KARP'S BACKEND

SPRÅKBANKEN

GÖTEBORGS UNIVERSITET

CONTENTS

1 ABOUT KARP

Karp is Språkbanken's lexical infrastructure which is used for publishing and editing lexical resources. Karp's backend allows you to make your lexicons searchable and editable. The lexicons must be in json format, but it is possible to have for example xml blocks in the json objects, which can still be made searchable.

There are two main types of searches available, simple (free text search) and extended (field search). In the free text search, the content of a predefined set of fields is used, as defined when you set the configuration files. In the extended search, the user can choose which specific fields to search, eg. baseform or part of speech. Which fields are searchable and how the text in them is analyzed is also defined during the configuration.

The backend is a WSGI application written in Python. The main components of the system are ElasticSearch and SQL. ElasticSearch (ES) provides fast searching and an easy way of indexing and analyzing your data. The SQL database is used only as a back-up and for keeping the revision history of edited resources.

The code base for Karp's backend contains one example resource, the Bilingual Glossary German-English with Probabilities (PANACEA) created by Linguatec GmbH, which can be used for testing your Karp installation.

2 INPUT FORMAT

Each lexical entry must be represented as one json object, and contain the name of the lexicon ("lexiconName"). It should also contain the order of the lexicons ("lexiconOrder"), which regulates in which order the search results are shown[1]. This should be an integer, starting from `0`. A lexicon is a list of objects. A simple example:

```
[{'lexiconName': 'saldo', 'lexiconOrder': 0, 'baseform': 'katt', 'partOfSpeech': 'nn',
  'xml': '<definition>Some <b>dumped</b> data</definition>'},
 {'lexiconName': 'saldo', 'lexiconOrder': 0, 'baseform': 'hund', 'partOfSpeech': 'nn'},
 {'lexiconName': 'lexin', 'lexiconOrder': 1, 'baseform': 'hund',
  'partOfSpeech': ['nn','vb'], 'lexinID': 'lx500'}
]
```

The entries may of course have a much more complex structure, and contain lists, other objects etc. ES will allow the type of a field to vary between lists and single objects, but not between other any two types. The above example is thus ok, even though 'partOfSpeech' varies in type, but the below is not:

```
[{'lexiconName': 'saldo', 'lexiconOrder': 0, 'baseform': 'katt', 'partOfSpeech': 'nn'},
 {'lexiconName': 'lexin', 'lexiconOrder': 1, 'baseform': 'hund',
  'partOfSpeech': {'swe': 'nn', 'eng': 'vb'}}
]
```

Your work will be simplified if all entries are similarly structured, but it is not required.

1 A standard use case of the Karp backend is to group all search results by lexicon. This also corresponds to the set-up in the frontend. You will also have to specify the order of your lexicons later in the configuration part.

3 PREREQUISITES

To run the Karp backend, you will need to install a number of other packages. If you are running Karp for test and development, you can get an easy start by using Docker[2]. If so, skip this section and continue at section 4.

- ElasticSearch
  Download version 1
  https://www.elastic.co/downloads/past-releases/elasticsearch-1-5-2
  Install as described here https://www.elastic.co/downloads/elasticsearch
  The only configuration needed is to set `cluster.name` in `config/elasticsearch.yml` to a desired name[3]

- Virtual Env
  For the python libraries. Install with:
  `pip install virtualenv`
  http://docs.python-guide.org/en/latest/dev/virtualenvs/

- SQL
  (preferrably MySQL (or MariaDB))
  https://www.mysql.com/ https://mariadb.org/

- a WSGI server
  for example mod_wsgi with Apache, Waitress, Gunicorn, uWSGI...

