Universitat de Lleida

TREBALL FINAL DE MÀSTER

ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: **Francesc Contreras Perez**

Titulació: Màster en Enginyeria Informàtica

Títol de Treball Final de Màster: Data integration tool for transform existing data sources to homogeneous linked data

Director/a: Roberto García González i Eloi Gabaldón Ponsa

Presentació

Mes: Juny

Any: 2022

Table of contents

List of Figures

List of Tables

1 Introduction

Nowadays, data storage and analysis are essential in our society to find solutions such as predicting the weather, fraud transactions, health problems, stock prices, and improving energy savings.

However, data providers (enterprises, organisations, or individuals) do not follow the same way of structuring their data. As a result, problems arise when they want to join different sources to analyse it. For this reason, when this happens, we are obliged first to harmonise the data to a common data model.

For instance, assume that an enterprise would like to consume different types of data provided by multiple sources, such as electrical and gas consumption, and the aim would be to perform an exhaustive analysis and contrast the other sources. The possibility of having different ways to obtain the data may be high. Moreover, each source has its way of storing the data (MongoDB, MySQL, InfluxDB, Neo4J, CSV files), regardless of the data model. For this reason, harmonizing the data provided from different source increase the interoperability of the data creating a solid structure to understand the data, independently of the technology that the providers use to store their data.

Currently, there are two main types of databases, SQL and NoSQL, and a considerable number of variants for each type. In addition, the main differences between them are that SQL uses relational databases, a structured query language, it can be vertically scalable, and it could represent the data as a table composed of rows and columns. Although NoSQL is a non-relational database whose data is unstructured, it can be scalable horizontally, and they feature many ways to represent the data, such as pairs of key-values, documents, graphs, or wide-column stores.

1.1 Project's Background

The background of this project is related to the project BIGG (Building Information aGGregation, harmonisation and analytics platform) launched in December 2020. It aims to demonstrate the application of big data technologies and data analytic techniques in the complete building life-cycle, in more than 4000 buildings in 6 large-scale pilots, 3 in Spain (Catalonia) and 3 in Greece.

BIGG is firmly grounded in reality and will start from a set of 6 high-level business cases, spanning a broad range of use cases and distinct international settings. A bottom-up business-case based methodology will be followed to define a widely applicable reference architecture. Each case will be demonstrated through its real-life pilots, building on the common architecture and toolbox.[1]

1.2 Objective & Purposes

Usually, before harmonising the data, we follow rigorous steps related to gathering, cleaning, transforming and loading the data. For this reason, the motivation lies in developing a tool that makes effortless the task of mapping the data provided by different data sources using an ontology to straightforward the harmonisation process.

Additionally, BIGG project has six high-level business cases related to big data and analytic techniques. The emphasis of this project is placed on the harmonisation of the data related to these use cases. This project focuses on developing a tool that receives input data in tabular form and uses an ontology to harmonise and integrate it. The ontology is related to buildings, building devices and their energy consumption. Although the project focuses on solving the BIGG use cases, the tool aims to be generic and usable with any ontology previously defined. Finally, we want to create a user-friendly tool that allows users to make the mapping without requiring any knowledge about the ontology being used.

2 Document Structure

The rest of the document is organized as follows:

- **State of the Art**: in this part, we collect all the ideas, features, definitions, and technologies that help us to decide the development path that we will follow.
- **Application Design**: this section describes the architecture used in the project, such as the tech stack and the main libraries.
- **Development**: this corresponds to the project's core. It contains all the factors involved during the development process: methodology, planning, budget, requirements, implementations, and results.
- **Conclusions**: this part includes my honest impressions and opinion about the project.
- **Future Work**: Finally, in this section, we can find the future improvements that are envisioned for the project.

3 Application Design

In this section, I will explain the design of the proposed solution for mapping the data to harmonised model.

The solution is a functional web application developed using Flask as backend, React as frontend, MongoDB as database, and docker images to transform the outcome files. The application aims to map a dataset using an ontology and generate a YARRRML file containing a human-readable representation of our Linked Data mapping. In addition, we may use the output generated by the platform to transform the configuration files to RDF. The first transformation will be YARRRML to RML or R2RML using *rmlio/yarrrml-parser* and the second will be RML to RDF using *rmlio/rmlmapper-java*.

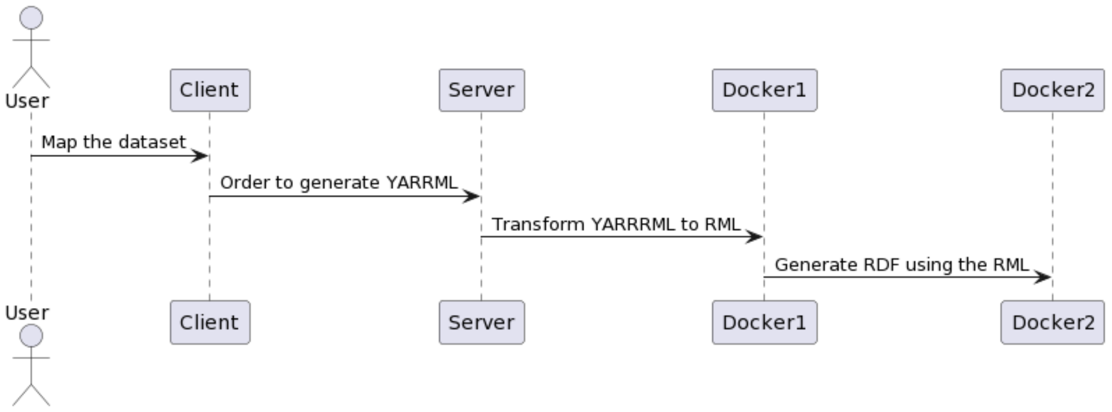


Figure 1: Mapping Workflow

After the mapping concludes, we will use a small workflow Figure 1. This workflow is composed of two docker images to transform the output file. It will be a human-readable configuration containing all the steps required to map the data; the format to structure this file is YARRRML.

The first part of the workflow is the docker image that transforms the YARRRML format to RML. Then, we process the output obtained to map the data and generate an RDF file. This last step is the one that allows the data to make sense of each other and to be able to perform SPARQL queries on it, also to be able to unify it with other data, making the concept of Linked Data a reality.

4 Development

4.1 Iteration 1: First contact

The objective of this first iteration has been to learn and deepen the technologies that I will use during the implementation. For this reason, I searched roadmaps that helped me gradually learn the technology while not getting frustrated during the process. In any case, I had previous experience with similar frameworks, such as Angular and Node.js, so it has not been a significant obstacle to learning these technologies.

In addition, to store the code and manage its versions, I use GitHub to control both projects.

4.1.1 Requirements

The requirements for this first iteration are divided into two parts: the presentation layer and the logic layer.

