Data Mapping tools

User manual

About Data Mapping Tool

Data mapping tool is an application that allows mapping ontologies and tabular data, generating a YARRRML file.

The application has two different parts:

- back-end: A Python application which performs the mapping process.
- front-end: A React application which allows interacting with the back-end side.

Both parts are communicated thanks to a RESTful architecture.

Requirements

For the front-end, Data Mapping Tools requires:
- npm 8.1.0 or greater.
- node 16.13.0 o greater.
- A list of node dependencies listed in the file package.json.

With respect of the back-end part, before using the application, it's required to meet the following specifications:
- Python 3.8 o greater: https://www.python.org/
- Docker and Docker compose: https://www.docker.com/
- MongoDB: https://www.mongodb.com
- All libraries listed in the file requirements.txt that can be installed through PyPi.

Please, note that MongoDB can be automatically installed and configured when deploying the Docker container.

Getting Started

Back-end

In the first place, we recommend installing the back-end part. The back-end code is hosted in a GitHub repository located at: https://github.com/rhizomik/data-mapping-tool-api

Down below, we listed the steps needed to install the back-end part:

1. Clone the repository. There are different ways to clone a repository. For example, in a terminal with git installed, you can execute the following command:

```
git clone https://github.com/rhizomik/data-mapping-tool-api.git
```

2. Go to the folder data-mapping-tool-api

```
cd data-mapping-tool-api
```

3. (Optional) Create a new Python virtual environment and activate it

```
python -m venv venv

source venv/bin/activate
```

4. Install all the dependencies listed in requirements.txt:

```
pip install -r requirements.txt
```

5. You need to create a file named .env with all the environment configuration. Please, refer to Section Environment files for more details.
6. Start the docker container. (It's probably that you'd need root permissions):

```
docker-compose up -d
```

7. Execute seed.py:

```
python seed.py
```

8. Now, it's time to launch the flask server to enable the API:

```
flask run –host localhost --port 5000
```

9. Done! The back-end is installed and configured.

The next time you need to enable the back-end, you only has to repeat the steps 6 and 8 (and activate the virtual environment if you has created one).

Front-end

The front-end part of the application iis hosted on another GitHub repository whose address is: https://github.com/rhizomik/data-mapping-tool-client

The steps to install and launch the front-end part are the following ones:

1. Clone the repository:

```
git clone https://github.com/rhizomik/data-mapping-tool-client.git
```

2. Open the folder data-mapping-tool-client:

```
cd data-mapping-tool-client/
```

3. Install all the required dependencies:

```
npm install
```

4. You need to create a file named .env with all the environment configuration. Please, refer to Section Environment files for more details.
5. Launch the React application:

```
npm start
```

The next time you need to launch the application, you only need to execute the command `npm start`.

Environment files

In each part of the application, you will need to create a .env file where defining all environment constants required by the application.

Back-end

The .env file has the following aspect:

```
FLASK_APP=app
FLASK_ENV=development

SECRET_KEY=4f3f980dc2ad8924ab6a4de3dd66a183a9f6580736be1330d9b5a270716d419d
SERVER_HOSTNAME=localhost
SERVER_PORT=5000

JWT_SECRET_KEY=996826a73d334efc885f132c49cef9a0016fe34c5f504e1defb6a1b46d473a01
JWT_ACCESS_TOKEN_EXPIRES=
JWT_REFRESH_TOKEN_EXPIRES=

MONGO_URI=mongodb://root:password@localhost:27017/db?authSource=admin

ADMIN_EMAIL=test@test.com
ADMIN_PASSWORD=123456
```

You need to generate two secret keys: one for the constant SECRET_KEY and the other one for the constant JWT_SECRET_KEY. It's up to you the way to generate both keys, but you can ask Python for help by execution the following command:

```
python -c 'import secrets; print(secrets.token_hex())'
```