- Authserver (if you plan to use Karp's editor)
  At Språkbanken, we use
  https://svn.spraakbanken.gu.se/repos/cjs/pub/wsauth/
  This requires **Drupal 7** together with the module Email Registration
  https://www.drupal.org/
  https://www.drupal.org/project/email_registration
  It is also possible to use a server of your choice, as long as it returns a json object containing at least:
  ```
  {'authenticated': true/false
  ,'permitted_resources' : {'lexica' : [...lexiconnames...]}}
  ```

- Python2 >= 2.7 with pip.
  https://www.python.org/downloads/
  http://pip.readthedocs.org/en/stable/installing/

2 https://www.docker.com/

3 This is important if you want more than one node in your cluster (see https://www.elastic.co/guide/en/elasticsearch/reference/1.4/_basic_concepts.html). Språkbanken currently uses three nodes running on three different servers.

4 DOWNLOADING THE CODE

Using Docker

If you want to set up a test and development instance of Karp for local use, download the code from
`git clone http://spraakbanken.gu.se/pub/karpdocker.git`
and follow the instructions in README.md. When doing offline calls (section 6-8) in this set-up, always prefix the commands with
`docker-compose run --rm karp`

Normal download

Download the code with git:
`git clone http://spraakbanken.gu.se/pub/karp.git`

In the next section, you will see what configurations you will have to do to run the system with your lexicons. There are two versions of Karp, a standard version and an extended version which targets problems specific to Språkbanken's own lexicons. The code specific for Språkbanken is located in `sb/`. If unsure, use the standard version.

The file `backend.py` is the main component, run by `backend.wsgi`. The script `upload_offline.py` will help you upload, delete and extract data from the system as an offline service, only available to the developers. It is run like a normal python script, you will learn more about this in Section 6-8.

5 CONFIGURATIONS

The first thing you have to do is to set up the virtual environment for python (this is however not needed when running Karp in Docker). Do this by running the following commands:

```
cd backend
virtualenv venv
source venv/bin/activate
pip install -r requirements.txt
```

From now on, whenever you want to run the code from a fresh terminal, you must reenter the environment: `source venv/bin/activate`
To deactivate the environment (when done working with the Karp), deactivate it:
`deactivate`
Next, there are some configurations to do in the files located in the directory called `config`. By default all the lexicon-related configuration files are set to handle the PANACEA lexicon mentioned in Section 1. This was done in order to provide a working example.

debugmode.py

Set `debug` to True if you want the server to print more detailed errors and to show a more detailed error report to the user. Since this means that information that might be sensitive will be exposed to any user, make sure to inactivate the debug mode before you release your Karp version. The same goes for the logging level in the last line:
`logging.basicConfig(stream=sys.stderr, level=logging.DEBUG)`
for an exposed version of Karp, you will probably want to set this to
`logging.basicConfig(stream=sys.stderr, level=logging.ERROR)`.
For non-Docker users: Change the file path of `debugfile` to be the absolute path of your log file and make sure that this file exists and that the user who will run the server (eg. the Apache user) has write access to this file. This is crucial, since the backend will not run if the write access is not ok.

dbconf.py

If you're using MySQL/MariaDB, all you have to do is to replace `karplabb` to the database name that you wish to use.
`mysql = 'mysql+pymysql://'+user+'/karplabb?charset=utf8'`

`admin_emails` should be a list with all email addresses to which notifications should be sent on any SQL error

`sender_email` should be the email address that the system puts as the sender of its emails. Emails are sent to administrators on errors but also to users that have submitted suggestions.

dbpassword.py

Create this file, and add the following line
`user="name:pass@server"`
where `name`, `pass` and `server` corresponds to your SQL login details.

setup.py

Necessary:
`script_path`: corresponds to `SCRIPT_NAME`; the initial portion of the request URL's "path" that corresponds to the application object. Should, in other words, show the relative path from the server's "root" to your application.
`sb_extended`: unless you want to work with the version specified on Språkbanken's lexicons, set this to False
`elasticnodes`: a list of urls to the ES nodes in your cluster
Optional:
`indexalias`: ES will store your data in an index, you can chose the name of that here.[4]
`index_internal`: should be set to the same as `indexalias` (not important, unless your using the extended version)
`_type`: this is the name of the type used in ES. It can be set to anything.
`sugg_index`: if you plan to allow suggestions (edits by non logged in users), they need to be stored in a different index than "normal" entries. Make up a name and put it here.