The first iteration is related to learning the technologies and integrating the different technologies described by the architecture. It means that I have to create both projects and integrate them, making that the backend can connect to the MongoDB database remotely and gather the data on the presentation layer.

The requirements proposed for this iteration are the following:

- Presentation Layer:
  - Create and structure a React project using TypeScript.
  - Add UI library to facilitate the designing part of the project.
- Logic Layer:
  - Create and structure a Flask project, including routers, functionalities, utilities, libraries, data validators, etc.
  - Connect Flask to MongoDB database.

4.1.2 Design

To create the presentation layer with their correspondent dependencies and configuration, I decided to use the command **npx create-react-app my-app –template typescript** that uses an existing template that provides a default React project configured with Typescript. After that, I will structure the project with different folders according to their aim. For instance: components, pages, asserts, utils, services, reducers or actions.

On the other hand, to create the logic layer, I create a new python environment with Flask and initialize the Flask server according to the official documentation. Also, I structure the code in different folders and files according to their objectives—for example, routes, utils, *database.py* or models.

4.1.3 Implementation

On the backend part, I installed all the dependencies required to connect our server to any MongoDB database and structure the endpoints using Blueprints. The reason for using Blueprints is simple; it makes our code clean and simple to understand. Also, I created a "*.env*" file to store private information such as secret keys, database connections, and environment variables.

On the other side, I installed Ant Design as the core of our UI and designed the routing part of the application using the library react-router. Finally, I made a "*.env*" 2 file to store the private variables.

```
FLASK_APP=app
FLASK_ENV=development

SECRET_KEY=<secret>
SERVER_HOSTNAME=127.0.0.1
SERVER_PORT=5000

JWT_SECRET_KEY=<secret>
JWT_ACCESS_TOKEN_EXPIRES=72
JWT_REFRESH_TOKEN_EXPIRES=30

MONGO_URI=<mongo_uri>
```

Figure 2: *.env* file

4.1.4 Results

The results of this iteration are scarce as most of the time has been spent researching and learning the technologies used. As a result 4b1a9b267256db435314670aaf57581ae3b0c7cf and 1acab5c8528f904469f383f21ad4405c5d2ed195 commits, we have two independent projects that are perfectly integrated and ready to start implementing the main functionalities of the project.

4.2 Iteration 2: User Management

In this second iteration, the objective is to practice all the knowledge acquired in the first iteration by implementing the user's management part of the application.

4.2.1 Requirements

All the requirements described below will be implemented in this iteration:

- Design the User Data Model.
- Endpoint to create users.
- Endpoint to get used by identifier.
- Endpoint to get all users.
- Endpoint to edit user by identifier.
- Endpoint to change the password by identifier.
- Endpoint to delete the user by identifier.
- Endpoint to authenticate a user using a JSON Web Token (JWT).
- Access control with a user role.

4.2.2 Design

To represent user information, I design a data model composed of different properties and the corresponding user interface components. Some of these properties are crucial for user functionalities, such as username, password, and role; others are only simple metadata such as first name and creation date. In addition, to validate the information processed through the "/user" endpoints in the backend API, I use pydantic to validate that data processed is appropriately defined based on the model.

Finally, to authenticate the user, I decided to implement an endpoint to validate the user and provide a **JSON Web Token** if it is appropriately authenticated.

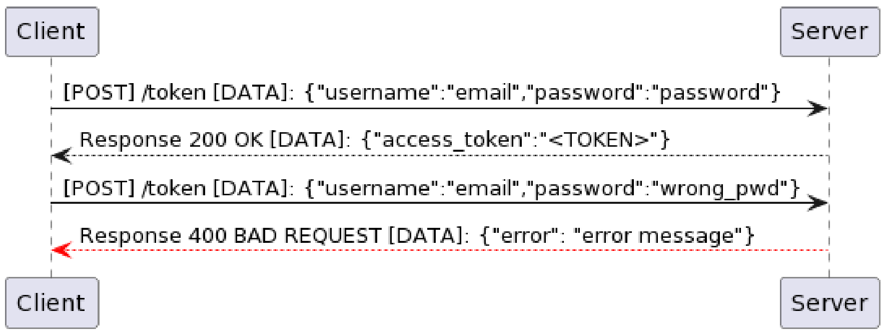

Client
Server
[POST] /token [DATA]: {"username":"email","password":"password"}
Response 200 OK [DATA]: {"access_token":"<TOKEN>"}
[POST] /token [DATA]: {"username":"email","password":"wrong_pwd"}
Response 400 BAD REQUEST [DATA]: {"error": "error message"}
Client
Server

Figure 3: Authentication Sequence

4.2.3 Implementation

The first step in this iteration was to design a data model representing the data as users. For this reason, I use the library *pydantic*, which allows us to validate the data and create instances based on the model. It will help us avoid incorrect data in our database and control the data correctly. For example, *pydantic* help us to check if the username used to create a new user is an email.

```
class UserModel(BaseModel):
    username: EmailStr
    password: str
    firstName: Optional[str]
    lastName: Optional[str]
    enable: bool = Field(default=True)
    roles: conlist(str, min_items=1) = Field(default=["User"])
    createdAt: datetime.datetime = Field(default=datetime.datetime.utcnow())
```

Figure 4: User Data Model Code

After defining the model that we use to represent the users, I registered a Blueprint called */users* that will contain all the **REST** endpoints. Also, I added two additional endpoints for authenticating the user and changing the password.

In addition, for all endpoints except create user, I check if the user is authenticated and its role. This validation aims to secure the application and prevent users from changing the information without consent.

Accordingly, in the presentation layer, I design the navigation bar that allows us to authenticate as the user. To store the authentication data, I use *react-cookies*. This library enables us to manage the cookies and store the authentication data. However, different solutions exist to store authentication data, such as session, context, or persistent Redux. Still, the option that I found more balanced according to efficiency and security was storing data into cookies.

4.2.4 Results

As a result of this iteration, I build a web that the users can authenticate, get their profile data and change its information using the visual interface. The presentation layer uses the logic layer to gather the data from the database, and the user can interact with it. However, the delete and get-all users functionalities are only accessible using a client such as Postman.