Front-end

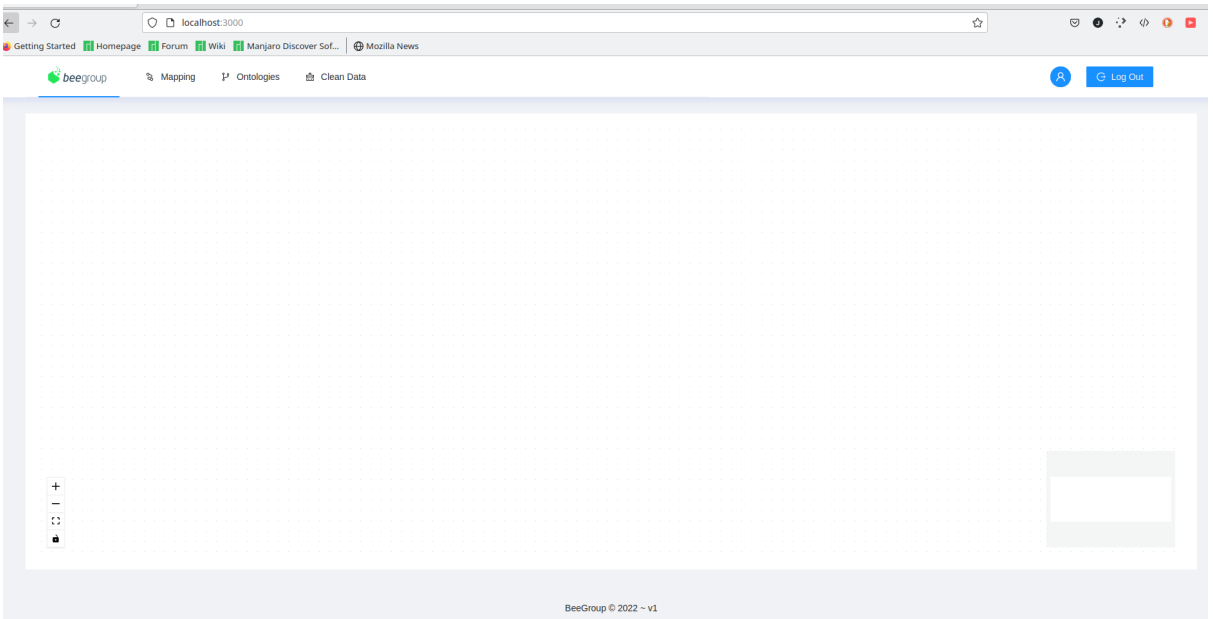
The .env file has the following aspect:

```
REACT_APP_API_URL=http://localhost:5000
REACT_APP_DEFAULT_ONTOLOGY_ID=
WDS_SOCKET_PORT=0
```

Just make sure that the port defined in the constant `REACT_APP_API_URL` matches the port defined for the back-end application.

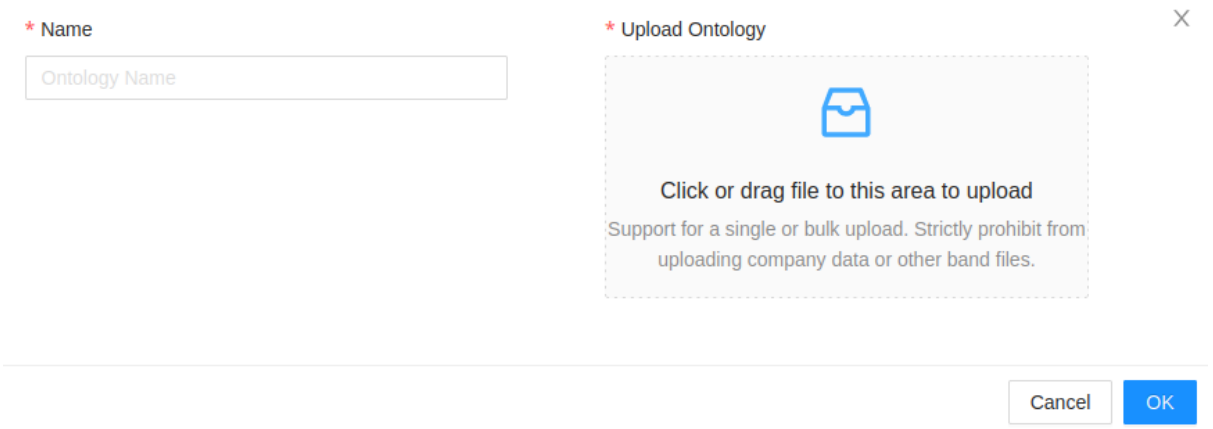
First steps

If you have installed both parts, then it is time to use the application. After launching the front end, you should see the following web interface in your default browser:



The very first step is clicking on the log in button, and insert the admin e-mail and the password you established in the environment file.

At the moment, the application isn't very useful. Let's change the situation. If you go to the tab Ontologies, you can introduce your first Ontology in the application. Just click on the button.