4 To be precise, this will be the name of an index alias, rather than index

authconfig.py
Necessary if you want to allow editing. If you're using the same system as Språkbanken, fill in the url and the secret string.

lexiconconf.py
`conf`: a dictionary where the keys are the names of the lexicons. The values are lists containing the order, and the path to the lexicons. Example:
`conf = {'panacea': [0, 'data/panacea/panacea.json']}`

fieldmappings.py:
In this file you define which fields in your data should be searchable, and map those to keywords. The keys are names which you will later be able to use in an extended query to the backend, and should hence match the field names for extended search in the frontend. For example, if a lexical entry looks like this

```
{'Form': {'baseform': '...', 'partOfSpeech': '...', 'example': '...'},
 'Sense': { 'example': '...'}}
}
```

and you wish to be able to search for baseform, part of speech and the text in any of the examples (but you do not make a distinciton between these two) you should add these lines

```
mappings = {'baseform': ['Form.baseform'],
            'pos': ['Form.partOfSpeech'],
            'example': ['Sense.example','Form.example']
           }
```

Mappings can be many-to-many; one key can be linked to many fields and many keys may refer to the same field. Keep the lines referring to the lexiconName and lexiconOrder.

While setting up the mapping (see below) you will learn more about how the fields are indexed.

searchconf.py:
This is where you define how different sorts of searches should be performed. You will specify which fields (or json paths) of your lexicons are interesting for different purposes. Before you complete this file, it might be useful to have a look at the mapping configuration (below).

`sort_by` : define the order by which your results are sorted. It is recommended to keep `lexiconOrder` in the first position, since the Karp frontend assumes this to be the first sorting criteria. Unless you want to totally ignore the score that the search engine assigns each hit, also keep `_score` in this list!

`head_sort_field`: if you wish to always have a specific primary result order, even if the user has inputted more fields to search on, you can specify that here. Example:

If the user wants to sort by part of speech, you still want the results to be sorted by lexiconName in the first place, and after that by part of speech.

`minientry_fields`: the minientry search is a search type where only the most important information for each entry is shown. Specify here which fields should be shown in a minientry search.

`statistics_buckets`: the statistics query does an aggregation[5] of the data; it groups the data by a chosen set of fields and shows the number in each "bucket". The user can input what fields he or she wants to use at query time, but the default grouping should be put here. Example: `["lexiconName", "pos"]` will, for each lexicon, show how many entries there are belonging to each part of speech.

`all_fields`: specifies which fields should be used for free text search (simple search). ES will keep a copy of all text from the fields of your choice (you will set this in the `mappingconf.json`) in one or more designated fields, usually referred to as `_all`. If you don't know yet what field names you want to put here, simply put
`all_fields = ["_all"]`
for now.

`boosts`: this is used for free text search. Apart from searching all fields, ES can boost the results which contain the query string in one or more specific fields. You might for example be more interested in results where the baseform matches the query string than where a comment matches it. Put the fields you consider to be most interesting here, in order of descending importance. For more control of how the boosting is done, you will need to modify the source code in `server/translator/parser.py`

`autocomplete_field`: Autocompletion will return all entries having fields that contain the requested word. Specify the fields you want to search. This function is called from Karp's sister project Korp.