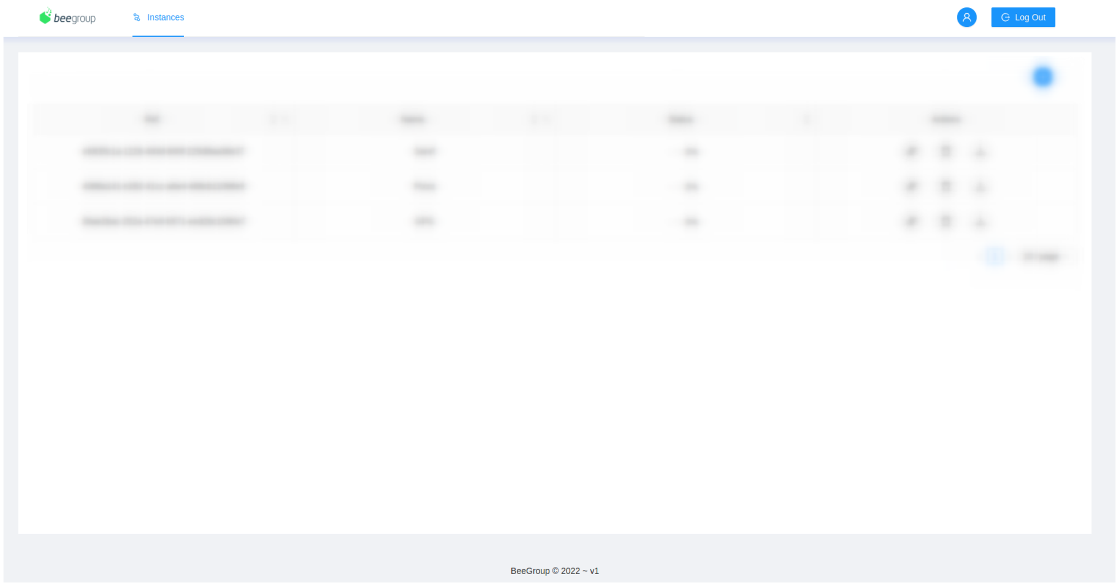
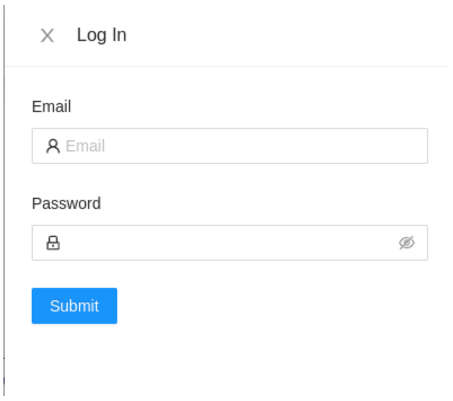
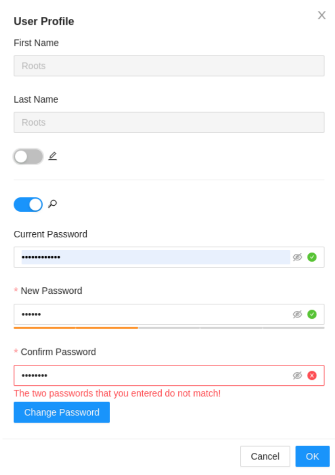


Figure 5: First version of the visual interface.



(a) Authentication Form



User Profile

First Name: Roots

Last Name: Roots

Current Password: ••••••••••••

* New Password: ••••••

* Confirm Password: ••••••••

The two passwords that you entered do not match!

(b) Edit User Visualization

4.3 Iteration 3: Ontology

During this iteration, I helped an office colleague implement the BIGG Ontology using the Web Ontology Language (OWL) from the existing BIGG UML diagrams. In addition to supervising and helping decide among the different ways to solve the obstacles that appeared, I explored how to integrate the ontology using Python with the backend.

4.3.1 Requirements

Most of the requirements of this iteration consist of designing and applying how I can use ontologies in the backend part.

- Supervise ontology designing.
- Design how to map data to the ontology.
- Create endpoints */ontology* to get ontology data.

4.3.2 Design

In this third iteration, I helped with the ontology design and researched more into ontology defining topics. While my colleague was developing the ontology, I also researched how I could map CSV files using the ontology. After different discussions about potential solutions, according to the papers read and the information obtained, I decided to take another approach to generate the mapped data. Instead of doing so directly from the developed application, the proposal is to generate a mapping file based on YARRRML. Then, this file can be parsed into RML or R2RML rules, which are standard languages understood by many tools that provide tabular data for RDF transformation. Both transformations, from YARRRML to RML/R2RML and from the tabular input data to RDF, are implemented using two docker containers that provide these functionalities.

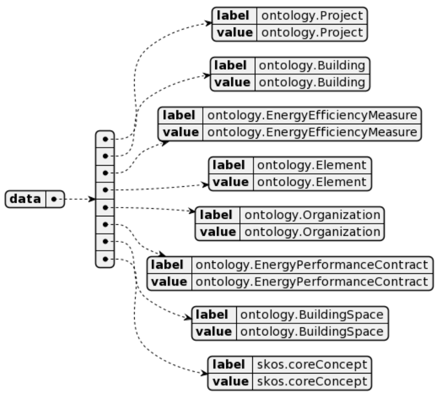
4.3.3 Implementation

Consequently, I found a reasonable solution to map the data using the ontology provided. I started to implement the endpoints */ontology* that the frontend will consume to get information related to the ontology. These endpoints will return the classes, relations, data and object properties. Also, I developed an endpoint to make queries using SPARQL.

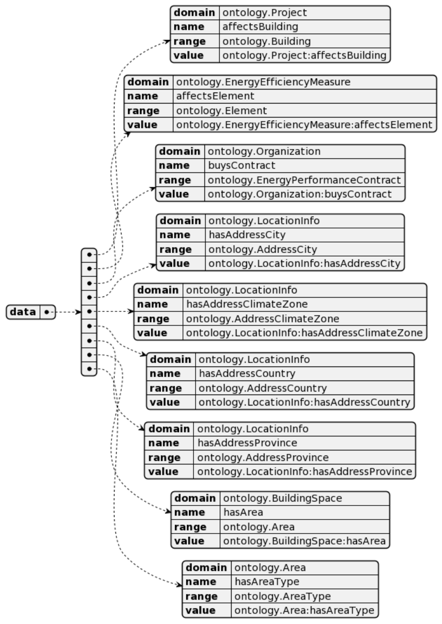
Additionally, I simulated the proposed solution workflow, verifying that it is feasible to implement the mapping from CSV to RDF once the mapping rules have been defined using YARRRML.

