.eu
Terms of service: http://lynx-project.eu/terms/

INDEX

Security and Authentication

SECURITY SCHEMES

KEY	TYPE	DESCRIPTION
Basic	http, basic	

API

1. EXT

1.1 POST /ext/extract/

Extract entities

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
server_url	uri min:1 chars	URL of the server
*project_id	string min:1 chars	Project ID in PP
*username	string min:1 chars	Your username
*password	string min:1 chars	Your password
language	string min:1 chars	en|de|es|nl

RESPONSE

STATUS CODE - 200: NIF RDF document

STATUS CODE - 400: Bad request.

STATUS CODE - 500: Internal server error.

1.2 POST /ext/extract_disambiguate/

Extract entities

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
server_url	uri min:1 chars	URL of the server
*project_id	string min:1 chars	Project ID in PP
*username	string min:1 chars	Your username
*password	string min:1 chars	Your password
language	string min:1 chars	en|de|es|nl

RESPONSE

STATUS CODE - 200: NIF RDF document

STATUS CODE - 400: Bad request.

STATUS CODE - 500: Internal server error.

2. LM

2.1 POST /lm/disambiguate_demo/

Demo of disambiguation with manual input

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*context	string min:1 chars	
title	string min:1 chars	Optional title
*start_ind	integer >=0	Position in the context where the target word starts.
*end_ind	integer	Position in the context where the target word ends.
*senses	array of string	Sense descriptors (use ";" to delimit and do not use commas)

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

STATUS CODE - 400: Bad request, error information

STATUS CODE - 500: Internal server error.

2.2 POST /lm/disambiguate_nif/

Disambiguate in context

Disambiguate tokens in the given context.

REQUEST

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - text/turtle

STATUS CODE - 400: Bad request, error information

STATUS CODE - 500: Internal server error.

2.3 POST /lm/induce/

Word Sense Induction

Induce word senses of the target word or words.

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
top_n_predictions	integer	Number of substitutes per occurrence
n_sense_indicators	integer	Number of sense indicators per induced sense

REQUEST BODY - application/json

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
ARRAY OF OBJECT WITH BELOW STRUCTURE		
uri	string	max:200 chars
extent	array	
sense_descriptors*	array	
lemma	string	

STATUS CODE - 400: Bad request.

STATUS CODE - 500: Internal server error.

2.4 GET /lm/sense_info/

Get the saved senses for the label of entity

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*lemma	string	The ambiguous label

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
uri	string	max:200 chars
extent	array	
sense_descriptors*	array	
lemma	string	

STATUS CODE - 400: Bad request, error information

STATUS CODE - 500: Internal server error.

2.5 POST /lm/sense_info/

Save provided senses

REQUEST

REQUEST BODY - application/json

RESPONSE

STATUS CODE - 200: Success

STATUS CODE - 400: Bad request, error information

STATUS CODE - 500: Internal server error.

2.6 DELETE /lm/sense_info/

Delete senses for the label and uri

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*lemma	string	The ambiguous label
uri	uri	URI. Leave blank to delete *all* senses without URIs.

RESPONSE

STATUS CODE - 200:

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
uri	string	max:200 chars
extent	array	
sense_descriptors*	array	
lemma	string	

STATUS CODE - 400: Bad request, error information

STATUS CODE - 500: Internal server error.

API Reference

Lexicala API

API Version: 0.0.1

The Lexicala API provides access to lexicographic data from K Dictionaries' global series of lexicographic resources.

CONTACT

NAME: API Support
EMAIL: api@kdictionaries.com

INDEX

API

1. DATA

1.1 GET /languages

list all supported languages

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
  language_names* {
  }
  resources* {
    global   [string]
    password [string]
  }
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

1.2 GET /search

search for entries with filters in query

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
source	enum **ALLOWED:** global, password, random	The resource to search within
language	string	The language code of the entry's language
morph	boolean	Whether to search in headword inflections if exist
text	string	The headword text to search for
regex	boolean	Whether given text is a regular expression
gender	string	The grammatical gender to search for
pos	string	The part of speech (POS) to search for
number	string	The grammatical number to search for

NAME	TYPE	DESCRIPTION
subcategorization	string	The subcategorization (e.g. countable, transitive, ...) to search for
monosemous	boolean	Whether to only return monosemous entries
polysemous	boolean	Whether to only return polysemous entries
sample	integer	Number of randomly-sampled results to return
page	integer >=1	Page number of results to return (1-indexed)
page-length	integer >=1	Number of results to display per page
analyzed	boolean	Whether to search using language analyzer or to get exact matches only

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
string
```

STATUS CODE - 403: Forbidden

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

1.3 GET /entries/{entry_id}

get entry by its ID

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*entry_id	string	The entry ID of the entry

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
```

```
  id* string
}
```

