Pangeanic Corporate Language Solution

On premises and SaaS models

General Description V.3.0, October 2021

Contents

1. Aim of the Document

The purpose of this document is to describe the functionalities, requirements, architecture and options of the Pangeanic Corporate Language Solution, one of the products of the Pangeanic Natural Language Processing technology suite.

At the time of last document revision (Apr 2019), the described product is the cornerstone of Pangeanic SaaS suite. This is in production on a mixed environment (on-premises and in AWS cloud) to offer language processing both to internal users (Pangeanic Translation Services) and external customers all over the world.

The document starts with an overall description of functionality and architecture (sections 3 and 4).

A detailed description of features follows (sections 5 to 12).

The last chapters deal with Deployment Scenarios (section 13), Maintenance, a section about Branding and white-labelling and Pricing schemas (section 16).

2. References

This document references other documents. Full details can be found in:

- PGBClient user guide
- Document Translation Service Operating Instructions
- Relay Database specifications
- Pangeanic General Architecture Overview
- Pangeanic Code Guidelines for API Services

3. Product Functionalities

The full product allows customers to replicate Pangeanic setup in order to provide the same wide range of services for internal use or to create a SaaS offering from internal or on cloud servers.

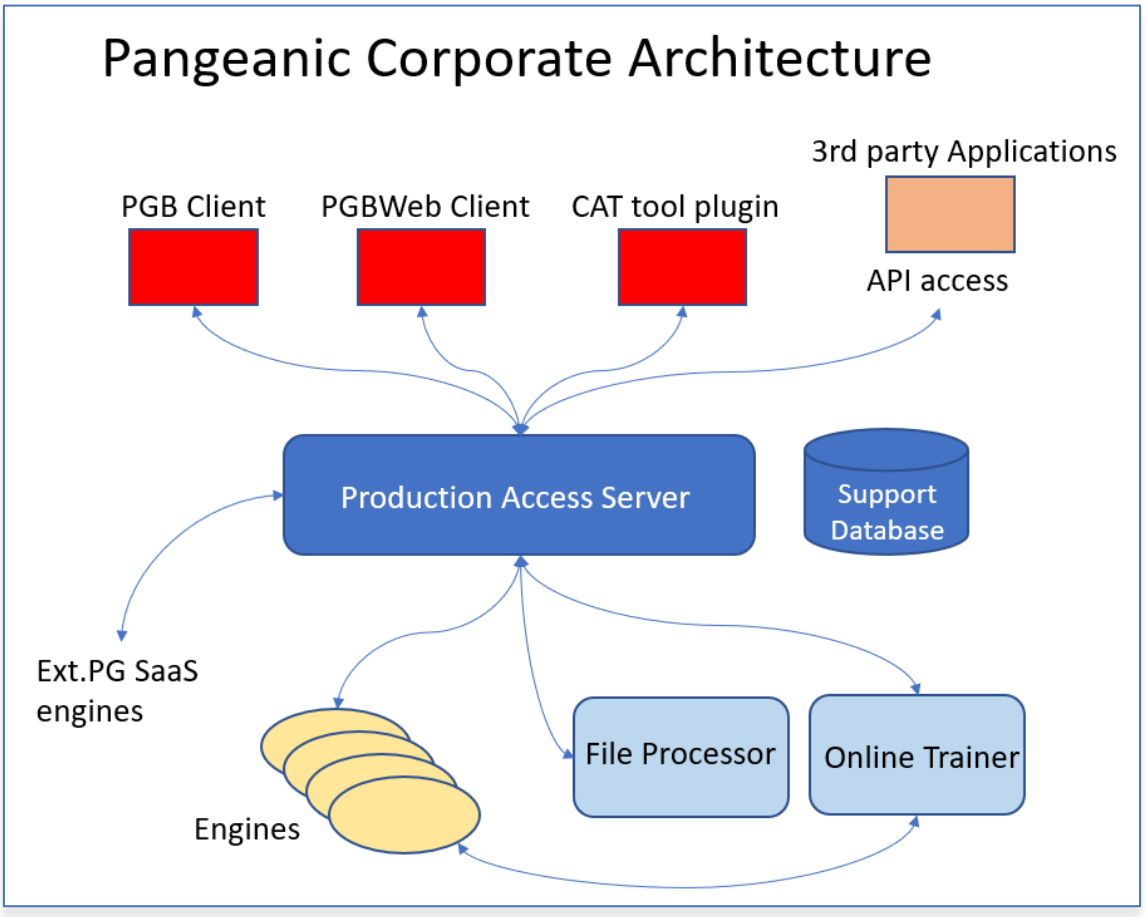
Some product components are also offered separately or as an option to better accommodate the needs of an organization.

3.1. SaaS offering

All the functionalities described for the full product in the document with a few exceptions (marked in that case during the description) are offered by Pangeanic with a SaaS model.

4. Architecture

The diagram shows the different logical blocks.



The solution is implemented on a fully distributed architecture with different modules:

- Users access the functionalities with a variety of client interfaces (described later) such as the PGB, Web applications, CAT tools or programmatically

with a RESTFul API for integrations.

- The Production Access Server manages user requests and orchestrates the rest of modules. It requires a standard SQL database to store the required data to fulfil the requests.
- The engines, either local (managed by the organization on their own premises or on their own cloud) or operated by Pangeanic with a SaaS model will perform the actual language processing.
- A file processor is in charge of dealing with converting files and documents when this feature is installed.
- An on-line trainer module is in charge of evolving the models according to the user preferences. This is integrated in the engine package when the on-line learning option is installed.

5. The engines

The engines are applications consisting typically of a neural network, NLP processes and a small access server.

They fulfil the tasks of processing language assets in the form of individual sentences, paragraphs or text chunks. The output format depends of the processing type. In the case of standard translation engines, a sentence is the input, and a translated sentence is the output.

The NLP processes included look after the pre-process and post-process of the input and output and couple the neural network to the rest of the engine.

The engine small access server implements a basic REST API that will be used by the corporate solution but that makes possible the deploying of single engines with no need of extra modules in simple deployment scenarios.

The process types provided by Pangeanic engines are (not a complete list):

- Generic translation
- Customized translation
- Language detection
- Text categorization
- Anonymization
- Summarization
- Relevance and sentiment analysis

When a NN-based (neural network) engine is installed together with the Online Learning feature, the engine can be re-trained with user-corrected data to improve the engine's accuracy according to client preferences.

Refer to the Online Learning section in the document to know more about the features implemented.

Pangeanic engines are delivered in a docker package and in two flavours, one to deploy and run the engine using one or more CPU cores and another one to deploy and run the engine using GPUs for improved performance.

Several engines can share the same host system, but GPU engines require dedicated access to one of the GPUs in the host system.

5.1. Translation Engines

Pangeanic offers different types of Translation Engines. For all most-spoken language combinations (pairs) there exists a Generic Translation Engine (GTE). While GTEs are good enough for most applications they don't offer near-human translation accuracy because they need to cope with multiple domains jargon, styles, dialog styles, legal texts, technical documents, unpredictable writing, etc.

Unlike GTEs, Custom Translation Engines (CTEs) are trained to translate text in a specific domain (legal, medical, finance…) and they typically contain data generated by the client, whose terminology and style are prioritized. CTEs are trained for customers requesting them, with homogeneous in-domain corpora provided partially or totally by the client.

5.2. Anonymization Engines

Anonymization Engines are similar to translation engines, the output can be considered text that is written in the same language of the source but where personal data is replaced by identifiers.

But in many user cases the anonymization engines are run on-premises whereas the rest of language services are accessed as SaaS. Running the anonymization on-premises guarantees that texts sent to a 3rd party (SaaS provider) will be clean of private/personal data in order to comply with GPRD-like regulations.

6. Production Access Server

The Production Access Server (PAS) manages user requests and orchestrates the rest of modules.

It is implemented using Java Tomcat Application Server and requires a standard MySQL database to store the required data to fulfil the requests. The implementation is a fully scalable, production-ready setup capable of serving hundreds of requests per second.

The functionalities covered by the PAS are to:

- Manage the configuration of the whole solution: define the engines and optional modules available, their state and location.
- Implement the interface (API) between client applications and engines and modules.
- Manage the ecosystem communicating the different engines and modules; for instance to chain engines when the user requested process implies that several engines will concatenate.
- Access Control with usernames, password and APIKeys
- Grants management rights, who can access what.
- Store statistics about usage and performance.
- Access external engines (Pangeanic SaaS) to extend the customer installed engines.

Running a PAS is mandatory when several engines or optional modules are

installed, it is also mandatory when one of the provided client applications is to be used (PGB, PGWeb, CAT tools or the generic API).

When running an on-premises setup the PAS is the only component that is exposed to the rest of the network but is also the most-centered element, with direct access to the rest of components of the system.

7. API Access

External programs, legacy systems, programmers using console or API clients can access the full solution functionalities using the generic APIs.

The updated description of APIs can be found at the following repositories:

- https://corporateapi2.docs.apiary.io/# (text processing),
- https://pgfile.docs.apiary.io/# (document processing) and
- http://prodb.pangeamt.com:8000/docs (anonymization)
- Pecat API (no online repo)

Pangeanic's API is a simple RESTFul implementation where typically the requests are sent as POSTs with a json encoded body.

When handling files the content is sent or received in base-64 encoded form.

As an example, the API definition for the translation of a single sentence would be:

POST to endpoint: `https://production_access_server_url:8080/NexRelay/v1/translate`

Headers: `Content-Type : application/json`

Request Body:

```
{
"src":"en",
"tgt":"es",
"apikey":"your_api_key",
"engineid":"your_engine_id",
"text":[
  "This is an example."
```

```
  ]
}
```

Answer Body:

```
[
    [
        {
            "src": "This is an example.",
            "tgt": "Esto es un ejemplo."
        }
    ]
```

Both https and http endpoints are configurable in the Access Server.

7.1. Text Processing API

HOST: http://prod.pangeamt.com:8080/NexRelay/v1/

List Engines [/corp/engines]

List All available engines [POST]

This edpoint is required to list the list of IDs of the available engines fo your APIKey

+ Request (application/json)

```
    {
            "apikey": "your-apikey-here"
        }
```

+ Response 200 (application/json)

```
    {
```

```
"engines": [
    {
        "id": 1,
        "processid": 1,
        "serviceid": 1,
        "inserviceid": 0,
        "src": "en",
        "tgt": "es",
        "descr": "ENES_B_plain",
        "domain": "",
        "flavor": "",
        "status": 0
    },
    {
        "id": 2,
        "processid": 1,
        "serviceid": 2,
        "inserviceid": 0,
        "src": "es",
        "tgt": "en",
        "descr": "ESEN_generic",
        "domain": "",
        "flavor": "",
        "status": 0
```

```
            }
        ]
    }
```

Process text [/translate]

Process a text segment [POST]

A text segment is a number of words, processing works best if the segment is a sentence with whole semantic content.

- src and tgt are 2-letter language codes

- engine is the id (string) of the engine to be used

- glossary_id, optional, is the id (integer) of the glossary to be used

+ Request (application/json)

```
{
        "src":"es",
        "tgt":"en",
        "apikey":"your-api-key-here",
        "engine":"2",
        "glossary_id": 1,
        "text":[
            "¿Cómo estás?",
```

```
            "Mi perro es negro"
        ]
    }
```

+ Response 200 (application/json)

```
[
    [
        {
            "src": "¿Cómo estás?",
            "tgt": "How are you?"
        }
    ],
    [
        {
            "src": "Mi perro es negro",
            "tgt": "My dog is black"
        }
    ]
]
```

7.2. Document Processing API

HOST: http://prod.pangeamt.com:8080/PGFile/v1

File Uploading [/sendfile]

Send a File [POST]

Main request. Send a file to the service for translation.

The process parameters have to be included

The success response is a json with file guid.

+ Request

(multipart/form-data;boundary=----WebKitFormBoundary8M3sSU13ul5lXSJm)

```
------WebKitFormBoundary8M3sSU13ul5lXSJm
    Content-Disposition: form-data; name="json"


    {
        "title":"filename.docx",
        "processname":"translation",
        "engine":"123",
        "src":"es",
        "tgt": "en" ,
        "apikey":"your-apikey-here",
        "username": "testuser",
        "notiflink": "testlink",
```

```
            "processoption": "1"

        }

    ------WebKitFormBoundary8M3sSU13ul5lXSJm

        Content-Disposition: form-data; name="file";
filename="filename.docx"

        Content-Type: application/vnd.openxmlformats-
officedocument.wordprocessingml.document

        data

    ------WebKitFormBoundary8M3sSU13ul5lXSJm--
```

+ Response 200 (application/json)

```
    {
        "fileId": "8d4e1c5be60d4e04850f55ec135f2554"
        }
```

+ Response 500 (application/json)

```
    {
        "error": true,
        "error_message" : "the-error-message"
        }
```

Checking file status [/checkfile?apikey&fileid]

Use /checkfile to get the processing status of a file, this is a json post where body

can have three forms:

- just the apikey, the response will return info about all files being processed for the apikey.