The field names in all settings in `searchconf.py` should always be - or evaluate to - the json paths in your lexical entries. You can use the function `F.lookup` and the keys from `fieldmappings.py`:
`sort_by = [F.lookup(pos)]`
to achieve this, or simply put the whole json path:
`sort_by = ["Entry.partOfSpeech"]`.
Note that the function `F.lookup` only picks the first path in case of an one-to-many mapping. If

```
"pos" : ["Entry.partOfSpeech", "Entry.simplePartOfSpeech",
         "Entry.translatedPartOfSpeech"]
```

then
`[F.lookup(pos)] = "Entry.partOfSpeech"`

mappingconf.json:
In this mapping you specify how ES should treat your data. You can set tokenizers, searchable fields etc.[6] The default `mappingconf.json` file contains a simple mapping for the PANACEA lexicon. A more advanced example of a mapping can be found in `example_mappingconf.json`

5 https://www.elastic.co/guide/en/elasticsearch/guide/current/aggregations.html
6 https://www.elastic.co/guide/en/elasticsearch/reference/1.5/mapping.html

Easy set-up: Using the default mapping
It it possible to partly ignore this step for the moment and go on to upload your data. In that case all text in your data will be searchable and tokenized by a standard european tokenizer[7]. In that case, set the `all_fields` to `["_all"]` in `searchconf.py` and rename the file `config/simplemappingconf.json` to `config/mappingconf.json`.

Advanced set-up: Creating a custom mapping
If - or when - you want more control of your data, you can edit this file. If this is done after the data has been uploaded for the first time, however, you will need to reload it (see section 7) after finishing the new mapping.
Have a look at the file `example_mappingconf.json`. In the first section, `settings`, you can define custom tokenizers, **analyzers** and filters etc. ES provides a set of built-in analyzers, and at Språkbanken we have added some more to fit our data:

`full_name`:	using a built-in tokenizer to split at whitespaces only
`xml_analyzer`:	for xml-blocks. Makes the text, but not the mark-up, searchable.

The default analyzer in ES will split words on special characters (-,_.,"...). If that's not what you want, consider the built-in `keyword` analyzer treats the whole string as one token (useful for different types of identifiers, or possibly multiwords expression that should not be analyzed as separate tokens). The analyzer `whitespace` splits only on whitespaces. If you are interested in other alternatives or defining your own analyzers, please read the ES documentation[8].

The section below, `mappings`, is a definition of your data's structure.
You can control the **_all fields** here:
`"_all" : {"enabled" : false}` prevents ES from making all fields searchable. If you do want all text to be searchable, put this to true. Note that this will enable free text searches to match the text in `lexiconName` and `lexiconOrder`.
`"all_text", "all_xml"`: These are used instead of `_all` in Språkbanken's version. Each field in the mapping below specifies whether its content should be copied to one of those. The difference between the two is the analyzers used; text copied to `all_text` is tokenized as normal, while the text copied to `all_xml` is tokenized as xml. Simply leave out these if you enabled `_all` above. Finally, remember to update `all_fields` in `searchconf.py` to match your current settings.

The rest of the content is only depending on your data's **type structure**. If your data is simple, just write down the types of it yourself. If it is more complex, you could let ES do the job for you, by inputting all data to ES and then extracting the automatically generated mapping. To do this, run one of the commands:
without Docker:
```
python upload_offline.py --getmapping > config/newmappingconf.json
```
with Docker:
```
docker-compose run --rm karp python upload_offline.py --writemapping \
                    config/newmappingconf.json
```

7 https://www.elastic.co/guide/en/elasticsearch/reference/1.4/analysis-analyzers.html
8 https://www.elastic.co/guide/en/elasticsearch/reference/1.4/analysis-custom-analyzer.html

This will give you the file `newmappingconf.json`, which you can use as your `mappingconf.py`. Modify the settings mentioned above as needed.

What's important in the type mapping is to look at the fields **copy_to, analyzer, index and type** for each of your data fields. In the below example, we see that "blissName" is of type "string", and is copied to `all_text`, meaning that it will be searchable in free text searches. Since no analyzer is specified, ES will use its standard text analyzer. "category" is also a string, but should be analyzed with our custom analyzer. "blissID" is not indexed at all, meaning that it will not be searchable, neither in simple nor extended queries.

```
"blissName": {
    "copy_to" : "all_text",
    "type": "string"
},
"category": {
    "copy_to" : "all_text",
    "type": "string",
    "analyzer" : "full_name"
},
"blissID": {
    "index" : "no"
}
```

Note that you do not need to specify whether a field contains a list or a single object. "blissName" could contain one string, or a list of strings.