STATUS CODE - 403: Forbidden

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 404: Entry not found

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

1.4 GET /senses/{sense_id}

get sense by its ID

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*sense_id	string	The sense ID of the sense

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
}
```

STATUS CODE - 403: Forbidden

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 404: Sense not found

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

2. TESTS

2.1 GET /

Root path

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
  message* string
}
```

STATUS CODE - 503: Service Unavailable

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

2.2 GET /test

Test that API is running

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
  message* string
}
```

STATUS CODE - 503: Service Unavailable

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

---

3. USERS

3.1 GET /users

get list of users

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

[{
Array of object:
}]

STATUS CODE - 403: Forbidden

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

3.2 POST /users

add new user

REQUEST

REQUEST BODY - application/json

```
{
  username* string
  password* string
  email*    string
  permissions* {
  }
}
```

RESPONSE

STATUS CODE - 201: Created

RESPONSE MODEL - application/json

{

```
  id*        string
  username*  string
  email*     string
  permissions* {
  }
}
```

STATUS CODE - 400: Bad request

RESPONSE MODEL - application/json

```
{
  status*   integer
  message*  string
}
```

STATUS CODE - 403: Forbidden

RESPONSE MODEL - application/json

```
{
  status*   integer
  message*  string
}
```

STATUS CODE - 409: User already exists

RESPONSE MODEL - application/json

```
{
  status*   integer
  message*  string
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*   integer
  message*  string
}
```

3.3 GET /users/{username}

get user by username

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*username	string	The username of the user

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
  id*        string
```

```
  username* string
  email*    string
  permissions* {
  }
}
```

STATUS CODE - 403: Forbidden

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 404: User not found

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

3.4 DELETE /users/{username}

delete user account

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*username	string	The username of the user

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
}
```

STATUS CODE - 403: Forbidden

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 404: User not found

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

3.5 PATCH /users/{username}

update user account

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*username	string	The username of the user

REQUEST BODY - application/json

```
{
  username string
  email    string
  password string
  permissions {
  }
}
```

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
  id*       string
  username* string
  email*    string
  permissions* {
  }
}
```

STATUS CODE - 400: Bad request

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

STATUS CODE - 403: Forbidden

RESPONSE MODEL - application/json

```
{
  status*   integer
  message*  string
}
```

STATUS CODE - 404: User not found

RESPONSE MODEL - application/json

```
{
  status*   integer
  message*  string
}
```

STATUS CODE - 409: User with username given in user_updates already exists

RESPONSE MODEL - application/json

```
{
  status*   integer
  message*  string
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*   integer
  message*  string
}
```

3.6 GET /users/me

get currently authenticated user

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Success

RESPONSE MODEL - application/json

```
{
  id*        string
  username*  string
  email*     string
  permissions* {
  }
}
```

STATUS CODE - 404: User not found

RESPONSE MODEL - application/json

```
{
  status*   integer
  message*  string
```

```
}
```

STATUS CODE - 500: Internal Server Error

RESPONSE MODEL - application/json

```
{
  status*  integer
  message* string
}
```

API Reference

Relation Extraction HTTP APIs

API Version: 1.0.0

Relation Extraction Web Service

This repository contains a code for running a relation extraction web service.
The repository containing the code and additional information can be found at https://gitlab.com/superlynx/relex-webapp.

Input -> Output

The input is a piece of text with extracted entities. The response contains the relation between the pairs of entities. The relations are always binary and the model is pre-trained.

Currently only one model is supported: [SemEval 2010 Task 8](https://aclweb.org/anthology/S10-1006), it contains following relations:

- Cause-Effect (CE). An event or object leads to an effect. Example: *those cancers were caused by radiation exposures
- Instrument-Agency (IA). An agent uses an instrument. Example: phone operator
- Product-Producer (PP). A producer causes a product to exist. Example: *a factory manufactures suits
- Content-Container (CC). An object is physically stored in a delineated area of space. Example: *a bottle full of honey was weighed*
- Entity-Origin (EO). An entity is coming or is derived from an origin (e.g., position or material). Example: letters** from foreign **countries
- Entity-Destination (ED). An entity is moving towards a destination. Example: *the boy went to bed
- Component-Whole (CW). An object is a component of a larger whole. Example: *my apartment has a large kitchen
- Member-Collection (MC). A member forms a nonfunctional part of a collection. Example: *there are many trees in the forest
- Message-Topic (MT). A message, written or spoken, is about a topic. Example: *the lecture was about semantics
- If none of these relations is found the system output the relation "Other". The system is also capable of recognizing the directionality of relation, i.e. the response contains the order of the entities.