4.3.4 Results

The results of this part will be the endpoints that we can use to get the data from the ontology loaded on the server—for example, the classes, relationship between the classes, data properties, object properties.


data
label ontology.Project
value ontology.Project
label ontology.Building
value ontology.Building
label ontology.EnergyEfficiencyMeasure
value ontology.EnergyEfficiencyMeasure
label ontology.Element
value ontology.Element
label ontology.Organization
value ontology.Organization
label ontology.EnergyPerformanceContract
value ontology.EnergyPerformanceContract
label ontology.BuildingSpace
value ontology.BuildingSpace
label skos.coreConcept
value skos.coreConcept

(a) Response */ontology/classes*


data
domain ontology.Project
name affectsBuilding
range ontology.Building
value ontology.Project:affectsBuilding
domain ontology.EnergyEfficiencyMeasure
name affectsElement
range ontology.Element
value ontology.EnergyEfficiencyMeasure:affectsElement
domain ontology.Organization
name buysContract
range ontology.EnergyPerformanceContract
value ontology.Organization:buysContract
domain ontology.LocationInfo
name hasAddressCity
range ontology.AddressCity
value ontology.LocationInfo:hasAddressCity
domain ontology.LocationInfo
name hasAddressClimateZone
range ontology.AddressClimateZone
value ontology.LocationInfo:hasAddressClimateZone
domain ontology.LocationInfo
name hasAddressCountry
range ontology.AddressCountry
value ontology.LocationInfo:hasAddressCountry
domain ontology.LocationInfo
name hasAddressProvince
range ontology.AddressProvince
value ontology.LocationInfo:hasAddressProvince
domain ontology.BuildingSpace
name hasArea
range ontology.Area
value ontology.BuildingSpace:hasArea
domain ontology.Area
name hasAreaType
range ontology.AreaType
value ontology.Area:hasAreaType

(b) Response */ontology/properties/:property_type*

4.4 Iteration 4: Mapping Classes

According to this fourth iteration, the primary purpose is to break down the complexity of the problem into minor problems (requirements) because I can't assume to develop the functionalities thinking in general. After all, there are multiple exceptions to have in mind. For this reason, this iteration is focused on map classes without their relations and generates the YARRRML file.

4.4.1 Requirements

The requirements of this iteration are related to how I can map the data sets using the ontology and generate a mapping file.

- Visual interface to map the data provided.
- Visual interface to manage mapping instances.
- Design data model to store the data mapped.
- Implement REST endpoints of the data model.
- Generate YARRRML file using the data mapped.

4.4.2 Design

To store and retrieve files, I use GridFS, which keeps files in two collections: *chunks* stores the binary chunks and *files* stores the file's metadata.

Otherwise, I design the model *mapping* that contains all the data related to the mapping process, where some properties of this model are flexible according to the ontology used and the values mapped. Notably, in this code version, the ontology must be static.

```
class InstanceModel(BaseModel):
    ref: str
    name: str
    description: Optional[constr(max_length=280)]
    filenames: conlist(str, min_items=1)
    createdAt: datetime.datetime
    createdBy: EmailStr
    status: int = Field(default=0)
    mapping: dict = Field(default={})
    relations: dict = Field(default={})
    classes_to_map: list = Field(default=[])
    current_ontology: str
```

Figure 8: Mapping Data Model Code

Apart from that, I designed a method to generate the YARRRML file automatically using the data model presented before. But unfortunately, I didn't find any library to create this kind of file. For this reason, I had to analyze the common patterns that We can find using YARRRML and create the different methods to generate them.

```
s = ""
s += add_prefixes()

s += init_mappings()

s += add_mapping("building")
s += init_sources()
s += add_source("building.csv")
```

Figure 9: Generate YARRRML Example

4.4.3 Implementation

After defining the data model, I started working on the **REST** endpoints and the correspondents form on the presentation layer. Behind that, I developed the method to map the data; it was the most complex functionality until now because I had to think of a precise and optimal way to store the data mapped and the workflow that the user will use. Furthermore, I need to define an endpoint to initialize the mapping instance according to the ontology that the user would like to map; this endpoint initializes the instances by inserting classes and the relations that they may have.

In order to implement the endpoints, I created a new Blueprint named */instances* that contains the endpoints related to mapping instances. For instance, create, get, list and initialize mapping instance.

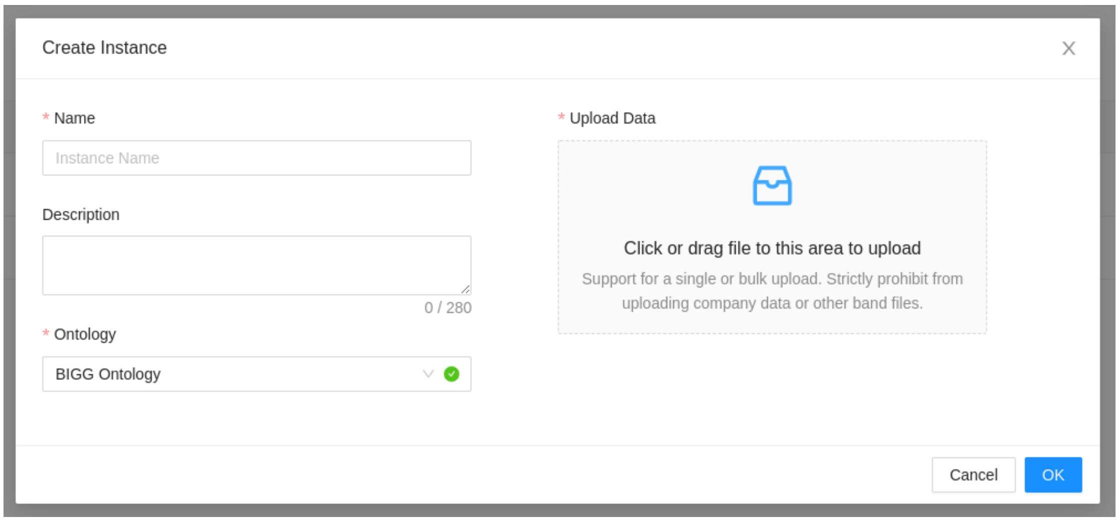
Subsequently, I created a new route into the frontend named */instances*; this route contains all the pages related to mapping our data, such as the list of our mapping instances.

According to the mapping process, I decide to make a progressive process. It means that the user can map their data relative to their needs. For this reason, mapping our data is performed by the following steps:

1. Complete the form to create a mapping instance.
2. Access to the detail of the mapping instance.
3. Select the classes that the user would like to map.
4. Select the class to map.
5. Set a subject and map the columns related to the ontology.
6. Save the form completed.
7. Generate YARRRML.

4.4.4 Results

In this iteration, the application has the first harmonizing and mapping data functionality. It suggests that the application begins to have sense in this step and follows the project's objectives.