Finally, the possibility of using **multiple analyzer** on a field is worth mentioning. This is done by adding the special field called `fields`[9]. In the example mapping, you'll see that `FormRepresentations.baseform` has an extra line:

```
"type": "string", "analyzer" : "full_name",
"fields" : {"sortform" : {"type" : "string", "analyzer" : "keyword"}}
```

By doing this, we can search both `FormRepresentations.baseform` - where multiword expressions have been tokenized - and `FormRepresentations.baseform.sortform` - where the whole expression is always treated as one inseparable unit. The first option will allow us to to find "car park" by searching for "park", but will also find "car park" when we search for words starting with "pa".

Before you reload (section 7) your data, check that the mapping is valid json:

```
jsonlint config/mappingconf.json
```

6 INPUTTING THE DATA TO THE SYSTEM

Once you have installed and configured the system, you can upload your json documents to the system. The data will be stored in the search engine ElasticSearch, and

9 https://www.elastic.co/guide/en/elasticsearch/reference/2.0/multi-fields.html

backed-up in SQL. The SQL database will also keep track of the revision history of each entry, which is useful if you allow your lexicons to be edited. The below command should only be used the *first time* lexicons are added. If you use it for reloads, the previous version of the lexicons will not be properly deleted in the SQL database and hence be multiplied. This might cause problems later on. For reloads, always execute the commands in section 7.

6.1 Loading data

To upload data, the first thing you need to do is to come up with a suitable (lower cased) index name, which should **not** be the same you put for `indexalias` in `config/setup.py`. A suggestion is to put the current date in the name. If we choose the index name `karp151103`, run

```
python upload_offline.py --create_load karp151103
```

or, if using Docker:

```
docker-compose run --rm karp python upload_offline.py --create_load karp151103
```

This will upload all lexicons listed in `config/lexiconconf.py` to the databases. If your interested in the details on why we upload the data to another index than `indexalias` specified in the setup, have a look at ES's index alias functionality[10]. The karp backend is using aliases, and the alias `karp` will be set to mirror the index `karp151103` once the upload is complete. The reason we do this is to avoid downtime for your users while you are reloading the data.

6.2 Adding more lexicons

If you decide to add more lexicons to your system, first add them to the configurations, as specified above. Also make sure your mapping covers them. To do the upload, execute the below command, where `newlexcionnameN` are the names of all new lexicons, ie. the ones that have not previously been uploaded.

```
python upload_offline.py --add_lexicon newindex151115 newlexicon1 newlexicon2 ...
```

All of your lexicons will now be searchable in the new index.

7 RELOADING THE DATA

At some points, you might need to reload the data to ES. There are two types of reloads:

A Reload from SQL

B Reload from file

10 https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-aliases.html

Use the first one if you for example found errors in your mapping, and want ES to re-index and re-analyse it. This type of reload will not leave any trace in the revision history and all changes to the resources done within the system will be kept.

Use the second one if you found errors in your data that you want to fix offline and then update the databases from new files. This type of reload will delete your revision history and should hence be avoided after the initial testing phase. As when you load data the first time, come up with a new index name, for example `karp151104`. Run one of the commands:

[A]: `python upload_offline.py --publishnew newindex`

[B]: `python upload_offline.py --reload newindex`

Again, prefix the commands above with `docker-compose run --rm karp` if you are using Docker.

After you have run any of these commands, the new index will be created and made available to your users. The old data will still be stored in ES, however. It will not be searchable by your users, but could be used as a back-up in case you find errors in the new data and want to "undo" your changes. In some cases, you still want to delete the old indexes. Do that by running

`python upload_offline.py --deleteindex oldindex`

or, with Docker:

`docker-compose run --rm karp python upload_offline.py --deleteindex oldindex`

where `oldindex` is the index to be deleted. Note that `oldindex` should never be equal to the index alias in `setup.py`!