Example of Usage

We demonstrate the usage of `extract_from_nif` call. The original NIF document looks as follows:

```
@prefix ns1: <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

```
<http://dkt.dfki.de/documents#offset_18_27> a ns1:Annotation,
        ns1:OffsetBasedString;
    ns1:anchorOf "survivors" ;
    ns1:beginIndex 18 ;
    ns1:endIndex 27 ;
    ns1:referenceContext <http://dkt.dfki.de/documents#offset_0_56> .
<http://dkt.dfki.de/documents#offset_39_55> a ns1:Annotation,
        ns1:OffsetBasedString;
    ns1:anchorOf "makeshift houses" ;
    ns1:beginIndex 39 ;
    ns1:endIndex 55 ;
    ns1:referenceContext <http://dkt.dfki.de/documents#offset_0_56> .
<http://dkt.dfki.de/documents#offset_0_56> a ns1:Context,
        ns1:OffsetBasedString;
    ns1:beginIndex 0 ;
    ns1:endIndex 56 ;
    ns1:isString "Ten million quake survivors moved into makeshift houses." .
```

The input for the `extract_from_nif` is provided as JSON body, therefore the NIF document has to be JSON escaped. The resulting cURL call:

```
curl -X POST "http://localhost:5000/v0/extract_from_nif?
model_name=SemEval2010&return_original=true"
  -H "accept: application/json" -H "Content-Type: application/json"
  -d "{\"nif_document_turtle\":\"@prefix ns1: <http://persistence.uni-leipzig.org/
nlp2rdf/ontologies/nif-core#> .\\r\@prefix rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#> .\\r\@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .\\r\@prefix
xml: <http://www.w3.org/XML/1998/namespace> .\\r\@prefix xsd: <http://
www.w3.org/2001/XMLSchema#> .\\r\<http://dkt.dfki.de/documents#offset_18_27> a
ns1:Annotation,\\r\ ns1:OffsetBasedString;\\r\ ns1:anchorOf \\\"survivors\\\" ;\\r\
ns1:beginIndex 18 ;\\r\ ns1:endIndex 27 ;\\r\ ns1:referenceContext <http://
dkt.dfki.de/documents#offset_0_56> .\\r\<http://dkt.dfki.de/documents#offset_39_55> a
ns1:Annotation,\\r\ ns1:OffsetBasedString;\\r\ ns1:anchorOf \\\"makeshift houses\\\"
;\\r\ ns1:beginIndex 39 ;\\r\ ns1:endIndex 55 ;\\r\ ns1:referenceContext <http://
dkt.dfki.de/documents#offset_0_56> .\\r\<http://dkt.dfki.de/documents#offset_0_56> a
ns1:Context,\\r\ ns1:OffsetBasedString;\\r\ ns1:beginIndex 0 ;\\r\ ns1:endIndex 56 ;\
\r\ ns1:isString \\\"Ten million quake survivors moved into makeshift houses.\\\" .
\"}"
```

INDEX

API

1. RELEX_WEBAPP

Relation extraction.

1.1 POST /v0/extract_from_nif

```
Extract from a NIF document using the annotations inside NIF.
Do not forget to do the JSON escaping of the `nif_document_turtle`.
```

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
model_name	enum ALLOWED: SemEval2010	Pre-trained model for relation extraction.
return_original	boolean	Whether opriginal NIF annotations and the context should be returned.

REQUEST BODY - application/json

```
{
  nif_document_turtle* string  NIF document in turtle notation
}
```

RESPONSE

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  errors {
  }
  message* string
}
```

1.2 GET /v0/extract_from_string_with_entities

```
This call expects the plain text input and the entities annotated inside
the text.

The input string must contain mentions of two entities. Use tags <e1> and <e2>
to point to the entities. Do not forget to close the tags. The entities might not
overlap.
Example input: Ten million quake <e1>survivors</e1> moved into <e2>makeshift houses</
e2>.
```

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*text	string	Text with annotated entities

RESPONSE

STATUS CODE - 200: RelationsSchema

RESPONSE MODEL - application/json

```
{
  e1        string
  e2        string
  relation  string
  support   string  The text where the relation was found.
}
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  errors {
  }
  message*  string
}
```

1.3 POST /v0/extract_from_text

This call expects a plain text input, a PoolParty server URL, a project ID, credentials.
The input text is annotated by PoolParty using the provided project.
Then the acquired annotations are used to identify entities and the relations between those entities are extracted.

REQUEST

QUERY PARAMETERS

NAME	TYPE	DESCRIPTION
*pp_password	string	PoolParty password
*pp_project_id	string	Project ID inside PoolParty server
*pp_server_url	string	URL of PoolParty server
*pp_username	string	PoolParty username
*text	string	text

RESPONSE

STATUS CODE - 200: RelationsSchema

RESPONSE MODEL - application/json

```
[{
Array of object:
  e1        string
  e2        string
  relation  string
  support   string  The text where the relation was found.
}]
```

STATUS CODE - default: Error

RESPONSE MODEL - application/json

```
{
  errors {
  }
  message* string
}
```