Create Instance

* Name

Instance Name

Description

0 / 280

* Ontology

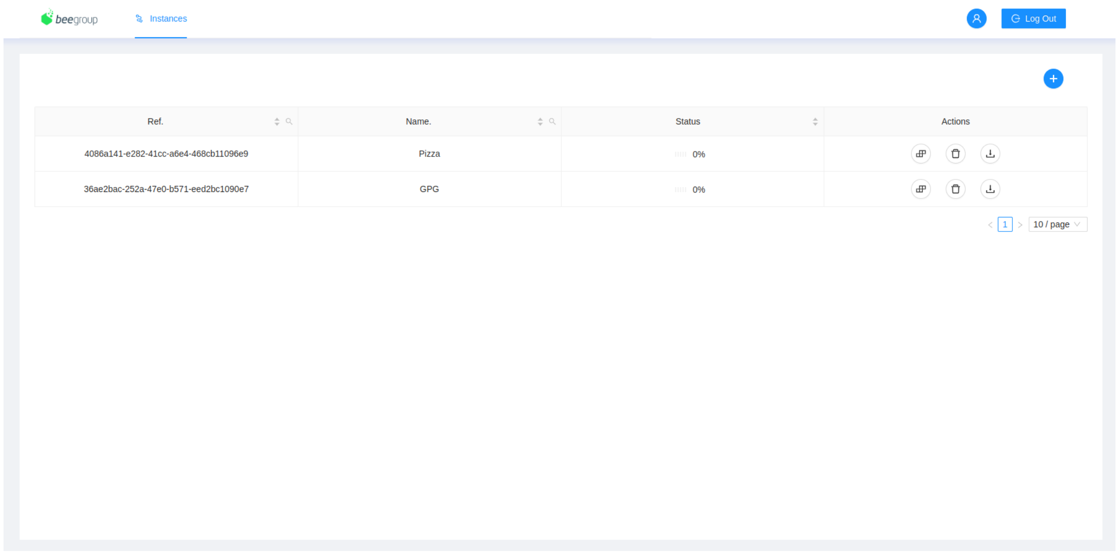
BIGG Ontology

* Upload Data

Click or drag file to this area to upload

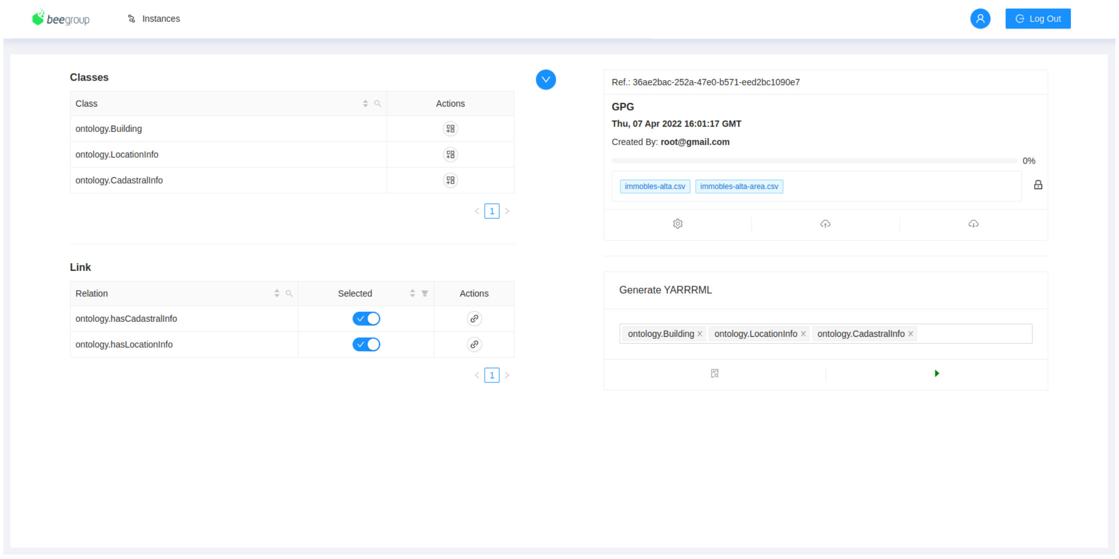
Support for a single or bulk upload. Strictly prohibit from uploading company data or other band files.

Figure 10: Create Mapping Form



Ref.	Name.	Status	Actions
4086a141-e282-41cc-a6e4-468cb11096e9	Pizza	0%	
36ae2bac-252a-47e0-b571-eed2bc1090e7	GPG	0%	

Figure 11: List Mappings



Classes

Class	Actions
ontology.Building	
ontology.LocationInfo	
ontology.CadastralInfo	

Link

Relation	Selected	Actions
ontology.hasCadastralInfo	✓	
ontology.hasLocationInfo	✓	