8 OUTPUTTING THE DATA FROM THE SYSTEM

Since Karp allows users to edit the resources, you might sometimes want to extract these to use for other purposes. If you want to extract the current version of a lexicon, called lexicon1, and print it to file, use

```
python upload_offline --printlatestversion lexicon1
          | python converter/mklist.py > extracted_lexicon1.json
```

If you are using Docker, prefix the command with

`docker-compose run --rm karp`

9 TESTING THE BACKEND

Once you finished the configuration and uploading, you might want to test how things are working. Below you'll find some basic examples of how to inspect the data. In these examples, we are running a test version locally. Start it by running

`python backend.py`

The web service will now run on `localhost:5000`. If you are using Docker, or if you have your WSGI server of choice running and set-up already, you should also be able to access the Karp backend through that, without running the python script manually.

The docker webservice runs on `localhost:8081/app`.
Errors will be logged to the file `debug.txt` (or to the docker logs, if you are using Docker). If you change the code, the WSGI application needs to be reloaded:
`touch backend.wsgi`
A detailed documentation on the API is available online[11].

Aggregation
To start with, you might want to do an aggregation over the data to see some statistics and make sure everything is there:
`curl 'localhost:5000/statistics'`
The result looks like this (ES provides you with details that you're probably not intersted in at the moment. If you are go to the docs[12]):

```
{
  "_shards": { "failed": 0, "successful": 10, "total": 10 }, // info from ES
  "aggregations": {
    "lexiconName": {
      "buckets": [
        {
          "doc_count": 131020, // there are 131020 entries in Saldo
          "key": "saldo",
          "pos": {
            "buckets": [
              {
                "doc_count": 85969,  // 85969 of them are nouns
                "key": "nn"
              },
              {
                "doc_count": 21839,  // 21839 of them are adjectives
                "key": "av"
              },
              ...
      ],
      "doc_count_error_upper_bound": 0, "sum_other_doc_count": 0 // info from ES
    }
  },
  "hits": {
    "hits": [],
    "max_score": 0.0,
    "total": 727846  // total number of entries in your data base
  },
  "timed_out": false,  // ES did complete the search without timing out...
  "took": 80           // in 80 milliseconds
}
```

11 https://ws.spraakbanken.gu.se/ws/karp
12 https://www.elastic.co/guide/en/elasticsearch/reference/1.4/_the_search_api.html

Simple search
If the statistics looked good, you could go on testing a free text search
`curl 'localhost:5000/query?q=simple||house'`
Check that the results match the query in the fields you added to `all_fields` in `searchconf.py` and that the index specified in each hit is that of the last upload you did.

Extended search
Now try and see that the configuration in `fieldmappings.py` works. Try searching different fields (*keys* from the fieldmappings).
`curl 'localhost:5000/query?q=extended||and|FIELD|equals|house'`

Random search
The random search will, as the name suggests, let you see random entries. The information will be displayed the same way as for minientries. It might be useful to test this function a few times to see that the entries show up and contain the information you expected.
`curl 'localhost:5000/random'`

10 CHECKING THAT ELASTICSEARCH IS HEALTHY

ElasticSearch provides many ways of managing your cluster and checking its status and health, and it is recommended to read up on those[13].

Here are a few calls that might be good to know about. We assume that you have a local node running on port 9200.
Check that your cluster is doing ok:
`localhost:9200/_cluster/health?pretty`
The status should be green, unless you are running a cluster with 1 node only. In that case the status should be yellow.
See the indices that you have created:
`localhost:9200/_cat/indices?v`

11 FRESH RESTART

If you want to start over and remove all lexicons you've added so far, run the command `python upload_offline.py --delete_all`
This will delete *all* ES indices, and remove all lexicons in your configuration file `config/lexiconconf.py` from the SQL database.

13 https://www.elastic.co/guide/en/elasticsearch/reference/1.5/_cluster_health.html