NASA 889 Compliance SAM Tool

Overview

This search uses the openGSA sam.gov Entity Management API. Vendors that have selected "DOES NOT" for both FAR 52.204-26(c)(1) and (2) are marked as compliant. A vendor is not selectable if they do not meet this requirement, do not have an active registration status, or have active exclusions. Selecting a compliant vendor will download PDF record of their compliance.

- https://open.gsa.gov/api/entity-api/

Search results omit entities without representations and certifications, which are those with a "purpose of registration code" of "Federal Assistance Awards" only (Code Z1) and child entities (non-zero/non-null EFT Indicator). This behavior is implemented in `client_v[0-9][0-9].js` calls to the python backend.

Objective

The purpose of this tool is to enable the broadest possible user-base, including non-procurement-experts, the ability to determine vendor 889 compliance from their SAM record as quickly as possible and with little or no training.

Libraries

The 889 Compliance SAM Tool is written in the Flask Python micro web framework and uses CSS/JS from the fomantic-ui development framework. PDFs are generated using the WeasyPrint Python library.

- Flask https://flask.palletsprojects.com (BSD3)
- Fomantic https://fomantic-ui.com (MIT)

Other python libraries include:

- requests https://github.com/psf/requests (Apache 2) -- Requests is a simple, yet elegant, HTTP library. -- Used to make calls the SAM Entities API in python
- Flask-WeasyPrint https://github.com/Kozea/Flask-WeasyPrint (MIT) -- Make PDF with WeasyPrint in your Flask app. -- Used to generate PDF records of vendor 889 compliance on the fly

The following libraries from requirements.dev.txt are not required for running a production instance, but may be useful in development:

- pytest https://github.com/pytest-dev/pytest/ (MIT) -- Makes it easy to write small, readable tests, and can scale to support complex functional testing for applications and libraries. -- Used to help write and execute tests.
- pylint https://pylint.pycqa.org/en/latest/ (GPL2) -- Pylint is a static code analyzer -- Used to help in writing clean consistent code

The python Flask application can be deployed in a production environment using a variety of tools. While they are not required, these tools may be useful:

- honcho https://honcho.readthedocs.io/en/latest/ (MIT) -- A command-line application which helps you manage and run Procfile-based applications -- Used to manage the Flask app in a production environment.
- gunicorn https://docs.gunicorn.org/en/stable/ (MIT) -- WSGI HTTP Server for UNIX -- Used to run multiple workers for the Flask app
- nginx (BSD2) or apache (Apache2) web server -- Full webservers for production deployment

Additional libraries used by the tool can be found in the requirements.txt (python libraries) and package.json (javascript libraries) files.

Features

The tool performs three main tasks.

1. Search term pre-processing. - Improves the quality of search results returned by the SAM Entities API by modifying the search expression provided by the user. Regular expressions are used to identify SAM UEIs, and US and NATO cage codes. See the search_preprocessor.py file and associated tests in tests/test_search_preprocessor.py
2. Determine entity compliance status. - Call the SAM Entities API and append the response data with a "samToolsData" section for each vendor containing compliance information. See compliance/compliance_rules.py for compliance rules and associated tests in tests/test_compliance_rules.py.
3. Render PDF record of vendor compliance.

Note: The code can display a "Recent updates" toast message to inform users of changes to the tool. Populate messages and message expiration dates in the `recent_website_update_messages.json` file. This file is read during application startup, so restarting the application is required. Example json file:

```
[
    {
        "message": "Updated NF1883 to version 1.1",
        "expiration_date": "2022/09/03"
    },
    {
        "message": "Search results cleanup. Fewer entities without
Reps&Certs will appear in the search results. Entities registered for
federal awards only and child entities are omitted.",
        "expiration_date": "2022/09/03"
    }
]
```

NASA SAM Tool API Endpoints

The 889 compliance SAM Tool provides two API endpoints. These endpoints allow for other tools that require 889 compliance data from SAM to obtain this from the SAM Tool. They can be found in `__init__.py`.

`<HOST_URL>/api/entity-information/v3/entities`

`<HOST_URL>/api/file-download/summary`

Both endpoints accept the same arguments and call the same function internally. The difference between the endpoints is in their responses.

The entity-information endpoint returns the complete information for all vendors in the search results.