- guid, to get info about a single file

Use a get request like:

https://server_address:server_port/PGFile/v1/checkfile?apikey=your_api_key

https://server_address:server_port/PGFile/v1/checkfile?apikey=your_api_key&guid=d9b95ffb3bc347efa6a7cb90ca80ca74

The returned answer is a json list containing one json object for every file.

The important field returned for every file is the status, an integer with possible values:

- 10, file is queued for process at the server

- 6 or 7, file is being analyzed

- 20, 30 or 40, file is being processed

- 100, file processed successsfully, waiting download

- 110 or 120, file downloaded

- -10 or -20 (negative values), processing error

Check the processing status of a file [GET]

+ Response 200 (application/json)


```
[
        {
            "fileId": "1dc77dc5ba6d44828b860537dae07187",
            "translatedPath": None,
            "engineId": 58,
            "glossaryId": 0,
            "src": "es",
            "tgt": "en",
            "isZip": False,
            "ztot": 0,
            "zfinished": 0,
            "translatedName": None,
            "processName": "translate",
            "processOptionId": 1,
            "link": "",
            "status": 10,
            "id": 16567,
            "fileName": "test.txt"
        }
    ]
```

Checking and downloading file [/retrievefile]

This is a POST request that can be used to get info about a file that is being processed and

download the processed file is process has finished.

WARNING this endpoint is NOT safe for very big files: the file is returned in base64

encoding and requires a lot of memory at server and client side.

When file is finished a 200 response is returned with the file.

If file is in processs a 200 response is returned with the status data.

Errors will generate a 401 response.

Check and get File [POST]

+ Request (application/json)

```
{
        "apikey": "your-api-key-here",
        "guid": "1dc77dc5ba6d44828b860537dae07187"
    }
```

+ Response 200 (application/json)

    # response when file is not finished

```
{
```

```
            "success": True,
            "error": {},
            "status": "10",
            "data": {}
        }
```


response when file is finished


```
    {
            "success": True,
            "error": {},
            "status": "110",
            "data": {
                "guid": "d74b06fd52434ad2bfbe82ebbff96464",
                "fileType": "txt",
                "filename": "test.txt",
                "file":
"U2VlIHRoaW5ncyB0byBkbywgcmVzdGF1cmFudHMsIGFuZCBob3RlbHMNCg=="
            }
        }
```


+ Response 401 (application/json)


```
      {
            "success": False,
            "error": {
                "statusCode": 401,
```

```
            "code": 6,
            "message": "Invalid GUID"
        },
        "data": {}
    }
```

Downloading file [/download?apikey&fileid]

Simple get request to retrieve a file. The get parameters are:

- apikey

- fileid, the file GUID

The file is returned as a stream (similar to a call to wget)

Send a File [GET]

+ Response 200

7.3. Anonymization API

Anonymize a file [/anonymize/xliff]

Anonymize a file [POST]

Main request. Send an XLIFF file to the service for anonimization.

Anonymize an XLIFF(XML Localization Interchange File Format) file. The request body must be Content-Type multipart/form-data

noqa Args:

anon_dict: TAGs, entities pairs that *should* be anonymized.

clear_list: entities that *should not* be anonymized

model: pre-trained model to use, default ner-fast

lang: language of the XLIFF file, default "en" *Please refer to default.json file for more info*

noqa File: An XML Localization Interchange File Format compliant file, refs: OASIS Standard. Returns: An anonymized XLIFF file, as a binary response.

Anonymize a sentence [/anonymize/sentence]

Anonymize a sentence [POST]

Anonymize a sentence or string compliance input. Args:

- **anon_dict**: TAGs, entities pairs that *should* be anonymized.
- **clear_list**: entities that *should not* be anonymized
- **model**: pre-trained model to use, default ner-fast
- **lang**: language of the XLIFF file, default en_US.UTF-8 *Please refer to default.json file for more info* # noqa Returns: A dictionary with the following keys:
- **status**: true or false,
- **message**: the NN response of the anonimization process.
- **anon_sentence**: If status is true, contains the anonymized sentence with the entities recognized between brackets. # noqa

Example body payload:

```
{
  "model_lang": "en",
  "anon_dict": [
    {
      "entity": "string",
```

```
      "name": "string",
      "path": "string"
    }
  ],
  "clear_list": [
    "string"
  ],
  "anon_list_tagset": [],
  "sensibility": 0,
  "mask": "TAG:idx",
  "profile_id": 1,
  "sentence": "string"
}
```

Responses

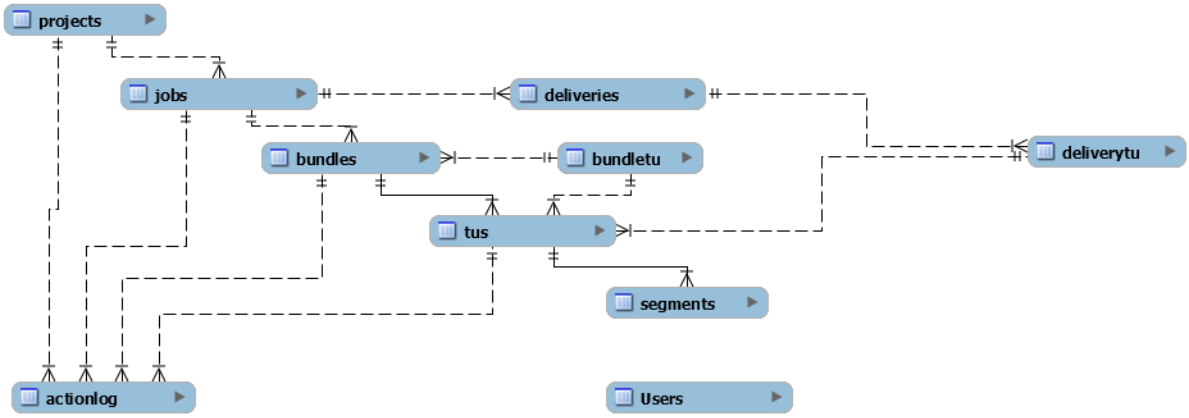
Code: 200, returns anonymized sentence

Code 500, validation error, returns:

```
{
  "detail": [
    {
      "loc": [
        "string"
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

7.4. Pecat API

Database Schema



Constants used when using the API

Projects

Status of a project

PROJECT_CREATED = 0 //the project was created successfully

PROJECT_PREPARING = 2 // preparation has started, bundling and assigning is possible

PROJECT_PREPARED = 3 // project fully prepared, translations can start

working

PROJECT_WIP = 4 // project progressing

PROJECT_DONE = 5 // project finished, no more translation, revision or eval required

PROJECT_CLOSED = 6; // no further actions allowed, except OPEN

PROJECT_ARCHIVED = 100 // not visible in normal lists

Jobs

job types

0: translation

1: revision

2: evaluation

translation types

0: simple

1: double, with alternative

Preparation status values

JOB_PREPSTATUS_CREATED = 0 // job defined, TUs loading in tables

JOB_PREPSTATUS_LOADED = 1 //TUs loaded, job can be prepared, bundling possible

JOB_PREPSTATUS_PREPAREREQUESTED = 2 //PM has requested preparation

JOB_PREPSTATUS_PREPARING = 3 // job being prepared

JOB_PREPSTATUS_PREPARED = 4 // job fully prepared

Status values , for every job two status are always kept for worse & best status of the included TUs/Bundles

JOB_STATUS_ERROR = -1 // Job failed loading or preparing

JOB_STATUS_CREATED = 0 // Job successfully created

JOB_STATUS_LOADED = 1 // Job TUs loaded

JOB_STATUS_PREPARING = 2 // Job being prepared

JOB_STATUS_UNBUNDLED = 3 // Job requiring bundling, some TUs not bundled

JOB_STATUS_BUNDLED = 4 // TUs bundled

JOB_STATUS_ASSIGNED = 5 // bundles assigned

JOB_STATUS_TRANSLATING = 6 // bundles translating

JOB_STATUS_EVALUATING = 7 //TUs being evaluated

JOB_STATUS_TRANSLATED = 8 // All TUs translated

JOB_STATUS_DELIVERING = 9 // Deliveries done

JOB_STATUS_DONE = 10 // Job finished

Bundles

Bundle type

0: translation

1: revision

2: evaluation

Bundle extraction mode

0: consecutive

1: hashed

Bundle Status values

BUNDLE_STATUS_ERROR = -1

BUNDLE_STATUS_CREATED = 0

BUNDLE_STATUS_PREPARING = 1

BUNDLE_STATUS_PREPARED = 2

BUNDLE_STATUS_ASSIGNED = 3

BUNDLE_STATUS_WIP = 4

BUNDLE_STATUS_TRANSLATING = 5 // Some TUs require translation, all evaluation done

BUNDLE_STATUS_TRANSLATINGANDEVALUATING = 6

BUNDLE_STATUS_EVALUATING = 7 // No more translation required

BUNDLE_STATUS_DONE = 8

BUNDLE_STATUS_FREED = 9 //can be delivered if not empty!

BUNDLE_STATUS_DELIVERED = 10

Qamode how translation and QA are linked

0: default, synch qa // the translator receives QA check synchronously after validation of a translation

1: asynch qa // QA triggers but result is sent to translator asynchronously

2: no QA // No QA is done after translation validation

Qemode, how evaluation for the bundle is defined

0: default, 10% early and last weight

1: 10% evenly distributed

2: 5% evenly distributed

3: No QE

Freemode, how a bundle is unassigned when PM requests it

0: free all tus, reset translations, QA and evaluations

1: free not QAed tus, keep translations, QA and QE

ACTIVITY bundle status,

0: Normal, translations and evaluation are accepted

1: Paused, PM has decided not more work can be done for the time being

TUs constants

Translation code, reasons for a source not to be translated

0:Translatable, no reason to skip translation

1: Nonsense

2: Encoding

3: Source not in the right language

4: Other

5:Not-alternative possible (used in alt translations)

Translation status, applied to main and alternative translations

0:Translation required

1: Translation done

Qastatus, how QA is for a TU

0: Not doable yet

1: QA required

2: QA to validate

3:QA validated, false positive

4: QA ok

5: QA ko

Qestatus, how QE is for a TU. When bundling some TUs are randomly selected for QE

0: Not candidate for QE

1: QE candidate, pending evaluation

2: QE done, positive

3: QE done, negative

TU Status

TU_STATUS_CREATED = 0 //no translation has been done

TU_STATUS_PREPARED = 1 //TU prepared

TU_STATUS_TRANSLATED = 2 //translation finished

TU_STATUS_QAED = 3 //QA finished

TU_STATUS_QED = 4 //QA evaluated

TU_STATUS_DONE = 5 // No more work required

TU_STATUS_REWORK = 6 // like CREATED, after some work has been done

TU_STATUS_FREED = 7 // like CREATED, after translator has been unassigned

REVISION STATUS

TU_REVSTATUS_NOTDOABLE = 0; //initial status

TU_REVSTATUS_REQUIRED = 1; //un-used

TU_REVSTATUS_OK = 2; //revision done, translation accepted

TU_REVSTATUS_UPDATED = 3; //revision done, translation was changed

TU_REVSTATUS_KO = 4; //translation not accepted, re-do

TU_REVSTATUS_PENDING = 5; //revision bookmarked by reviewer for later

TU_REVSTATUS_SKIPPED = 6; //the revision will not be done

QE constants

Eval Types

0: Accuracy

1:Fluency

2: Terminology

3: Style

4: Locale

5: Instructions

6: Other

Eval SubTypes

0: Under-translation

1: Untranslated text

… …

Eval Criticity

0: Preferential

1: Minor

2: Major

3: Critical

Bundle-TU relation

Status

0: Normal

1: Old // TU belonged to this bundle but was unassigned

Deliveries

Delivery Type

0: Partial

1: Final

Delivery Mode

0: standard (all QAed TUs of bundles whose QEScore > 90)

1: fine (Only TUs which idinbundle< smallest badQE)

2: unconditional (all QAEd TUs)

5: Emergency (All TUs, use pretrans if not translated)

Status values

-1: error

0: requested

1: preparing

2: Done

Endpoints

Production URL: http://prod.pangeamt.com:8080/PGWF/v1

Requests are in all cases application/json and the Content-Type header should be sent. Except the /addjob request which is form-data to be able to send large files.

Responses are always json format.

/projects

Get the list of all projects with data required to show lists. An array is returned containing the project data and the current kpi/counters values.

GET

Body: none

Response:


```
[
    {
        "id": 1,
        "name": "Project1",
        "descr": "P1 description",
        "customerid": 2,
        "pmid": 1,
        "status": 1,
        "statusmsg": "Loaded",
        "kpis": {
            "tus": 101576,
            "preparedtus": 0,
            "bundledtus": 0,
            "assignedtus": 0,
            "translatedtus": 0,
            "untranslatedtus": 0,
            "unalttranslatedtus": 0,
            "QAedTus": 0,
            "QAposTus": 0,
            "QAnegTus": 0,
            "QAFPTus": 0,
            "QEplannedTus": 0,
            "QEposTus": 0,
            "QEnegTus": 0,
            "QEScore": 0,
```

```
            "progress": 0,
            "jobs": 11
        }

    },
    {
        "id": 2,
        "name": "Project4",
        "descr": "P4 description",
        "customerid": 4,
        "pmid": 5,
        "status": 0,
        "statusmsg": "Created",
        "kpis": {
            "tus": 101576,
            "preparedtus": 0,
            "bundledtus": 0,
            "assignedtus": 0,
            "translatedtus": 0,
            "untranslatedtus": 0,
            "unalttranslatedtus": 0,
            "QAedTus": 0,
            "QAposTus": 0,
            "QAnegTus": 0,
            "QAFPTus": 0,
            "QEplannedTus": 0,
            "QEposTus": 0,
            "QEnegTus": 0,
            "QEScore": 0,
            "progress": 0,
            "jobs": 11
        }
    }
]
```


/addproject

Create a new project. Name and customerid are mandatory, PMid is optional, can be set later. The response contains the id of new project, and 0 if the process failed.

POST

Body:


```
{
    "name":"PxProject",
    "descr":"Description of the project",
    "customerid":1,
```

```
    "pmid":2 //optional

}
```

Response:

```
{
    "projectid": 3
}
```

/archiveproject

Moves all project assets to archive (project, jobs, bundle, tus and relations). Archived projects can only be deleted or un-archived.

POST

Body:

```
{
    "id": 4,
}
```

Response:

```
{
    "rc": 0 //0 if no error!
}
```

/delproject

Deletes a project, its jobs, tus and bundles. This process cannot be undone, all references to the project, jobs, bundles and tus are deleted in the database and cannot be recovered.

POST

Body:

```
{
    "id": 4,
}
```

Response:

```
{
    "rc": 0 //0 if no error!
}
```

/editproject

Change existing project values. The body can contain any combination of new values and only the id to identify the project is required. Change of status might interfere with the internal dynamic of status propagation.

POST

Body:

```
{
    "id": 4,
    "name":"PxProject4", //opt
    "descr":"PPDesc", //opt
    "pmid": 4, //opt
    "cutomerid": 4, //opt
    "status": 1 //opt
}
```

Response:

```
{
    "projectid": 4
}
```

/projectlog

Get the log contents for the project. An array is returned containing the relevant events affecting the project.

POST

Body:

```
{
    "id": 4 //the project id
}
```

Response:


```
[
      {
      "id": 32,
      "ts": 1596898208000,
      "date": "2020-08-08 14:50:08.0",
      "actorid": 0,
      "actionid": 2,
      "objectid": 12,
      "rc": 0,
      "descr": "Project created"
   },
      {
      "id": 47,
      "ts": 1596898335000,
      "date": "2020-08-08 14:52:15.0",
      "actorid": 0,
      "actionid": 2,
      "objectid": 12,
      "rc": 0,
      "descr": "Project status changed to 1"
   }
```


/prepareproject

Request preparation of existing project, triggers preparation of all jobs. The response is received immediately, and preparation is executed asynchronously generating events for the project and its jobs.

POST

Body:


```
{
    "id": 4,
}
```


Response:


```
{
```

```
    "rc": 0 //0 if no error!
}
```

/jobs

Get the list of all jobs with data required to show lists. The projectid in the request as a get query parameter) is optional and its purpose is to filter only jobs belonging to the project. The response is a json array, each item containing the full data for a job, counters and current kpis.

GET

Parameters: (optional)

projectid, to restrict the response to jobs of a project and

jobid, to get data of a single job

Response:

```
[
    {
        "id": 1,
        "projectid": 1,
        "name": "report_tgt_equal_tgt2.txt",
        "src": "it-IT",
        "tgt": "uk",
        "jobtype": 0,
        "transtype": 0,
        "QEbundling": 0,
        "QEbundlesize": 100,
        "wstatus": 0,
         "wstatusmsg": "Created",
        "bstatus": 0,
       "bstatusmsg": "Created",
        "startts": "1970-01-15 00:00:00.0",
        "duets": "1970-01-15 00:00:00.0",
        "kpis": {
            "tus": 40000,
            "preparedtus": 0,
            "translatedtus": 4,
            "untranslatedtus": 0,
            "unalttranslatedtus": 0,
            "QAedTus": 0,
            "QAposTus": 0,
            "QAnegTus": 0,
            "QAFPTus": 0,
```

```
            "progress": 0.004999999888241291
        }

    },
    {
        "id": 2,
        "projectid": 1,
        "name": "report_tgt_equal_tgt2.txt",
        "src": "es",
        "tgt": "en",
        "jobtype": 0,
        "transtype": 0,
        "QEbundling": 0,
        "QEbundlesize": 100,
        "wstatus": 0,
        "bstatus": 0,
        "startts": "1970-01-15 00:00:00.0",
        "duets": "1970-01-15 00:00:00.0",
        "kpis": {
            "tus": 10000,
            "preparedtus": 0,
            "bundledtus": 10000,
            "assignedtus": 0,
            "translatedtus": 1,
            "untranslatedtus": 0,
            "unalttranslatedtus": 0,
            "QAedTus": 0,
            "QAposTus": 0,
            "QAnegTus": 0,
            "QAFPTus": 0,
            "QEplannedTus": 0,
            "QEposTus": 0,
            "QEnegTus": 0,
            "QEScore": 0,
            "progress": 0,
            "bundles": 2
        }

    }
]
```


/addjob

Create a new job for a project, uploading the TSV file and defining job-level parameters. The response is a json object with informations about the creation and the counters of the file.

Note: Adding a job synchronously uploads and counts lines, words and characters but the creation of TUs in the database is triggered and will run asynchronously.

POST form-data

Body, key values:

file, the file to upload with the TUs of the job

src, language code

tgt, language code

projectid: the id for the project this job belongs to

jobtype: 0 translation (default), 1 revision, 2 evaluation

transtype: 0 normal (default), 1 double-translation

QEbundling: 0 normal (default)

QEbundleSize: a number (default)

Response:

```
{
    "message": "Job file uploaded to : /home/ubuntu/jobs/chunk143_agriculture
machinery_tourism_hotel.txt",
    "jobid": 7,
    "lines": 5394,
    "words": 61114,
    "chars": 445714
}
```

/joblog

Get the log contents for the job. An array is returned containing the relevant events affecting the project.

POST

Body:

```
{
    "id": 4 //the job id
}

Response:
[
      {
```

```
    "id": 32,
    "ts": 1596898208000,
    "date": "2020-08-08 14:50:08.0",
    "actorid": 0,
    "actionid": 2,
    "objectid": 12,
    "rc": 0,
    "descr": "Job best status changed to 6"
},
    {
    "id": 47,
    "ts": 1596898335000,
    "date": "2020-08-08 14:52:15.0",
    "actorid": 0,
    "actionid": 2,
    "objectid": 12,
    "rc": 0,
    "descr": "Job best status changed to 6"
}
```


/editjob

Change existing job parameters. The id is required and the only interesting values to change are the start and duedate (in long millisecs values). Changing status is usually pointless because their values will be internally recalculated.

POST

Body:

```
{
    "id": 4,
    "src":"it-IT", //opt
    "tgt":"en-US", //opt
    "wstatus": 3, //opt
    "bstatus": 4 //opt
    "startts": 1234567890 //opt
    "duets": 1234567890 //opt
}
```

Response:

```
{
    "jobid": 4
}
```

/deliverjob

Generates a TSV, with all translated TUs. Three modes are possible:

- 0, normal, the delivery TSV contains the translated TUs of bundles whose QEScore are > than a threshold entered by the PM (90 as default)
- 1, unconditional, all translated TUs are delivered
- 2, emergency, all TUs are delivered, untranslated TUs are delivered with the pre-translation.

The response informs of how many TUs were in the delivery, what was the final QEScore and two links, one for the delivery TSV and another one with the QA report.

POST

Body:

```
{
    "id": 4,
    "mode": 0,
    "qescoremin": 85  //only used for mode 0
}
```

Response:

```
{
    "rc": 0 //0 if no error!
    "tsvlink": "http://pgwf.pangeamt.com/deliveries/jdshakhdwewqlkdsamn.tsv",
    "qareportlink": "http://pgwf.pangeamt.com/deliveries/jdshakhdwewqln.tsv"
    "tus": 1234,
    "qescore": 90,

}
```

/getjobdeliveries

Get the list of all deliveries done for a job.

GET

Body: (optional)

```
{
```

```
    "jobid": 1
}
```

Response:

```
[
    {
        "id": 1,
        "deliverydate": "1970-01-15 00:00:00.0",
        "tsvfilename": "xxxxx.tsv",
        "qareportfilenamename": "zzzzzz.txt",
        "tus": 12345,
        "mode": 0,
        "kpis": {
            "tupct": 80,
            "qescore": 90,
            "qaissues": 12,
            "untranslatedtus": 5,
            "unalttranslatedtus": 7
       },
    {
        "id": 2,
        "deliverydate": "1970-01-15 00:00:00.0",
        "tsvfilename": "xxxxx.tsv",
        "qareportfilenamename": "zzzzzz.txt",
        "tus": 12345,
        "mode": 0,
        "kpis": {
            "tupct": 80,
            "qescore": 90,
            "qaissues": 12,
            "untranslatedtus": 5,
            "unalttranslatedtus": 7
       }
    }
```

/deljob

Deletes job, its tus and bundles. This action cannot be undone and all the job data is deleted from the database.

POST

Body:

```
{
```

```
    "id": 4,
}
```

Response:

```
{
    "rc": 0 //0 if no error!
}
```

/preparejob

Request preparation of existing job. Same functionality and process than /prepareproject but at joblevel.

POST

Body:

```
{
    "id": 4,
}
```

Response:

```
{
    "rc": 0 //0 if no error!
}
```

/bundles

Get the list of all bundles with data required to show lists.

GET

Query parameters (optional and exclusive):

- jobid, to restrict the answer to the bundles of a job
- actorid, to restrict the answer to the bundles assigned to an actor as translator

- bundleid, to restrict the answer to a single bundle

Response:


```
[{
   "id": 141,
   "jobid": 64,
   "start": 0,
   "end": 0,
   "bundletype": 0,
   "qamode": 0,
   "qemode": 3,
   "status": 5,
   "statusmsg": "translating",
   "actorid": 3,
   "reviewerid": 4,
   "evaluatorid": 0,
   "actstatus": 0,
   "actstatusmsg": "Normal",
   "startts": "2020-09-17 00:00:00.0",
   "duets": "2020-09-17 00:00:00.0",
   "jobtype": 0,
   "translationtype": 0,
   "src": "en",
   "tgt": "de",
   "allowhistrev": 0,
   "kpis":    {
      "tus": 100,
      "words": 747,
      "chars": 3843,
      "preparedtus": 100,
      "bundledtus": 100,
      "assignedtus": 100,
      "translatedtus": 55,
      "untranslatedtus": 9,
      "unalttranslatedtus": 0,
      "QAedTus": 53,
      "QAposTus": 26,
      "QAnegTus": 0,
      "QAFPTus": 27,
      "reviewedtus": 100,
      "reviewedOKtus": 11,
      "reviewedKOtus": 1,
      "reviewedchangedtus": 0,
      "QEplannedTus": 0,
      "QEposTus": 0,
      "QEnegTus": 0,
      "QEScore": 0,
```

```
      "QAScore": 0,
      "revScore": 1,
      "GdAvg": 55,
      "progress": "55.0",
      "preparationprogress": "100.0",
      "translationprogress": "55.0",
      "revisionprogress": "100.0",
      "evaluationprogress": "0.0"
   }
},
    {
        "id": 2,
        "jobid": 4,
        "start": 101,
        "end": 200,
        "bundletype": 0,
        "qamode": 0,
        "qemode": 0,
        "status": 0,
         "statusmsg": “Created”,
        "autorId": 0,
        "evaluatorId": 0,
        "startts": "1970-01-15 00:00:00.0",
        "duets": "1970-01-15 00:00:00.0",
        "jobtype": 0,
        "translationtype": 0,
        "src": “en”,
        “tgt”:”es”,
        "translationtype": 0,
        "kpis": {
                "tus": 0,
                "words": 0,
                "chars": 0,
                "preparedtus": 0,
                "bundledtus": 0,
                "assignedtus": 0,
                "translatedtus": 0,
                "untranslatedtus": 0,
                "unalttranslatedtus": 0,
                "QAedTus": 0,
                "QAposTus": 0,
                "QAnegTus": 0,
                "QAFPTus": 0,
                "QEplannedTus": 0,
                "QEposTus": 0,
                "QEnegTus": 0,
                "QEScore": 0,
                "progress": 0
        }
```