Ref.: 36ae2bac-252a-47e0-b571-eed2bc1090e7

GPG

Thu, 07 Apr 2022 16:01:17 GMT

Created By: root@gmail.com

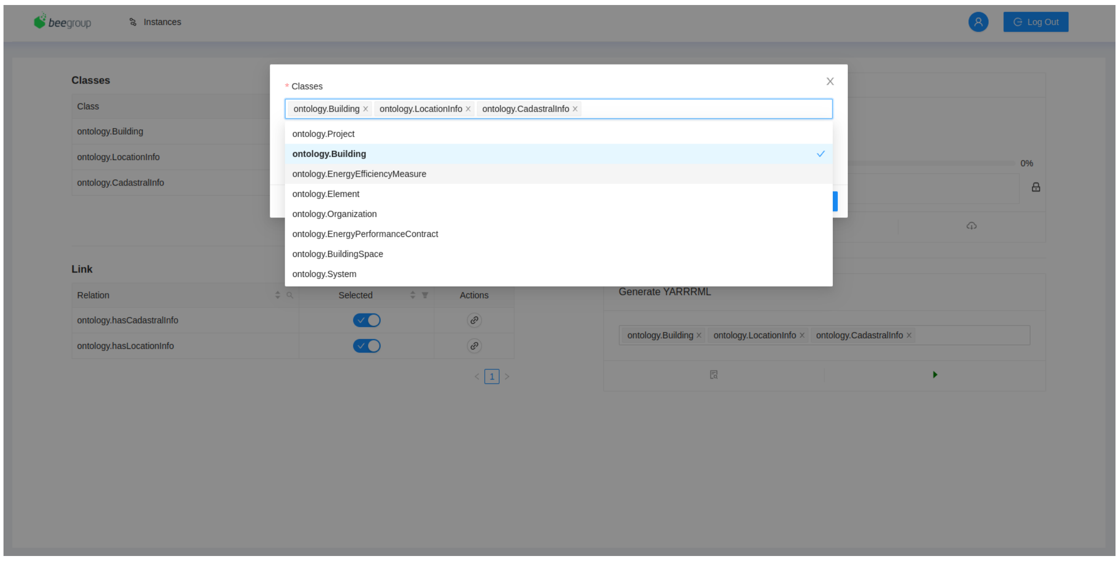
0%

immobles-alta.csv immobles-alta-area.csv

Generate YARRRML

ontology.Building × ontology.LocationInfo × ontology.CadastralInfo ×

Figure 12: Detail Mapping

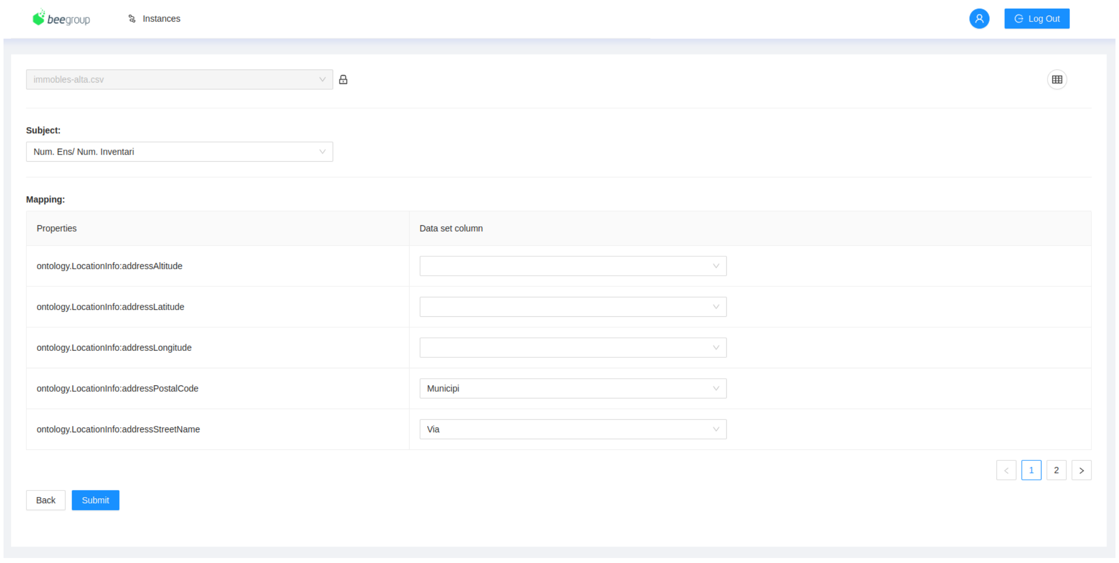


* Classes

ontology.Building × ontology.LocationInfo × ontology.CadastralInfo ×

ontology.Project
ontology.Building ✓
ontology.EnergyEfficiencyMeasure
ontology.Element
ontology.Organization
ontology.EnergyPerformanceContract
ontology.BuildingSpace
ontology.System

Figure 13: Select Classes to Map



immobles-alta.csv

Subject:

Num. Ens/ Num. Inventari

Mapping:

Properties	Data set column
ontology.LocationInfo:addressAltitude	
ontology.LocationInfo:addressLatitude	
ontology.LocationInfo:addressLongitude	
ontology.LocationInfo:addressPostalCode	Municipi
ontology.LocationInfo:addressStreetName	Via

Figure 14: Mapping Data set

4.5 Iteration 5: Mapping Relations

This fifth iteration aims to relate the classes to obtain coherence and cohesion between the mapped data. Hence, the system will only make available the relations between the classes mapped and consider the cardinality between the relationships. For instance, the most simple case of cardinality between classes is when the user has to relate Buildings and Cadastral References, one building may have n cadastral references.

4.5.1 Requirements

The requirements are related to making possible the relationship between the mapped data and guaranteeing the correct cardinality:

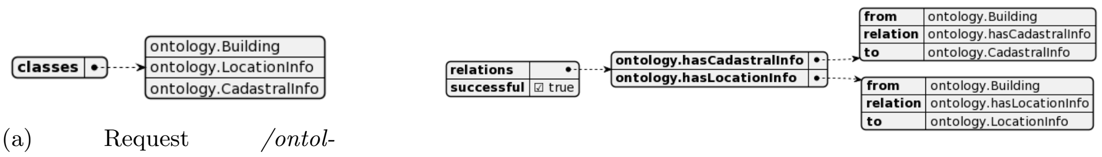
- Show the available relations of the current instances.
- Map relations.
- Take care to guarantee the cardinality between the classes.
- Show graphical preview of the results.

4.5.2 Design

This iteration did not require a designing phase.

4.5.3 Implementation

First of all, I implemented the endpoint that provides the available relations between the classes mapped; it receives as an input a list of classes selected and returns as output the relationships defined in the ontology. After that, I developed the code that makes it possible to visualize the endpoint in the presentation layer and choose the relations that the user would like to use during the mapping process.



(a) Request */ontology/classes/relations*

(b) Response */ontology/classes/relations*

Figure 15: Endpoint */ontology/classes/relations*

When mapping relations, a problem arises when the user wants to map data about different entities related to a cardinality distinct from 1 to 1. For example, the problem could appear when a building has multiple cadastral references. In order to address it, the idea is to follow a similar approach to that used in relational databases. A column with unique values in one of the input CSV files is also used as a foreign key from the related files, so it is possible to join them.

In some cases, the input CSV files do already follow this approach. However, if it is not the case, it is highly recommended to use a transformation software such as *OpenRefine* to restructure the data emphasizing the columns acting as identifiers and dealing with those including more than one value per column to normalize them. For example, if we have the following dataset, we must have the subsequent transformation to map the relations correctly.

ID	Postal Code	Cadastral Ref.
8	08028	6412411DF2861C0093GK;6412411DF2861C0091DH;6412412DF2861C0148BF
11	08370	5693335DF0959S0001LS

Table 1: Original Data

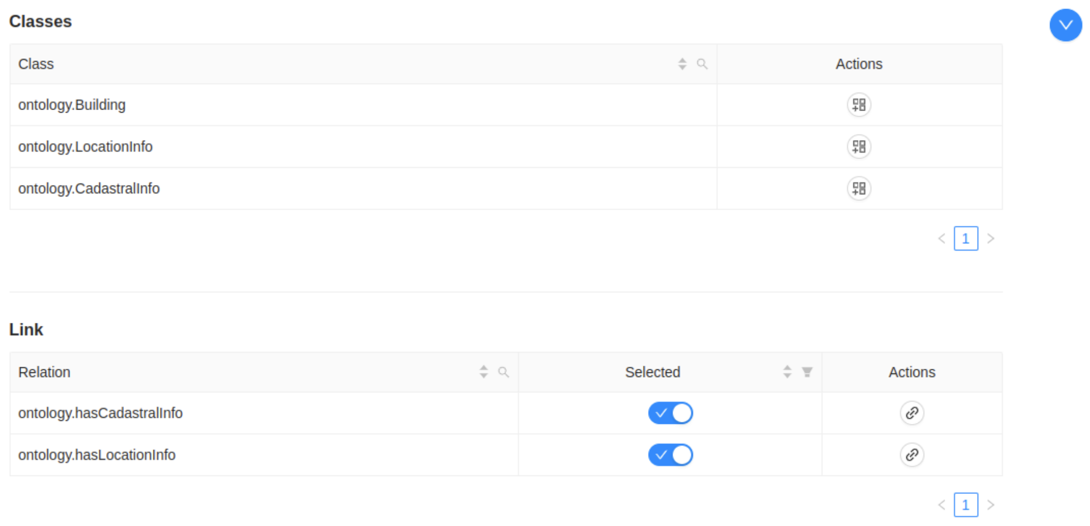
ID	Postal Code	Cadastral Ref.
8	08028	6412411DF2861C0093GK
8	08028	6412411DF2861C0091DH
8	08028	6412412DF2861C0148BF
11	08370	5693335DF0959S0001LS

Table 2: Transformed data

Finally, to show the data mapped, I developed a react component where the user can see the classes mapped and the relations used. To develop this part, I used a library named *react-flow*; it is a highly customizable React component for building node-based editors and interactive diagrams.