The file-download endpoint will return a PDF summary of 889 compliance information, only if there is a single vendor returned by the search, otherwise it will raise an error. CAGE codes are unique to individual entities in SAM. SAM UEIs however include child entities that share the SAM UEI of their parent entity. Therefor if searching using SAM UEIs you must include `&entityEFTIndicator=` to ensure only the parent entities (which have entityEFTIndicator of 0000 - implemented as null) are returned.

Both endpoints use the same arguments as the OpenGSA Entities API, but with two additional arguments.

- A additional argument, 'samToolsSearch' can be used and will use the previously described search pre-processor to set the search arguments before it is passed to the SAM Entities API.
- 'samToolsData' can be included in the 'includeSections' argument of the SAM Entities Management API

Examples:

`<HOST_URL>/api/entity-information/v3/entities?samToolsSearch=mcmaster&includeSections=[samToolsData,entityRegistration,coreData]&registrationStatus=A`

`<HOST_URL>/api/file-download/summary?ueiSAM=LMLHENSX2W97&entityEFTIndicator=`

`<HOST_URL>/api/file-download/summary?cageCode=9B9F4`

Code structure

- `__init__.py` --> Main Flask application
- samtools/compliance --> Objects for determining compliance from SAM data
- samtools/sam_api --> Run search preprocessor, call SAM Entities Management API, append compliance data to response
- production --> Scripts for setting up nginx, gunicorn, etc.
- tests --> Tests

Tests can be run using pytest:

`pytest tests/test_entity_information.py`

`etc...`

Local development installation instructions

Clone the repository into a directory

```
```

```
git clone <CODE_REPOSITORY>
cd samtools
```

Install python >= 3.8, pip, and virtualenv using apt-get, brew, etc.

These commands must be run as root or with superuser privileges (sudo). Example: `sudo apt-get update`

```
apt-get update
apt-get install -yq git python3 python3-pip
pip3 install --upgrade pip virtualenv
```

Install weasyprint

See instructions on installing weasyprint in different operating systems here: https://weasyprint.readthedocs.io/en/stable/install.html

Build the npm modules, add static css/js files to the static folder, create a python virtual environment, and update the 'last_updated.txt' file

The SAM Tool comes with a bash script that automates several of the build steps.

```
cd samtools
bash build_samtools.sh
source venv/bin/activate
```

```
pip install -r requirements.dev.txt  # install development-only python
requirements
```

Setup instance-specific data (SAM Entity Management API key and contact email)

Create an instance folder and add the Flask configuration file with the API key and contact email.

```
mkdir instance
echo "SAM_API_KEY = '<ADD_SAM_API_KEY>'" > instance/samtools.cfg
echo "CONTACT_EMAIL = '<ADD_CONTACT_EMAIL>'" >> instance/samtools.cfg
```

Run the Flask application:

```
flask --debug --app samtools run
```

Optional: use gunicorn as a web server gateway interface (WSGI)

This is not required to run a development instance of the Flask application, but is required for a production instance. If you would like to run gunicorn in front of Flask you can run the Flask application with:

```
gunicorn samtools.wsgi:app
```

NOTE: gunicorn runs on port 8000 by default.

That's it!

Hopefully that all went smoothly and now you can continue to develop and improve the SAM tool on your local machine!

Production deployment

The production deployment uses a reverse proxy and web server gateway interface (WSGI). We use nginx as a reverse proxy and gunicorn as a WSGI, but we do not use any specialized features of these programs and it's expected that anything with similar capabilities will suffice.

Updating the site

Production uses a soft link from a specific version (e.g., `/opt/samtool_v0.3.12`) to a production folder (`/opt/samtool_production`). To update the site just point the production soft link to the desired version folder and reboot supervisor and nginx:

```
sudo ln -sfn /opt/samtool_v0.3.12 /opt/samtool_production
sudo service supervisor restart
sudo systemctl restart nginx
```

This makes it simpler to roll-back to a previous version if needed.

You can see if any packages have newer versions with `pip list --outdated` and `npm outdated`

Acknowledgements

The NASA 889 Compliance SAM Tool was developed by Benjamin Jensen, Godfrey Sauti, Anne Haley, Charles Liles, Sally Kim, and Emilie Siochi. Policy guidance from Tracy Hall. Please contact us at benjamin.d.jensen@nasa.gov, godfrey.sauti-1@nasa.gov, tracy.h.hall@nasa.gov.

Note