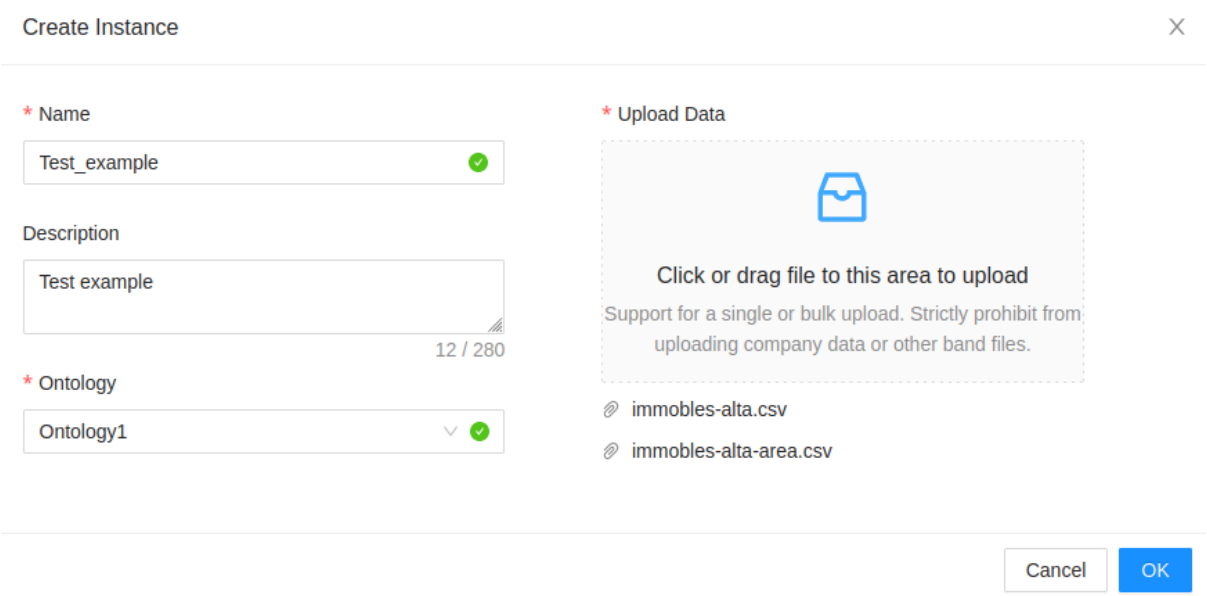


Here you can define an Ontology name and upload an .owl file. If you need inspiration, you have some examples in the folder **example/ontologies**.

After creating the ontology, you should see a table with information about the recently created ontology as well as somo actions:

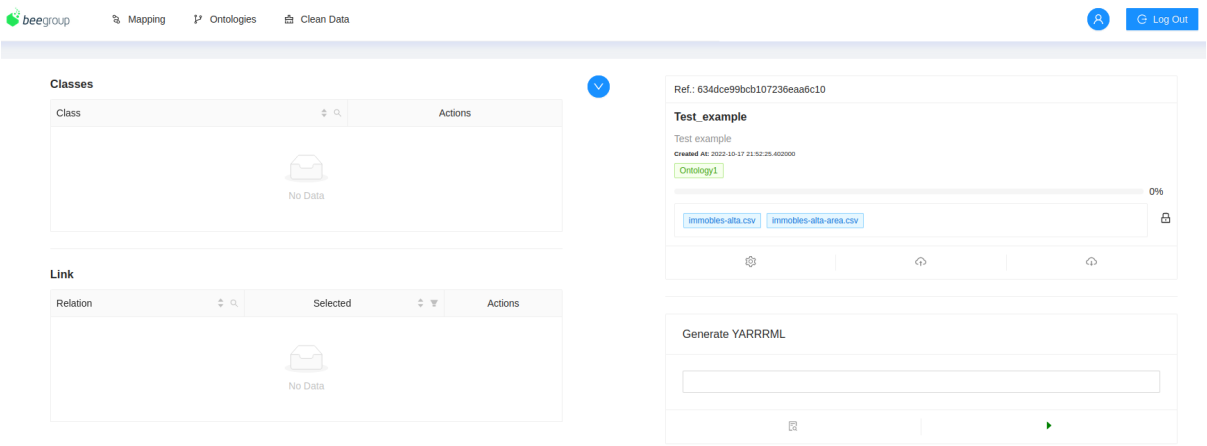
You can edit the ontology , download the .owl file or delete the ontology .