```
    },
    {
        "id": 3,
        "jobid": 4,
        "start": 201,
        "end": 300,
        "bundletype": 0,
        "qamode": 0,
        "qemode": 0,
        "status": 0,
         "statusmsg": "Created",
        "autorId": 0,
        "evaluatorId": 0,
        "startts": "1970-01-15 00:00:00.0",
        "duets": "1970-01-15 00:00:00.0",
        "kpis": {
            "tus": 3,
            "preparedtus": 0,
            "bundledtus": 3,
            "assignedtus": 0,
            "translatedtus": 1,
            "untranslatedtus": 0,
            "unalttranslatedtus": 0,
            "QAedTus": 0,
            "QAposTus": 0,
            "QAnegTus": 0,
            "QAFPTus": 0,
            "QEplannedTus": 0,
            "QEposTus": 0,
            "QEnegTus": 0,
            "QEScore": 0,
            "progress": 16.66666603088379
        }
    }
]
```


/addbundle

Create a new bundle for a job.

Note: If parameter tus is specified the bundle will include that number of free tus. Alternatively start and end (inclusive both) can be used.

POST

Body:


```
{
    "jobid":4,
```

```
    "tus":100,
    "type":1, //optional, 0 (translation, by default)
    "qamode":1, //optional, (0 default)
    "qemode":1, //optional, (0 default)
    "startts": 12334566665, //optional, long int
    "duets": 12334566665, //optional, long int

}
```

Response:

```
{
    "bundleid": 2,
    "status": 0,
    "reason": "Bundle created successfully",
    "TUs": 1000
}
```

/addbundles

Create a new bundle for a job.

Note: This is the recommended way to define bundles for a project. When less than the required nbundles*tus are available nbundles is internally recalculated to create all bundles with the same number of TUs. In this case the last bundle can contain less bundles.

The response is an array of one-bundle responses.

POST

Body:

```
{
    "jobid":4,
    "nbundles":4,
    "tus":100,
    "type":1, //optional, 0 (translation, by default)
    "qamode":1, //optional, (0 default)
    "qemode":1, //optional, (0 default)
    "startts": 12334566665, //optional, long int
    "tuets": 12334566665, //optional, long int

}
```

Response:

```
[
```

```
        {
            "bundleid": 2,
            "status": 0,
            "reason": "Bundle created successfully",
            "TUs": 1000
        },
        {
            "bundleid": 2,
            "status": 0,
            "reason": "Bundle created successfully",
            "TUs": 1000
        },

        {
            "bundleid": 3,
            "status": 0,
            "reason": "Bundle created successfully",
            "TUs": 1000
        },

        {
            "bundleid": 4,
            "status": 0,
            "reason": "Bundle created successfully",
            "TUs": 1000
        }
]
```


/pausebundle

Set ACTstatus=1 for the bundle, the bundle is paused and no translator/evaluator actions are allowed

POST

Body:


```
{
    "id": 4 //the bundle id
}
```


Response:


```
{
    "rc": 0
```

```
}
```

/resumebundle

Set ACTstatus=0 for the bundle, the bundle is resumed, in normal state and translations and evaluations can be processed.

POST

Body:

```
{
    "id": 4 //the bundle id
}
```

Response:

```
{
    "rc": 0
}
```

/bundlelog

Get the log contents for the bundle. An array with the relevant events for the bundle is returned.

POST

Body:

```
{
    "id": 4 //the bundle id
}
```

Response:

```
[
    {
    "id": 32,
    "ts": 1596898208000,
    "date": "2020-08-08 14:50:08.0",
    "actorid": 0,
    "actionid": 2,
```

```
    "objectid": 12,
    "rc": 0,
    "descr": "Bundle created"
  },
    {
    "id": 47,
    "ts": 1596898335000,
    "date": "2020-08-08 14:52:15.0",
    "actorid": 0,
    "actionid": 2,
    "objectid": 12,
    "rc": 0,
    "descr": "Bundle status changed to 6"
  }
]
```


/editbundle

Change existing bundle parameters. Status change is pointless as it will be recalculated internally.

POST

Body:

```
{
    "id": 4,
    "status": 3, //optional, don't use it
    "startts": 12334566665, //optional, long int
    "duets": 12334566665, //optional, long int
}
```

Response:

```
{
    "bundleid": 4
}
```

/delbundle

Deletes bundle (not the TUs!), the bundle data is deleted from the database and TUs are freed for later bundling and assignation.

POST

Body:

```
{
    "id": 4
}
```

Response:

```
{
    "rc": 0
}
```

/assignbundle

Assign the bundle to translator/posteditor, reviewer or evaluator. This action does NOT notify the actor.

POST

Body:

```
{
    "id": 4,
    "role": 0 // 0:translator, 1:reviewer, 2: evaluator
    "actorid": 3 //set to 0 to unassign
    "mode": 0 // Optional, defaults 0 (future use)
    "pct": 0 // Optional, defaults 100, to limit revision
}
```

Response:

```
{
    "bundleid": 4
}
```

/changeallowhistrev

Change the flag allowing actors to access TUs history. The flag has either value 0 (no permission) or 1 (permission granted. Calling this API will swap values. Only one parameter is passed: the id of the bundle.

POST

Body:

```
{
    "id": 4 //id of the bundle

}
```

Response:

```
{
    "rc": 0 //no error
}
```

/bookmarkbundle

Change the bookmark flag for a bundle

POST

Body:

```
{
    "id": 4 //id of the bundle
    "mode": 0, 1,…, 4 //bookmark mode
    "actorid": 4 //the pm id

}
```

Response:

```
{
    "rc": 0 //no error
}
```

/freebundle

Free the TUs from a bundle for re-bundling. **The affected TUs are reset for**

rework no matter the status they were in.

POST

Body:

```
{
    "id": 4,
    "mode": 0 //opt, 0 by default: full release. 1 free pending, 2 free some
    "tus": 10 //used when mode=2, number of pending tus to release

}
```

Response:

```
{
    "bundleid": 4
}
```

/addtustobundle

Add free TUs to an existing bundle. The new added TUs (always to be reworked) will generate planned evaluations with an even distribution at 0-5-10% rate according to the bundle definition.

The number of TUs to add is a maximum, actual added TUs might be less if there are not enough free TUs.

POST

Body:

```
{
    "id": 4, //id of the bundle to add the tus
    "tus": 100, //number of tus to add (if enough tus are available!)
}
```

Response:

```
{
    "rc": 0,
    "addedtus": 100
}
```

/translatetu

A tu or alt-tu is translated by a translator, notice the bundleid is mandatory. This request is sent by CAT tools when a translator confirms a translation. This action will trigger immediately QA request on the received translation.

POST

Body:

```
{
    "id": 4000, //the tuid
    "bundleid": 4, //the bundle the tu belongs to
    "actorid": 1, // the translatorid
    "source": "xxxxxx",
    "translation": "xxxxxx",
    "alttranslation": "xxxxxx",
    "pretranslation": "xxxxxx",
    "gtranslation": "xxxxxx",
    "transcode": 0, // opt, 1: nonsense, 2: encoding
    "oltransstatus": 0, // opt, 0 for new (default), 1: was already translated
    "transchanged": 0, // 1: translation was changed translated
    "alttranschanged": 0, // 1: alt-translation was changed
}
```

Response:

```
{
    "rc": 0
}
```

Or

```
{
  "rc": 0,
  "qainfo":   {
    "rc": 0,
    "translation_units": [     {
      "id": 4000,
      "qastatus": 4,
      "qaissues":         [
```

```
                {
            "text": "",
            "issue": "Translation is equal to pre-translation, no post-edition was done.",
            "type": 0,
            "subtype": 0,
            "criticity": 5
          },
                {
            "text": "",
            "issue": "Alternative translation equal pre translation, no post-edition was done.",
            "type": 0,
            "subtype": 0,
            "criticity": 5
          },
                {
            "text": "",
            "issue": "Translation and Alternative translation are equal.",
            "type": 0,
            "subtype": 0,
            "criticity": 5
          }
        ]
      }]
  }
}
```


/reviewtu

A tu or alt-tu is reviewed by a reviewer, notice the bundleid is mandatory.

The revstatus indicates which input data is mandatory. For status 0,1,2,4,5, that is when the translation is NOT changed the mandatory fields are id, bundleid, actorid and revstatus.

For status 3 (translation changed) the fields translation, alt-translation and transcode have to be sent also.

POST

Body:

```
{
    "id": 4000, //the tuid
    "bundleid": 4, //the bundle the tu belongs to
    "actorid": 1, // the translatorid
    "revstatus": 3,
    "translation": "xxxxxx",
    "alttranslation": "xxxxxx",


    "transcode": 0, // opt, 1: nonsense, 2: encoding
}
```

Response:

```
{
    "rc": 0
}
```

/reviewallpending

Used to update review status to 2 (STATUS_OK) for all TUs of a bundle which review is not done (0,1 and 5) and have finished trans (with no QA Pending)

POST

Body:

```
{
    "bundleid": 4, //the bundle to update
    "actorid": 1, // the translatorid
}
```

Response:

```
{
```

```
    "rc": 0,
    "message": "Updated successfully 234 TUs"

}
```

/confirmtu

A tu or alt-tu with required QA confirmation is confirmed by a translator as a 'false-positive', notice the bundleid is mandatory. Request sent by the CAT tool to confirm the translator has payed attention to the potential issue and has marked it as false positive.

The response includes the TU QAStatus after the action.

POST

Body:

```
{
    "id": 4000, //the tuid
    "bundleid": 4 //the bundle the tu belongs to
}
```

Response:

```
{
    "rc": 0,
    "qastatus": 4

}
```

/evaluatetu

A tu or alt-tu is evaluated by an evaluator, notice the bundleid is mandatory. The id has to correspond to a TU defined as candidate for QE. This request is sent from a QETool when an evaluator finishes the evaluation of a translation.

POST

Body:

```
{
    "id": 4000, //the tuid
    "bundleid": 4, //the bundle the tu belongs to
    "actorid": 1, // the evaluatorid
    "altmode": 0, //opt, 1 when it is alternate translation evaluation
    "translation": "xxxxxx", //opt, if evaluator provides a new translation
    "qescore": 0,
    "qeinfo": [  //array of issues found
                {
                   "type": 1, //accuracy, fluency,…
                   "subtype": 1, //spelling, undertranslation…
                   "sourcetxt": "THE" // the text the issue relates (optional)
                   "criticity": 2
                 }
             ]
}
```

Response:

```
{
    "rc": 0
}
```

/savetu

A tu or alt-tu is saved by a translator, notice the bundleid is mandatory. The id has to correspond to a TU.

POST

Body:

```
{
    "id": 4000, //the tuid
    "bundleid": 4, //the bundle the tu belongs to
    "translation": "xxxxxx",
    "alttranslation": "xxxxxx",
    "transchanged": 0, // 1: translation was changed translated
    "alttranschanged": 0, // 1: alt-translation was changed
}
```

Response:

```
{
    "rc": 0
}
```

/marktu

A tu is marked by an actor, notice the bundleid is mandatory. The id has to correspond to a TU.

POST

Body:

```
{
    "id": 4000, //the tuid
    "bundleid": 4, //the bundle the tu belongs to
    "mark": 1

}
```

Response:

```
{
    "rc": 0
}
```

/logtu

Get the log contents for the tu

POST

Body:

```
{
    "id": 4 //the tu id
}
```

/preparetus

Used by 3rd-party to fill preTranslation and gTranslation of an array of TUs of a job

POST

Body:

```
{
    "jobid": 1, //the jobId the tus belong to
```

```
    "tus": [
              {
                "id": 1, //the tu id
                "pretrans": "This should be the pretranslation1",
                "gtrans": "This should be the g-translation1"
              },
             {
                "id": 2, //the tu id
                "pretrans": "This should be the pretranslation2",
                "gtrans": "This should be the g-translation2"
              }

           ]
}
```

Response:

```
{
    "rc": 0
}
```

/gettus

Retrieve the actual TUs for a bundle or a job.

The full TU info is returned

POST

Body:

```
{
    "bundleid":1,   //Optional, set if requesting bundle tus
    "jobid":1,      //Optional, set if requesting job tus
    "start":1,      //optional in-bundle-id, starts in 1
    "end":2         //optional
}
```

Response:

```
[
    {
      "id": 100096,
```

```
      "idinbundle": 56,
      "srctext": "Frühkindliche Bildung benötigt gut ausgebildete pädagogische
Fachkräfte",
      "pretrans": "pre-trans",
      "trans": "xxxxxx",
      "transcode": 0,
      "alttrans": "xxxxxx",
      "gTrans": "g-trans",
      "status": 1,
      "statusmsg": "Translated",
      "transstatus": 1,
      "transstatusmsg": "translation done",
      "qastatus": 2,
      "qastatusmsg": "QA to validate",
      "qestatus": 3,
      "qestatusmsg": "QE failed",
      "qainfo": "[{\"text\":\"\",\"issue\":\"Translation is equal to pre-
translation, no post-edition was
done.\"},{\"text\":\"\",\"issue\":\"Alternative translation equal pre
translation, no post-edition was
done.\"},{\"text\":\"\",\"issue\":\"Translation and Alternative translation
are equal.\"}]",
      "qeinfo":
"[{\"type\":1,\"subtype\":1,\"sourcetxt\":\"THE\",\"criticity\":2}]",
      "actorid": 1,
      "evaluatorid": 1,
        "jobtype": 0,
        "translationtype": 0,

   },
      {
      "id": 100097,
      "idinbundle": 57,
      "srctext": "Das heisst wie bitte.",
      "pretrans": "pre-trans",
      "transcode": 0,
      "gTrans": "g-trans",
      "status": 0,
      "statusmsg": "Created",
      "transstatus": 0,
      "transstatusmsg": "translation required",
      "qastatus": 0,
      "qastatusmsg": "QA cannot be done",
      "qestatus": 1,
      "qestatusmsg": "QE pending",
      "actorid": 0,
      "evaluatorid": 0,
        "jobtype": 0,
        "translationtype": 0,
```

```
}]
```

/getpendingqetus

Retrieve the TUs for a bundle which have pending QE (qestatus=1)

The full TU info is returned (a.id, b.idinbundle, a.srctext, a.pretrans, a.trans, a.transcode, a.alttrans, a.altcode, a.gtrans, a.status, a.transstatus, a.alttransstatus, a.qastatus, a.altqastatus, a.qestatus, a.altqestatus, a.qainfo, a.qeinfo, a.actorId, a.evaluatorId)

POST

Body:

```
{
    "bundleid":1,
    "limit":100     //optional to limit the number of TUs in the response
}
```

Response:

```
[
    {
        "id": 10004,
        "idinbundle": 1,
        "srctext": "alle Vornamen: Geben Sie hier bitte alle Vornamen ein, als
o inklusive Rufnamen.",
        "translation": "alle Vornamen: Geben Sie hier bitte alle Vornamen ein,
 also inklusive Rufnamen.",
        "transcode": 0,
        "status": 0,
        "qastatus": 0,
        "qestatus": 0,
        "altqestatus": 0,
        "autorId": 0,
        "evaluatorId": 0,
    },
    {
        "id": 10005,
        "idinbundle": 2,
        "srctext": "Den fehlenden Yard sicherte sich dann Fullback Jerome Morr
is mit dem Seitenwechsel (28:0, PAT Birg).",
        "translation": "alle Vornamen: Geben Sie hier bitte alle Vornamen ein,
 also inklusive Rufnamen.",
        "transcode": 0,
```

```
        "status": 0,
        "qastatus": 0,
        "qestatus": 0,
        "altqestatus": 0,
        "autorId": 0,
        "evaluatorId": 0,
    }
]
```


/getpendingpreptus

Used by 3rd party tools to prepare a Job. Retrieve the TUs for a Job which need preparation, a limit parameter is used to ask for a maximum number of TUs

The answer is a list TUs that is prioritized according to their status.

POST

Body:

```
{
    "jobid":1,
    "limit":100     // to limit the number of TUs in the response
}
```

Response:


```
[
    {
        "id": 10004,
        "src": "de",
        "tgt": "fi",
        "srctext": "alle Vornamen: Geben Sie hier bitte alle Vornamen ein, als
o inklusive Rufnamen."
    },
    {
        "id": 10005,
        "src": "de",
        "tgt": "fi",
        "srctext": "Den fehlenden Yard sicherte sich dann Fullback Jerome Morr
is mit dem Seitenwechsel (28:0, PAT Birg)."
    }
]
```

/autoqa

Dummy endpoint to be served usually by a 3rd system. It receives the full texts of a tu to return an array of QA issues.

POST

Body:


```
{
  "translation_units": [
    {
      "id": 0,
      "srccode": "es",
      "tgtcode": "en",
      "source": "hola",
      "trans": "hello",
      "pretrans": "Hi",
      "altrans": "Morning",
      "gtrans": "Hi",
      "transcode": 0,
        "jobtype": 0,
        "translationtype": 0,

    },
    {
      "id": 1,
      "srccode": "es",
      "tgtcode": "en",
      "source": "hola",
      "trans": "helo my mann adadjandajdpals!",
      "pretrans": "hello",
      "gtrans": "hello",
      "transcode": 0,
      "altrans": "",
        "jobtype": 0,
        "translationtype": 0,

    }
  ]
}Reponse:
{
  "rc": 0,
  "translation_units": [
```

```
    {
      "id": 0,
      “qastatus”: 4,
      "qaissues": []
    },
    {
      "id": 1,
       “qastatus”: 2,
      "qaissues": [
        {
          "issue": "There is no second translation and it was not explicitly indicated that it was impossible to provide one",
          "text": "",
          "issue_type": 5,
          "criticity": 3
        },
        {
          "issue": "Spelling mistake in main translation",
          "text": "helo 12 0: hole, help, helot, hello, halo, hero, hell, held, helm, he lo, he-lo, Heloise",
          "issue_type": 0,
          "criticity": 2
        },
        {
          "issue": "Spelling mistake in main translation",
          "text": "mann 8 8: Mann, man, manna, mane, mans, main, many, man n",
          "issue_type": 0,
          "criticity": 2
        },
        {
          "issue": "Spelling mistake in main translation",
          "text": "adadjandajdpals 13",
          "issue_type": 0,
          "criticity": 2
        }
      ]
    }
  ]
```

/deliveries

Get the list of all deliveries with data required to show lists. The jobid as query parameter is an optional filter.

GET

Query parameter: (optional)

- jobid

Response:


```
[
    {
        "id": 2,
        "jobid": 4,
        "name": "the delivery name",

       "deliveryype": 0,
       "deliverymode": 0,
        "ts": "1970-01-15 00:00:00.0",
       "filename": "filename.tsv",
       "qafilename": "qafilename.tsv",
       "qefilename": "qefilename.tsv",
       "tus": 1000,
        "words": 100000,
       "chars": 100000000,
       "kpis": {
               "tus": 0,
               "words": 0,
               "chars": 0,
               "preparedtus": 0,
               "bundledtus": 0,
               "assignedtus": 0,
               "translatedtus": 0,
               "untranslatedtus": 0,
               "unalttranslatedtus": 0,
               "QAedTus": 0,
               "QAposTus": 0,
               "QAnegTus": 0,
               "QAFPTus": 0,
               "QEplannedTus": 0,
               "QEposTus": 0,
               "QEnegTus": 0,
               "QEScore": 0,
               "progress": 0
        }
```

```
    },
    {
        "id": 3,
        "jobid": 4,
        "name": "the delivery name",

        "deliveryype": 0,
        "deliverymode": 0,
         "ts": "1970-01-15 00:00:00.0",
        "filename": "filename.tsv",
        "qafilename": "qafilename.tsv",
        "qefilename": "qefilename.tsv",
        "tus": 1000,
         "words": 100000,
        "chars": 100000000,
        "kpis": {
                "tus": 0,
                "words": 0,
                "chars": 0,
                "preparedtus": 0,
                "bundledtus": 0,
                "assignedtus": 0,
                "translatedtus": 0,
                "untranslatedtus": 0,
                "unalttranslatedtus": 0,
                "QAedTus": 0,
                "QAposTus": 0,
                "QAnegTus": 0,
                "QAFPTus": 0,
                "QEplannedTus": 0,
                "QEposTus": 0,
                "QEnegTus": 0,
                "QEScore": 0,
                "progress": 0
         }
    }
]
```


/adddelivery

Create a new delivery for a job.

POST

Body:


```
{
```

```
    "jobid":4,
    "name":"my_delivery_name",
    "deliverytype":1,      //optional, (0 default)
    "deliverymode":1,      //optional, (0 default)
    "actorid": 1           //who is asking for it? optional, (0 default)

}
```

Response:

```
{
    "rc": 0  //0 if not error
}
```

/deliverylog

Get the log contents for the delivery. An array with the relevant events for the bundle is returned.

POST

Body:

```
{
    "id": 4 //the delivery id
}
```

Response:

```
[
    {
    "id": 32,
    "ts": 1596898208000,
    "date": "2020-08-08 14:50:08.0",
    "actorid": 0,
    "actionid": 2,
    "objectid": 12,
    "rc": 0,
    "descr": "Delivery created"
  },
```

```
    {
    "id": 47,
    "ts": 1596898335000,
    "date": "2020-08-08 14:52:15.0",
    "actorid": 0,
    "actionid": 2,
    "objectid": 12,
    "rc": 0,
    "descr": "Delivery status changed to DONE"
  }
]
```


/editdelivery

Change existing delivery parameters.

POST

Body:


```
{
    "id": 4,
    "name": "delivery_new_name"
}
```


Response:


```
{
    "deliveryid": 4
}
```


/deldelivery

Deletes delivery, the data is deleted from the database.

POST

Body:

```
{
    "id": 4
}
```

Response:

```
{
    "rc": 0
}
```

8. PGWeb

Typical user access to the Corporate Solution services is through a web application to process files. This application is the PGWeb.

Since end users accessing the system will deal with files, the File Processor module (described later) is mandatory.

PGWeb is served by the Production Access Server. The network setup chosen to install the PAS will define who can access it and from where. If installed in a local server with no open ports to off-premises PGWeb will work as an Intranet Application. If installed in an Internet cloud (AWS for instance) with the right firewalling PGWeb will be working as a secure Internet application, open to anyone, anywhere.

Four different features are implemented:

1. Offer web visitors a description of the services offered.
2. Allow simple and obvious use of services for occasional users. With straightforward *workflow: upload, quote, payment*, processing and delivery.
3. Allow advanced features after registration to the professional user.
4. Offer a simple interface for administration and activity monitoring.

PGWeb is in fact 3 differentiated applications:

1. Application for the occasional user, who lands on the web page normally being redirected from another page and who may access file-processing with just an email address.
2. Application for the professional user, who uses the service recursively and needs to process several files in a different way, access the job logs, usage

statistics, etc.

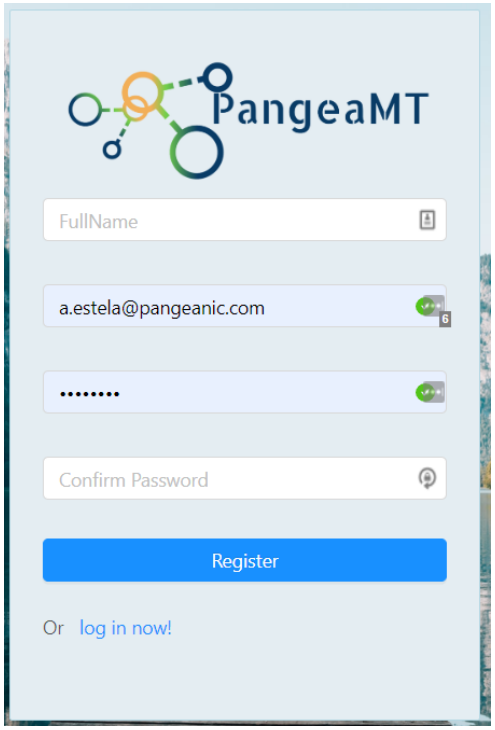
3. Basic Administration Application.

8.1. Web Browsing

The web interaction is on a single page (SPA) using the normal techniques of *scrolling* Menu, Tab panes, ...

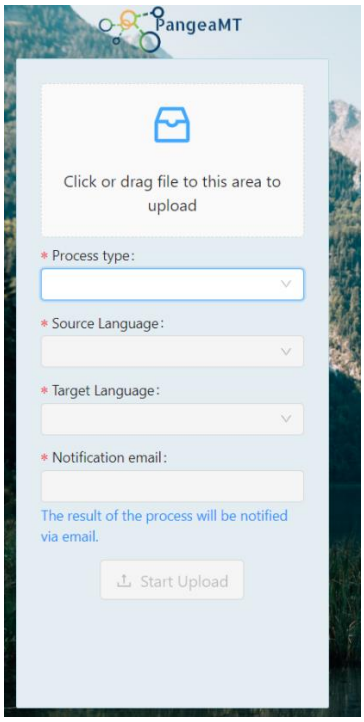
The application maintains the authenticated 'session' requiring login and password During the user interaction.

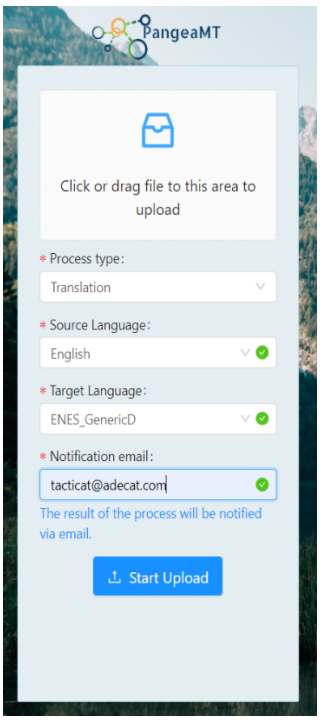
Login popup for users that have already registered or created by an admin and register popup:

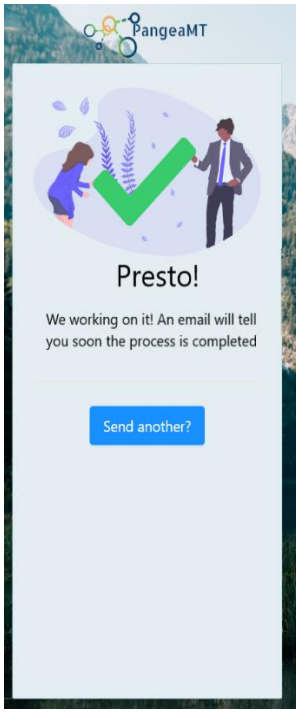


Unregistered user access screenshots:

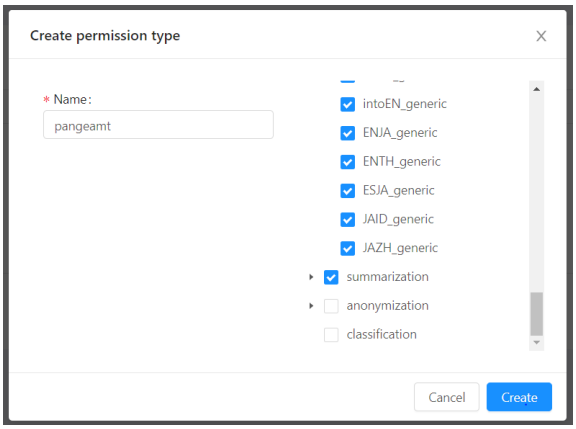
The dialogs can be easily embedded on existing web pages.



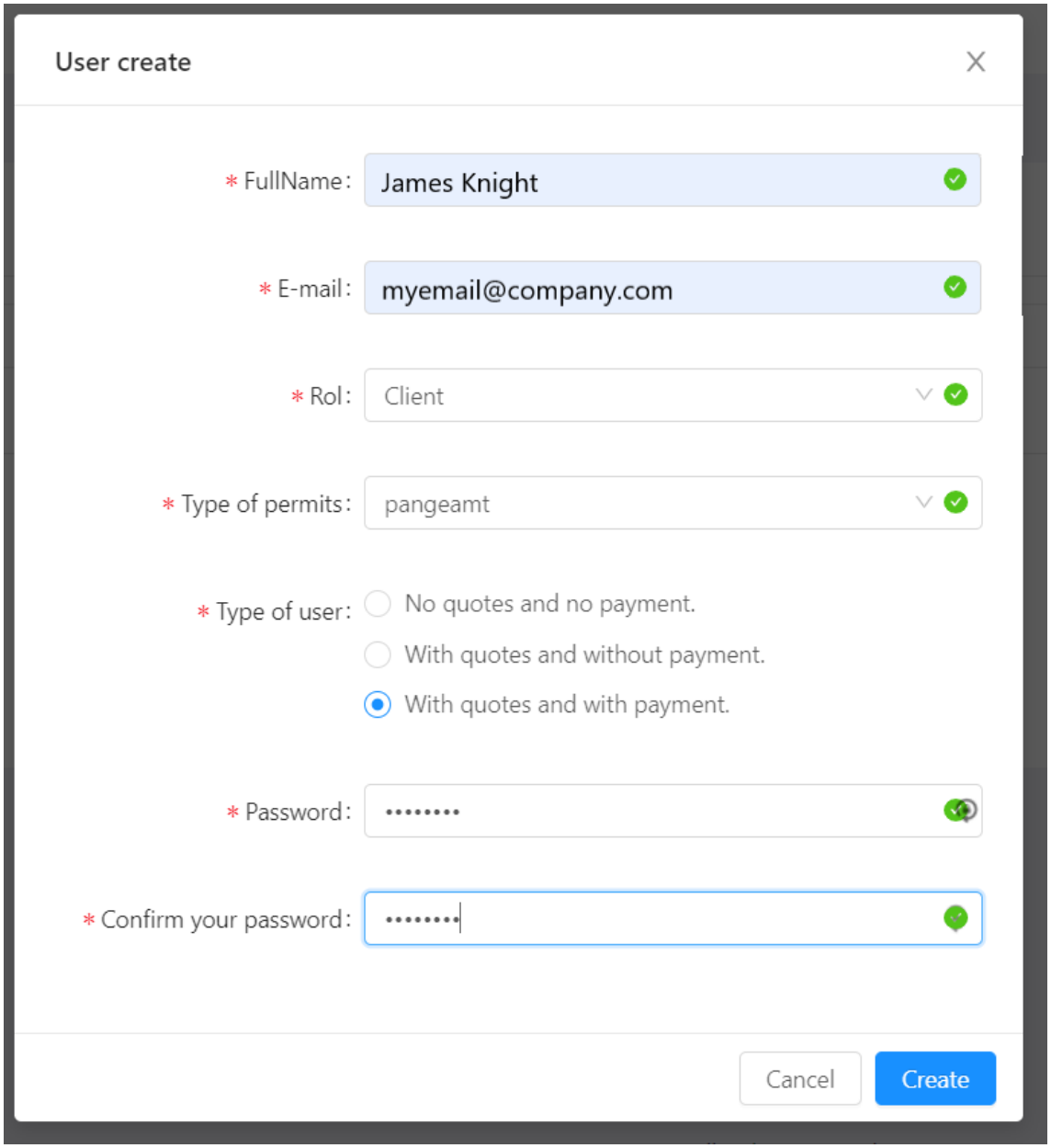




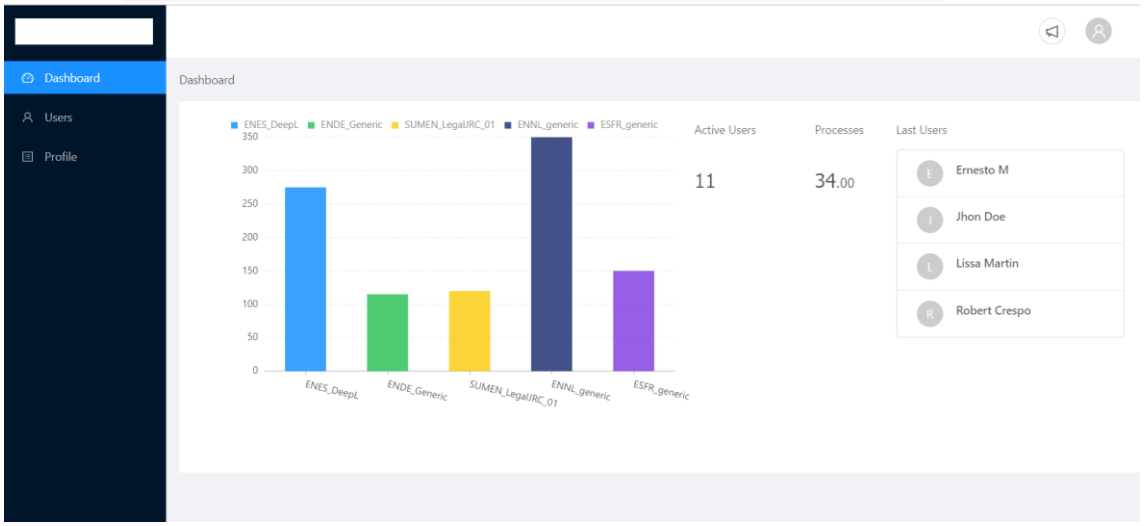
Admin page to create access profiles for new users:



Admin page to create new users or companies (groups of users):



Statistics pages, showing usage at three levels: general, company or user:



8.2. User Management

The application implements standard logging management, *log-in, sign-in,* and maintain profile for 'professional' users and administrators.

User management is implemented at two levels to allow the registration and management of user groups in a 'company'.

Registration of new users can be validated by email.

The different types of user profiles are listed below:

- Unregistered User is a user who accesses the system for the first time, or user who requires services occasionally. It may be a user of another type that has not yet logged in. Access only to the home page, from which they can see the description of the services, company, contact,... And access to the *log-in*, *register* procedures.

- Registered User is a user who has completed and validated the registration process and has logged in. Access the bot only the home page but the 'advanced ' application to process files or other functionalities. This user

may or may not be associated with a **Company**

- Company Manager has access tights to a company's management features: User management and monitoring/usage statistics.
- General Manager, super-admin. Can create new companies and users, associate users to companies and monitor the entire system, a company or a user.

8.3. File management

The object of the PGWeb application is to process files using the Corporate Language Solution backend.

The interface supports *drag&drop* in an upload area or selection of files browsing the file system.

The file process will in turn generate a result, usually in the form of one (or more) file (s) that will eventually be downloaded using a *link*.

8.4. Payments management

Two types of payment procedure are managed:

- Pre-Payment

For unregistered users and those registered for which mandatory payment has been selected. Each request is analysed and valued (quotation). The user accepts the quotation or cancels, if the user accepts he moves to to the payment procedure. Once the payment is made and verified file processing is started.

- Monthly Payment

Same as Pre-Payment but if the user accepts the quotation the process starts immediately. This type of payment affects only registered users (members or not of a company. The amount / balance due is reflected on a monthly basis in the

balance to be paid (by user or company) although billing is undertaken by an independent system.

8.5. Email management

When a casual user, using the simple *workflow*, requests a service and the workflow starts all notifications are done by email:

- The quotation of the work is sent by email, with a *link* to the web application so the user can accept or reject.
- When the process is completed the user is informed by email with a *link* to download the output file.

8.6. Administration

Two levels of administrators are defined: general administrators and a company administrators.

The **General Admin** can create **Users**, **Companies**, and **Company Administrators**.

The General Manager accesses the profile and usage information of the users and companies.

The Company Administrator can create company users and access the profiles and usage of company users.

9. PGB Java Application

PangeaBox is a simple user interface to Pangeanic's Document Processing RESTFul API.

It allows the user to easily submit documents for translation, checking the process and eventually retrieve the translated output.

PangeaBox is a Java application that runs on any platform with JRE installed.

9.1. Requirements

PangeaBox has been tested and verified on Windows 10 and OSX 10.12 where Java JRE V8 was installed.

PangeaBox requires connection access to the http API Server endpoint Usually the endpoint is answering on a non-standard port. Make sure that the firewall settings allow that connection.

9.2. Installation

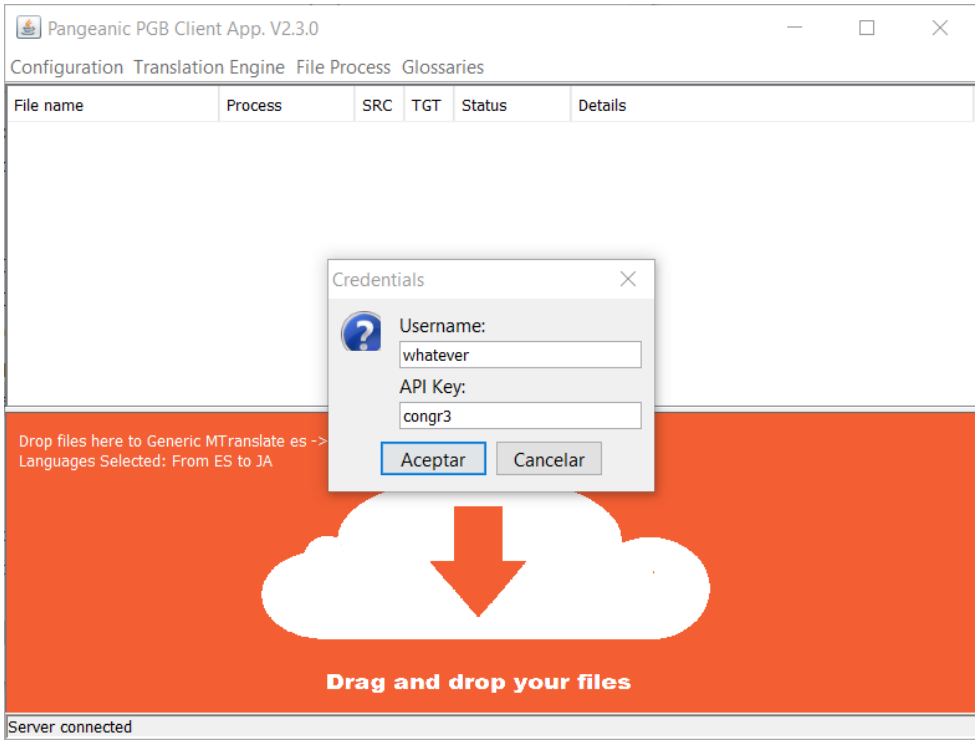
PangeaBox is distributed in a ZIP file containing a jar file and a folder including other jar files (a jar file is the standard way to pack a Java library).

To install, just unzip the file in the desired location. It will create the PangeaBox.jar and the lib folder.