4.5.4 Results

As a result, I had functional software that allows users to generate YARRRML based on previously defined mappings. At this point, the fundamental functionalities have already been developed, and future iterations will focus on improvements or fixing bugs.



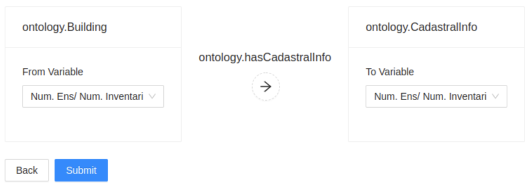
Classes

Class	Actions
ontology.Building	
ontology.LocationInfo	
ontology.CadastralInfo	

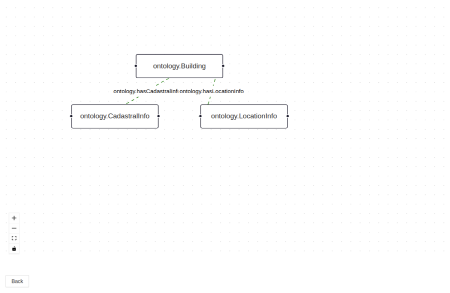
Link

Relation	Selected	Actions
ontology.hasCadastralInfo	✓	
ontology.hasLocationInfo	✓	

Figure 16: Available relations according to the selected classes

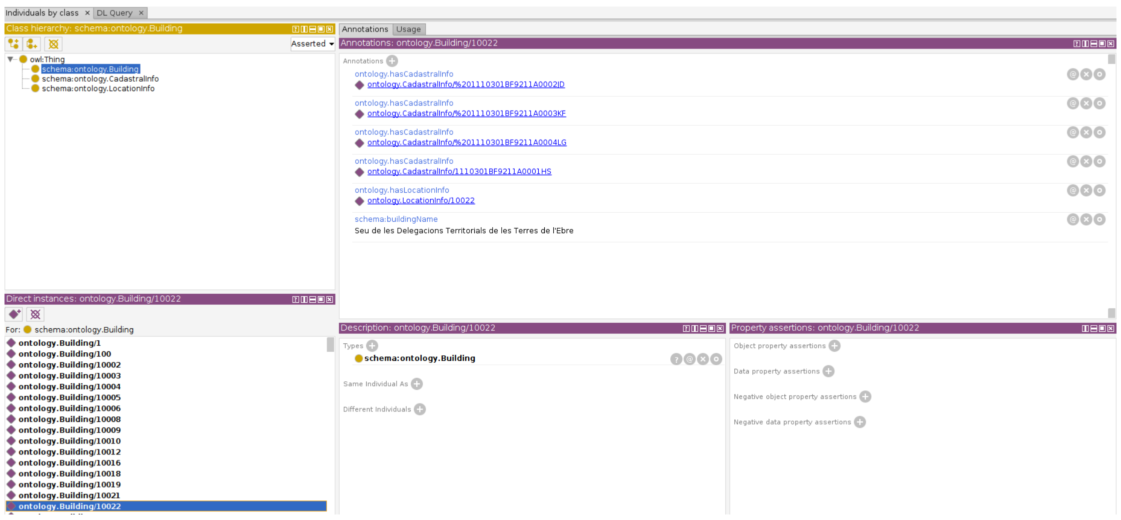


(a) Mapping relation *hasCadastralInfo*



(b) Preview mapping

Figure 17: Mapping relations



Class hierarchy: schema:ontology.Building

owl:Thing
- schema:ontology.Building
- schema:ontology.CadastralInfo
- schema:ontology.LocationInfo

Annotations: ontology.Building/10022

Annotations

ontology.hasCadastralInfo
ontology.CadastralInfo/%201110301BF9211A0002JD

ontology.hasCadastralInfo
ontology.CadastralInfo/%201110301BF9211A0003KF

ontology.hasCadastralInfo
ontology.CadastralInfo/%201110301BF9211A0004LG

ontology.hasCadastralInfo
ontology.CadastralInfo/1110301BF9211A0001HS

ontology.hasLocationInfo
ontology.LocationInfo/10022

schema:buildingName
Seu de les Delegacions Territorials de les Terres de l'Ebre

Direct instances: ontology.Building/10022

For: schema:ontology.Building

ontology.Building/1
ontology.Building/100
ontology.Building/10002
ontology.Building/10003
ontology.Building/10004
ontology.Building/10005
ontology.Building/10006
ontology.Building/10008
ontology.Building/10009
ontology.Building/10010
ontology.Building/10012
ontology.Building/10016
ontology.Building/10018
ontology.Building/10019
ontology.Building/10021
ontology.Building/10022

Description: ontology.Building/10022

Types
schema:ontology.Building

Same Individual As

Different Individuals

Property assertions: ontology.Building/10022

Object property assertions

Data property assertions

Negative object property assertions

Negative data property assertions

Figure 18: Visualize results on *Protégé*

4.6 Iteration 6: Multiple Ontologies

In this sixth iteration, the aim will be to isolate the ontology and the platform, making possible the use of multiple ontologies to map our data. Likewise, this part requires adapting the current code and data models to this new improvement.

4.6.1 Requirements

The requirements for this iteration will be the following:

- Design data model to represents Ontology.
- Ontology REST endpoints.
- Visual interface to manage ontologies.
- Isolate ontology and software.

4.6.2 Design

In order to avoid hardcoding the mapping to a specific ontology, I created a new data model named *Ontology* that allows us to store the metadata such as name, description, and visibility from the ontology and simultaneously save the ontology as a file using GridFS. Also, I adapted the instance data model, adding a new property to identify the ontology used by each mapping.

```
class VisibilityEnum(str, Enum):
    public = 'public'
    private = 'private'


class OntologyModel(BaseModel):
    filename: str
    file_id: str
    description: Optional[str]
    ontology_name: str
    createdAt: datetime.datetime
    createdBy: EmailStr
    visibility: VisibilityEnum
```

Figure 19: Ontology Data Model

4.6.3 Implementation

For the purpose of managing the ontologies, I adapted the methods created on*/ontologies* to */ontology/:id/* and I added the REST methods. The reason for this change was simple, the endpoints were created using a single ontology, and until now, we did not need to specify the *id*.