Then, it's time to perform the mapping by going to the mapping section. This time, you have to press the button to create a mapping instance. In this step, you need to attach one (or more than one) .csv files (you have example of .csv files in folder **examples/data**):



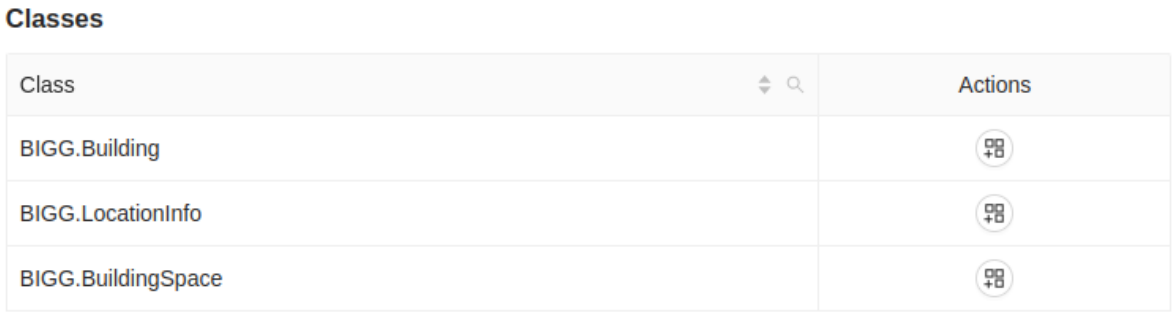
And that's all! You have created the instance. Now you can click on the mapping action button to start the mapping.

At this step, you should see the following screen:

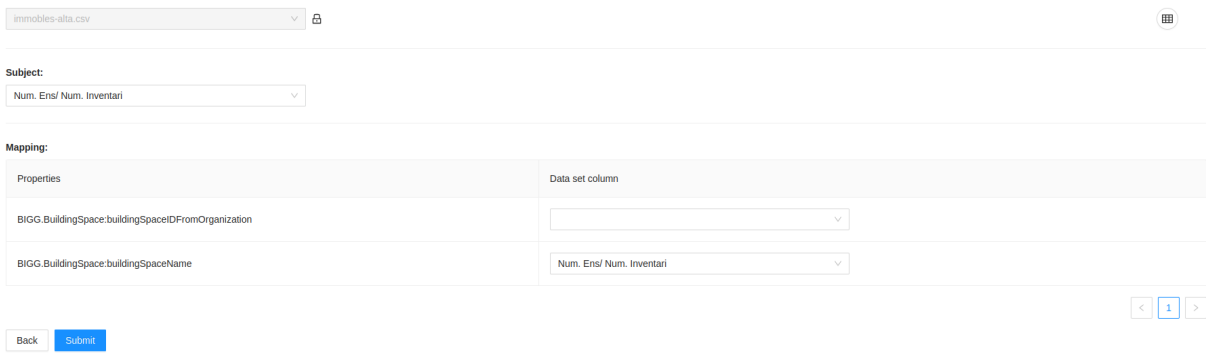


Click on the  button to open a modal dialog where you can choose the classes you want to map. For example, we have selected the following ones:

- BIGG.Building
- BIGG.LocationInfo
- BIGG.BuildingSpace



Then, click on the Map button to map the class. You'll see the following screen:



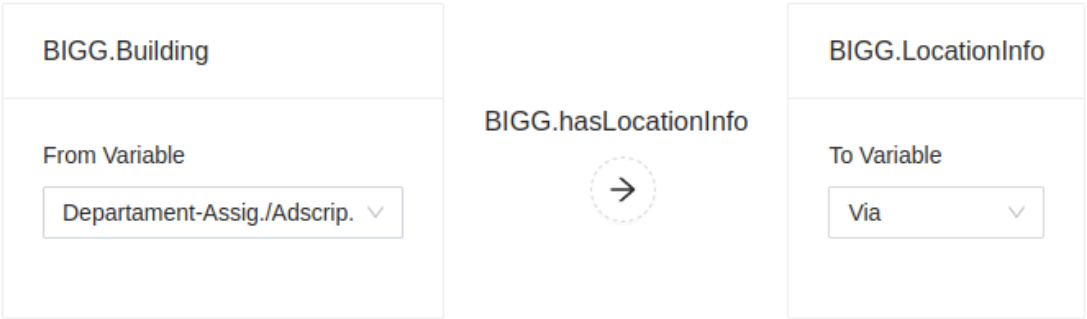
You can use the button  to see a preview of the .csv file.

When you finish, you can click on Submit and then go to the Link section to toggle the relations.