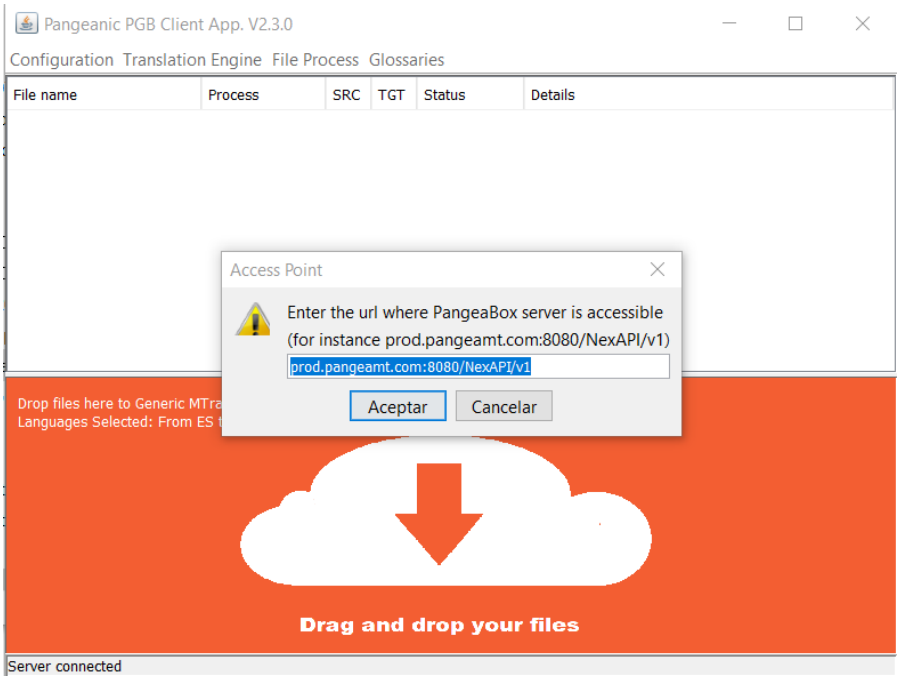
9.3. Running PGB

After installing the APP the interface can be launched double-clicking on the jar file.

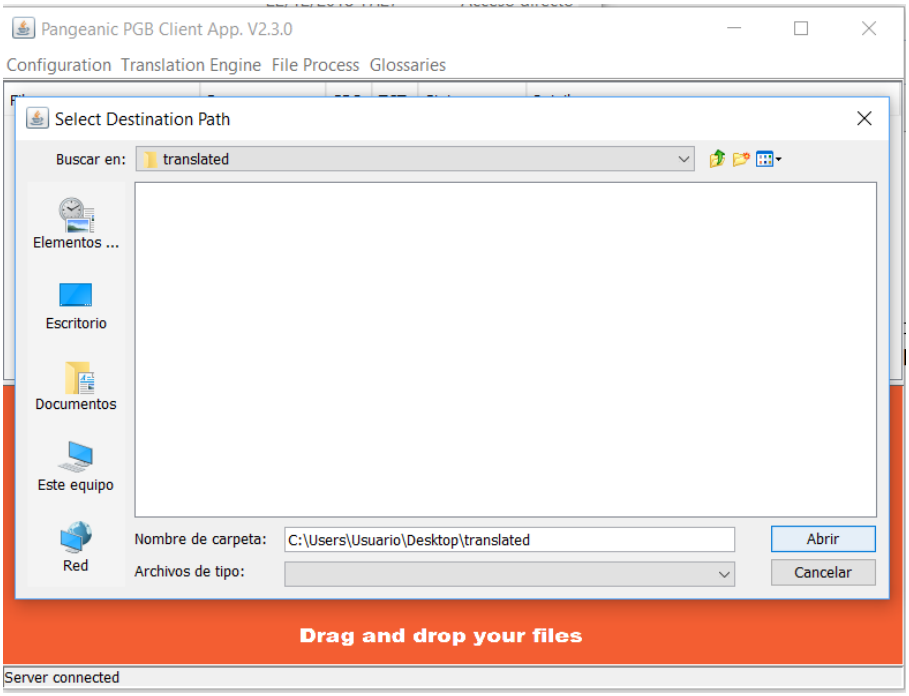
First thing is to configure the access, entering the credentials:



Entering the Production Access Server end point:



And entering the location in the user computer where the processed files will be automatically downloaded:



After the APP is configured the user can choose the process option on the top menu, the chosen option will appear described in the orange lower area.

File process begins just by dragging and dropping one or multiple files into the lower, orange area.

Files in process appear in a list in the top area, with the processing status and the details of the process.

After the output files are processed they are automatically downloaded into configured folder.

Compatible files are “html" , "htm" , "xhtml" , "strings" , "xlsx" , "xltx" , "pptx" , "potx" , "docx" , "dotx" , "ods" , "ots" , "odg" , "otg" , "odp" , "otp" , "odt", "ott" , "txt" , "msg" , "eml", "pdf"

Mail messages (*.msg and *.eml) and PDF files are first converted to plain text and then to docx format

The dropped document appears in the file list and its status is shown as:

- PENDING UPLOAD, the document is still in the box and will be sent to the translation service
- UPLOADING
- UPLOADED
- PREPROCESS (the file is being prepared by the service)
- PROCESSING
- POSTPROCESS (after translation, when a specific formatting is required)
- FINISHED (everything is finished at the service level). The document is transferred back and made available in a few seconds

- DOWNLOADED
- FAILED, the file could not be translated

Multiple files can be dropped in the translation box at the same time.

The translated documents will appear in the "Download Folder" as initially configured.

10. CAT Tools Plugins

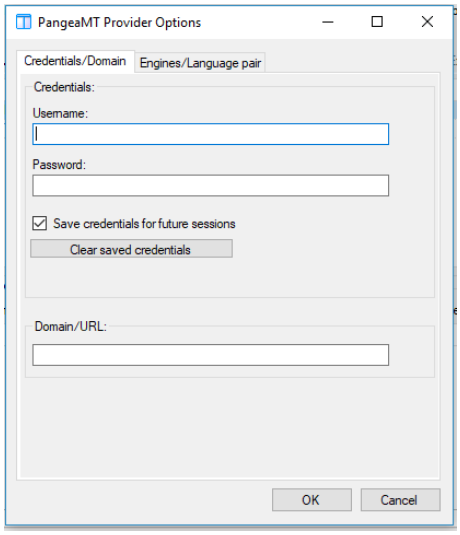
A typical usage of the Corporate Language Solution is as Machine Translation source for Computer Aided Translation Tools.

The Production Access Server implements endpoints for the most common CAT Tools using specific Machine Translation plugins.

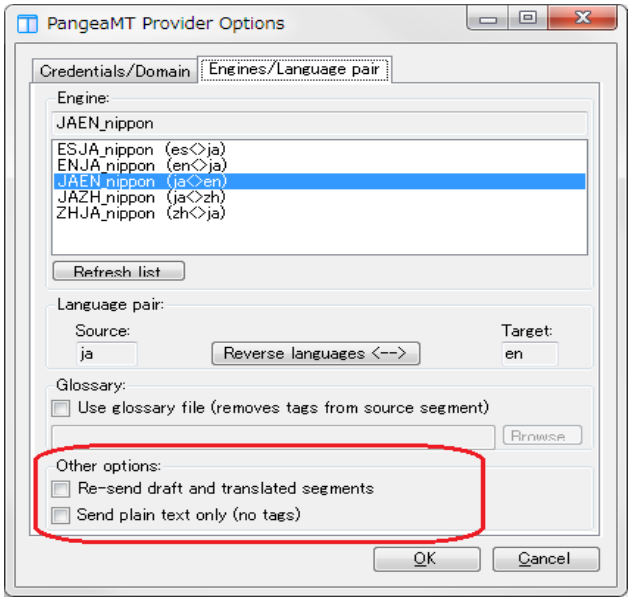
Pangeanic provides plugins for:

- Trados Studio
- MemoQ
- Memsource
- OmegaT

The different CAT tools will allow configuration to some extent. For instance it is possible in Trados to configure if validated segments can be sent back to the Production Access Server to be used in Online Learning. In all cases, the credentials have to be entered, together with the Production Access Server endpoint address:

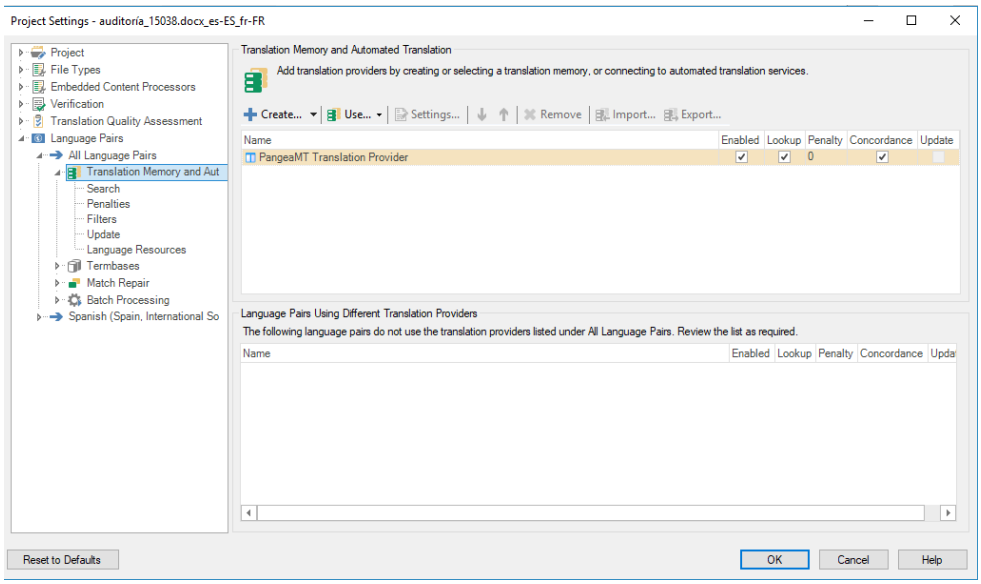


The first step to select PangeaMT is to select the MT provider, so the right engine is used during the tasks:



The first step to select PangeaMT is to select the MT provider, so the right engine is used during the tasks:

Other CAT tools can be integrated easily using our generic API:



11. File Processor Module

Users can request the process of files rather than plain text chunks. The File Processor Module implements the needed logic to deal with the most common filetypes.

Users access the File Processor functionalities implicitly when they are using the different supplied interfaces which deal with files:

- The generic API when using the processfile endpoint
- The java application PGB
- The web application PGWeb

11.1. Main Functions

Users running any of the supplied interfaces have to enter their assigned credentials (APIKey) which will determine the list of processing engines they can use with the documents.

An engine has to be selected before a user can request a document to be translated. Once selected, users will use the translation file drag & drop-ing into the PGBClient.

The general workflow to process the files is as follows:

1. The file is received at the Production Access Server, assigned a unique id and queued for process in the File Processor.
2. The file is analysed to verify the format and the text segments are extracted in a temporary file (xliff format)
3. The segments are sent to the Production Access Server to be routed to the

right engine

4. Dockerized engines are flared up when needed to process the segments
5. The processed texts are sent back to the File Processor which in turn updates the temporary file with the bilingual info.
6. When all text is processed, the output data is merged into the original document keeping the original document layout and format.
7. When the new, processed file is created, the client interface can be used again to download it into the user machine.

11.2. Compatible Formats

The service currently accepts the following file formats:

- txt, text plain files. The text to process is segmented by the obvious dots and by the new-line character.
- rtf, format is preserved
- MS Office family, only new file formats are accepted (docx, pptx and xlsx)
- Open Office family, all formats are accepted
- Microsoft Mail, msg files are accepted but after process the format is not preserved and plain text is returned.
- PDF files. Two options are being used in the current version. In the 1st one only "searchable" pdfs are accepted. The pdf is converted to text, processed and plain text is returned, no layout is preserved. The 2nd option the PDF is OCR'd and whatever the PDF contents the detected text is processed and returned preserving format and layout.

11.3. Glossaries Functionality

When translating a document, it is sometimes interesting to previously define which terms and expressions have to be kept invariant, preserving the original wording, or require a specific (custom) translation. That's usually the case with the names of brands or brand terminology, proper names, jargon expressions, etc.

The PGB client (the java client application running at the user's machine) and the generic API offer the possibility to upload one or more 'glossary' files. These files define the terms/expressions for custom translation. It is possible to decide which glossary file to use if any when a document is requested for translation.

Glossary files are plain text files using the TAB separator.

11.4. System Architecture

The File Processor is a multi-process Python application requiring Python3.5+ and a list of requirements defined in the standard requirements.txt

For the extraction and merge subprocesses and for different format conversions a number of java jar libraries are required. The full set is distributed with the process server package.

The server is configured using a config.ini file, only relevant fields are shown:

```
[Main]

directorMaxConcurrentProcess = 5



mysqlHost = localhost

mysqlUser = root

mysqlPassword = password

mysqlDb = relay



relayMaxWorker = 1

relayMaxSnippetsPerRequest = 32
```

11.5. Setup and Installation

The Process Server is a multi-process Python application requiring Python3.5+ and a list of requirements defined in the standard requirements.txt

For the extraction and merge subprocesses and for different format conversions a number of java jar libraries are required. The full set is distributed with the process server package.

11.6. Configuration

The server is configured using a config.ini file, only relevant fields are shown:

```
[Main]

directorMaxConcurrentProcess = 5


mysqlHost = localhost

mysqlUser = root

mysqlPassword = password

mysqlDb = relay

relayMaxWorker = 1

relayMaxSnippetsPerRequest = 32
```

11.7. Operation

Refer to the Service Operations Manual.

12. Online Learning. Online Trainer Module

The Online Trainer Module is an optional feature allowing ongoing engine customization and refinement in the installed engines.

A common use in industry of natural language processing neural engines is providing initial hypotheses, which are later supervised and post-edited by a human expert. During this revision process, new training data is continuously generated, and the experts' corrections are excellent material for the engine to learn from error, providing high-priority human-approved statistics. Engines can benefit from these new data, incrementally updating the underlying models under an online learning paradigm. Therefore, the systems are continuously adapting to a given domain or user.

The Online Trainer module implements two functionalities:

- Receive user corrected data and use it to incrementally improve the neural engine and
- Manage the different instances that evolve differently from a single base model.

The first functionality is achieved in collaboration of the Production Access Server that receives the new data from users. After new data is received, the module will store it and periodically send a re-train request to the engine with the new data. Potentially, this can happen on-the-fly in some circumstances, although this scenario is only required when using online learning together in professional translation scenarios (CAT tools and human post-editors).

The data used to retrain has to be stored in order to allow future re-trains from scratch that might be needed if the neural engine is upgraded to a new

architecture.

Engine re-training creates a new "mirror" instance for a particular application or client which is saved and can be re-called at a later stage. The preferred terminology of that particular application or client terms are stored and kept for the next use.

12.1. Engine Instances ("Mirrors" or "Flavors")

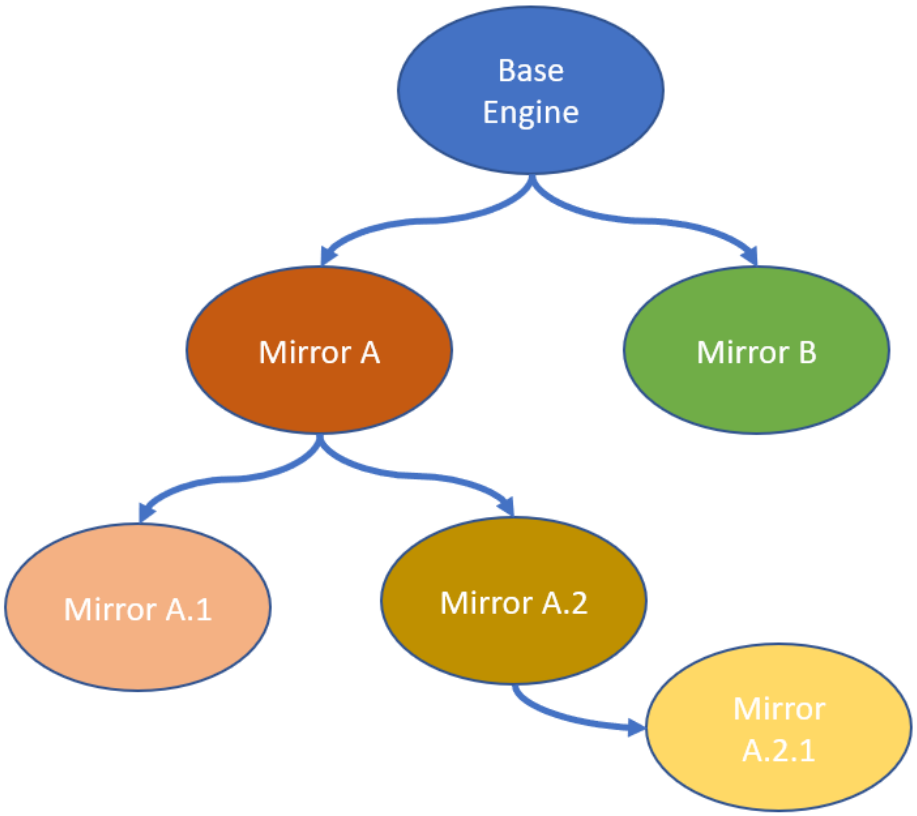
When an engine is installed, its initial language model defines de Base Engine. Before starting the re-train module, the Online Trainer module instantiates a new language model, actually creating a perfect copy of the base engine. The cloned engine starts retraining with the user corrected data and the language model starts diverging from the initial one. Let's call the new engine "Mirror A".

The "Mirror A" engine *learns* the characteristics of the text that is being translated (post-editing) during the scope of a project or afterwards as feedback, let's call it "Project A".

New processing tasks belonging to Project A can be assigned to Mirror A engine with an expected improved performance.

When a new project is started, the administrator can decide whether to use the initial Base Engine or the evolved (Mirror) Project-A engine. And in both cases, the administrator will have had to decide if the corrected data from the new Project can be used to improve the engine used or if a new instance has to be created, let's call it Mirror A.1 or Mirror B (depending if the A engine or the Base engine is chosen).

The following diagram shows the general case:



Base Engines should never be re-trained unless there is a major update or a lot of new in-domain data. However, an organization can create several new engines from an initial Base Engine - all based on the existing Base Engine and all adapting differently to new project requirements (online training),

If we think about translation engines in a big organization, we could have a generic Base Engine to translate from English to Dutch but after some time and projects, the need might arise to have different English to Dutch engines for domains like:

- Legal (Mirror A)
- Legal-Contracts (Mirror A.1)

- Marketing (Mirror B)
- Marketing-WEB-Services (Mirror B.1)
- HR (Mirror C)

And so on.

The number of instance engines cloned directly or indirectly from a single base engine is 30 in the default licensing schema.

The Online Trainer must keep track of all instances, be able to clone existing engines and catalogue them to allow differentiated access to users in the organization.

There are some restrictions to notice:

- new instances can only be created in the same machine where the original engine cloned is installed.
- GPU instances share the assigned GPU in the host machine making concurrent use of original and clone not optimum.
- The number of instance engines cloned directly or indirectly from a single base engine is 30 in the default licensing schema.

13. Tech Specifications. Deployment Scenario.

In this section guidelines to setup a full solution are given, with some real examples.

All components have been tested and homologated to run on Ubuntu 16.04 TLS or higher version. GPU engines require Ubuntu 18 with CUDA drivers.

13.1. Hardware Requirements

To run the client components (PGB, PGWeb, CAT tools) standard client computers are enough provided they can install and run Java 8 (PGB), a modern Web Browser or the CAT Tool.

The rest of the components (Server modules) are usually installed in one central server (hosting PAS, DB and the optional File Processor and On-line Trainer) and one or more machines to host the engines.

The minimum configuration to run the central component (PAS + DataBase) is a 2 CPU, 4 GB Ram machine (equivalent to an AWS t2.medium instance).

When the File Processor or OnLine Trainer are installed in the same machine the minimum requirements are 2 CPU and 8GB (equivalent to AWS t2.large instance).

The recommended configurations are machines with 4 CPU and 16GB (t2.xlarge) or even 8 CPUs and 32Gb for high performance load (t2.x2large).

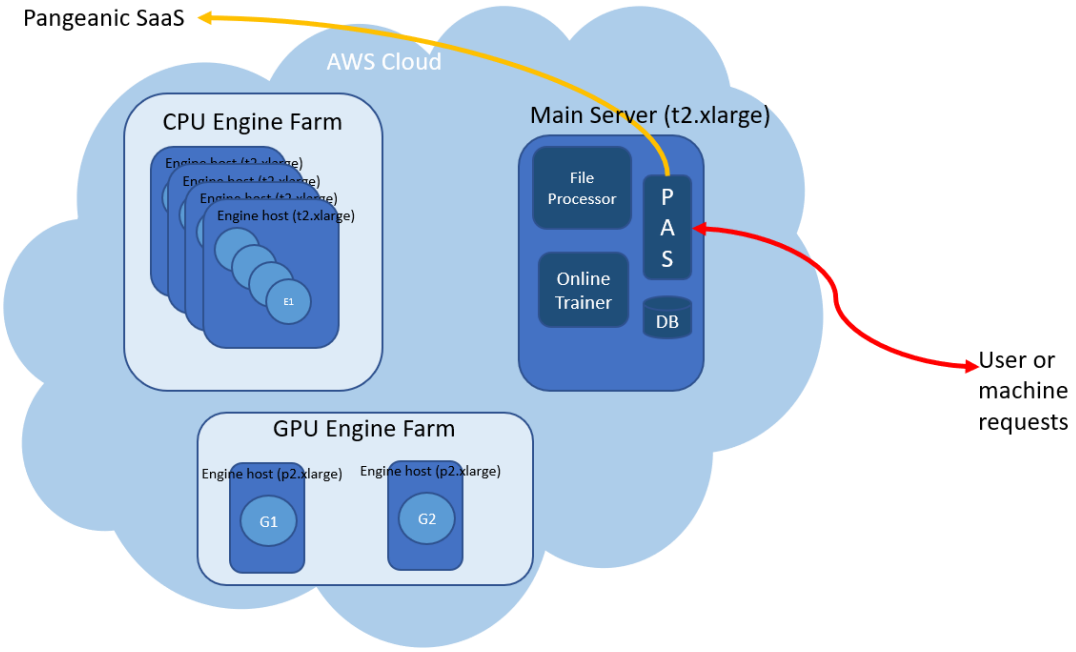
CPU engines require a CPU core per instance to run concurrently. The PAS will take into account the amount of CPU cores available at the machines hosting the engines to wake up or stop the engines as they are requested but the typical number of concurrent engines working has to be planed in advance to plan for the machine(s) hosting the engines.

Every CPU engine will require typically 4GB and a CPU core. To run 4 engines

concurrently on the same box the required resources are then 16GB and 4 CPUs (a t2.xlarge AWS instance would do it).

13.2. Example scenario

The next scenario is for a full deployment of the Solution on Amazon (AWS) cloud.



The Main Server (a t2.xlarge instance) hosts PAS, DataBase and the modules for File Processing and Online Trainer. For high reliability the machine can be duplicated to be used as backup system.

For high load the t2.xlarge instance can be replaced by a t2.x2large instance duplicating the available resources.

A number of t2.xlarge machines are used (4 in this case), each hosting 4 CPU

engines that can be run concurrently. These machines form the CPU engine farm. More engines can be deployed in each machine provided the use is normally non-concurrent.

2 p2.xlarge instances (GPU instances with a graphic card each) form the GPU Engine Farm for high performance processes. A single engine is deployed in each machine to allow uninterrupted use of the neural engines.

Notice that in this case the PAS is configured with a connection to Pangeanic SaaS, and that means that the system can answer to requests directed to engines other than the locally installed.

14. Maintenance activities and fees.

Pangeanic offers a maintenance package on all software products. The maintenance includes retraining of the base engines, but most importantly allows keeping pace with all technological evolutions.

Artificial Intelligence (AI) and more specifically deep learning and neural engines applied to language are very active fields with constant improvements and innovation. Our maintenance policy ensures that our customers have access to improved software assets year after year, and that their investment will not turn obsolete in the short or medium term.

15. Branding, white-labelling.

Pangeanic Corporate Solution is usually serviced as SaaS from Pangeanic Cloud or procured as a PangeaMT product AS-IS, with no other customized branding but its own. Both offerings are targeted to customers using the solution internally at the customer's individually, at department or even corporate level.

It is possible, though, to customize the solution to include the customer branding and style. Pangeanic offers this customized solution that empowers the customer to sell their own service in a geographically determined area or business area / vertical.

Customization is an option and includes two costs: 1) the agreement to distribute the service that has to be agreed with Pangeanic and may include exclusivity clauses and 2) the customization and integration efforts that have to be quoted as a project.

Some tasks that may be part of a customization and integration project are listed below:

- Change in the PGB interface to include the customer branding
- Change in PGWeb interface to include the customer branding
- Integration with customer software for operational workflow and billing
- Integration with customer Dashboard
- Integration with external database and stats
- Changes to the users enrolment
- Ad-hoc, etc.