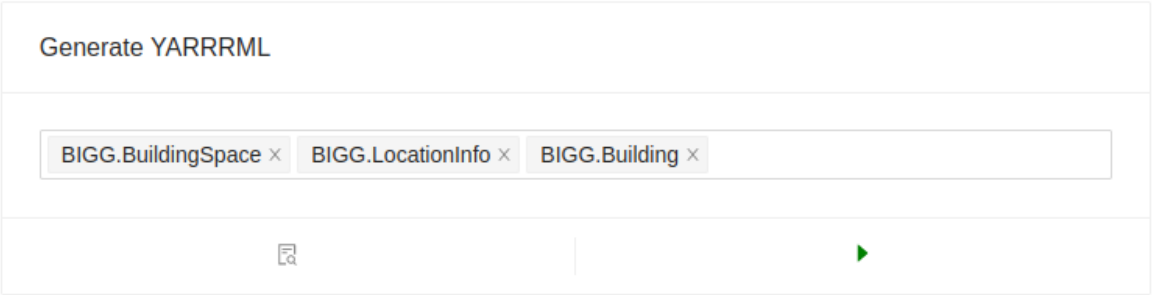
Link

Relation	Selected	Actions
BIGG.hasLocationInfo	✓	
BIGG.hasSpace	✓	
BIGG.hasSubSpace	✓	

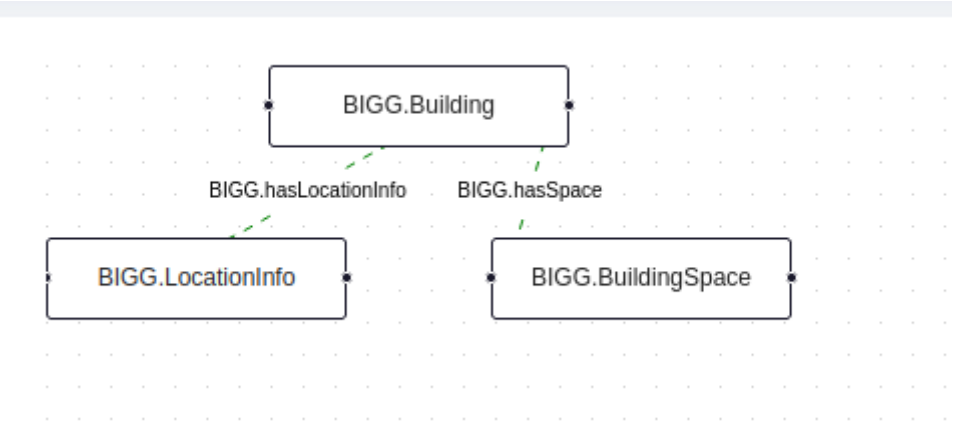
In action, you can define the links between elements in the ontology:



Finally, it's time to generate the YARRRML file, if you go to the Generate YARRRML section, you'll see two different possible actions:



The first one, it's the preview  , which lets you see the resultant ontology graph.



The second one is the Run action , which generates the YARRRML file.

After click in the Run action, you'll be able to download the file:

```yaml
prefixes:
  dbo: https://dbpedia.org/ontology/
  bigg: https://bigg-project.eu/ontology#

mappings:
  buildingspace:
    sources:
      - [ 'immobles-alta.csv~csv' ]
    s: bigg:BIGG.BuildingSpace/$(Num. Ens/ Num. Inventari)
    po:
      - [ a, schema:BIGG.BuildingSpace ]
      - [ schema:buildingSpaceIDFromOrganization, $(Num. Ens/ Num. Inventari) ]
      - [ schema:buildingSpaceName, $(Espai) ]
      - p: bigg:BIGG.hasSubSpace
        o:
          mapping: buildingspace
          condition:
            function: equal
            parameters:
              - [ str1, $(Num. Ens/ Num. Inventari) ]
              - [ str2, $(area_value) ]


  locationinfo:
    sources:
      - [ 'immobles-alta.csv~csv' ]
    s: bigg:BIGG.LocationInfo/$(Num. Ens/ Num. Inventari)
    po:
      - [ a, schema:BIGG.LocationInfo ]
      - [ schema:addressAltitude, $(area_value) ]
      - [ schema:addressLatitude, $(area_value) ]
      - [ schema:addressLongitude, $(area_value) ]
      - [ schema:addressPostalCode, $(Municipi) ]
      - [ schema:addressStreetName, $(Via) ]
      - [ schema:addressStreetNumber, $(Num. via) ]
      - [ schema:addressTimeZone, $(Espai) ]

  building:
    sources:
      - [ 'immobles-alta.csv~csv' ]
    s: bigg:BIGG.Building/$(Num. Ens/ Num. Inventari)
    po:
      - [ a, schema:BIGG.Building ]
      - [ schema:buildingClosingHour, $(Tipus d'ús) ]
      - [ schema:buildingConstructionYear, $(Ref. Cadastral) ]
      - [ schema:buildingIDFromOrganization, $(Num. Ens/ Num. Inventari) ]
      - [ schema:buildingName, $(Via) ]
      - [ schema:buildingOpeningHour, $(Num. via) ]
      - p: bigg:BIGG.hasLocationInfo
        o:
          mapping: locationinfo
```