After defining the data model, I decided to find a way to isolate the ontologies. So, I read the documentation of the library *owl2ready* in search of a possible solution. Finally, I found an object named *World* that allows us to handle several Worlds in parallel. It is interesting if you want to load several versions of the same ontology, for example, before and after reasoning.

For some unknown reason, I have to create a world on the fly when the user uses the ontology's endpoint. So, I make a function that returns an ontology instance with the ontology uploaded on *fs.chuncks*.

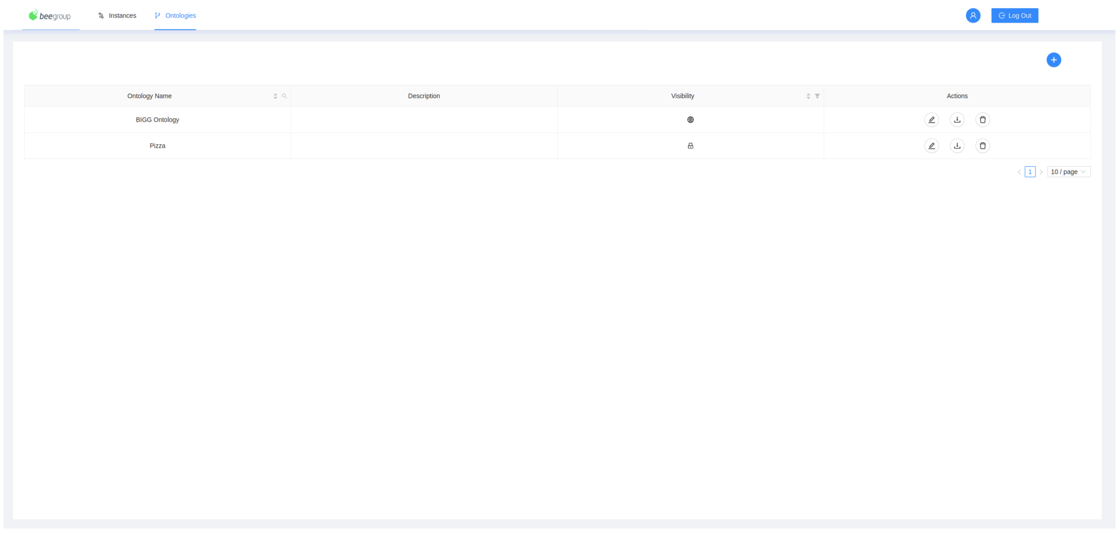
```
def define_ontology(ontology_id):
    # https://owlready2.readthedocs.io/en/latest/world.html

    ontology_record = mongo.db.ontologies.find_one({"_id": ObjectId(ontology_id)})
    ontology_file = get_file(ontology_record['file_id'])
    ontology_instance = World()

    with tempfile.NamedTemporaryFile(dir='output', mode='w', suffix='.owl') as file:
        file.write(ontology_file.getvalue())
        ontology_instance.get_ontology(file.name).load()
    return ontology_instance
```

Figure 20: Define Ontology function

Finally, I developed the presentation layer to manage the ontologies.

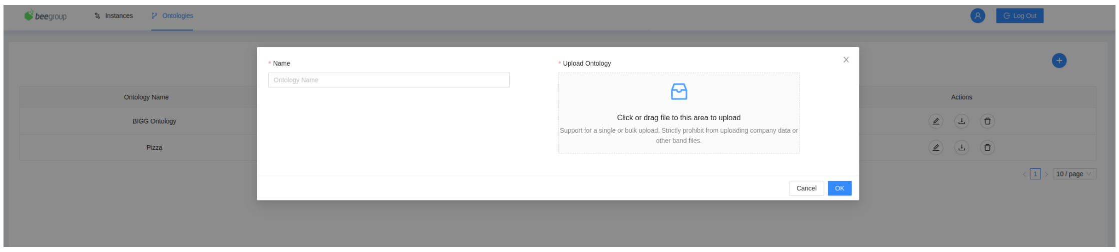
4.6.4 Results

The results of this iteration are the following improvements: manage and make it possible to use map data using different ontologies.



Ontology Name	Description	Visibility	Actions
BIGG Ontology			
Pizza			

Figure 21: List of ontologies



* Name

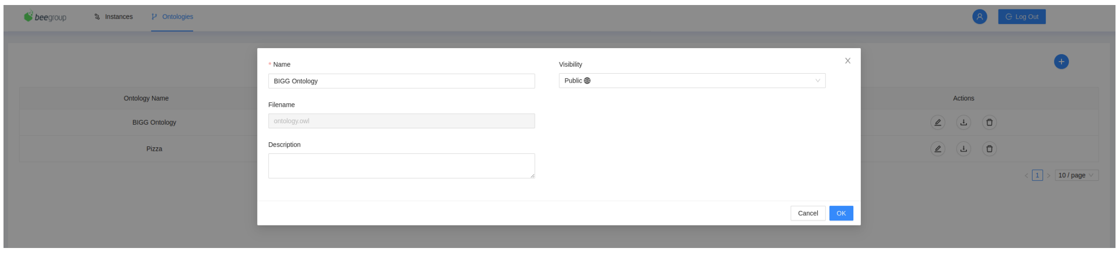
Ontology Name

* Upload Ontology

Click or drag file to this area to upload

Support for a single or bulk upload. Strictly prohibit from uploading company data or other band files.

Figure 22: Create ontology



* Name: BIGG Ontology

Visibility: Public

Filename: ontology.owl

Description:

Figure 23: Edit ontology

5 Deployment

This section has the aim of explaining the process followed to deploy it. In addition, this section is separated from the general explanation of each development process to explain it in depth.

In software terms, when I talk about the deployment as the process that includes all the steps and activities required to make available the software to the users, for instance, on a server or device.

According to the previous definition, we have to deploy the three layers described in the architecture section to make possible the integration among the layers and get for complete functionality of the project.

In the beginning, I decided to deploy the different layers into hosting services and databases providers that have free plans such as Heroku, MongoDB Atlas and Render. However, after different tries, we found that the free plans are not enough to deploy our presentation layer because they require more resources than those available in their free versions. Therefore, it forced us to search for a different alternative to complete the deployment. So, finally, I use the company resources to deploy this project: a Kubernetes Cluster and a MongoDB hosted in a private server.

The deployment part is simple. We only need to build the backend and the frontend as docker images and generate the proper configuration to serve the images from the Kubernetes Cluster. They will also use a *Load Balancer* to assign a public IP and make it publicly accessible. It is important to emphasize that this deployment is only for development and could be insecure in a production environment.

In addition to automatizing the process of building the docker images when the iteration ends, I create a Github Action that builds and push the images automatically to the docker registry. This process runs automatically when a Pull Request from the development into the main branch is approved.

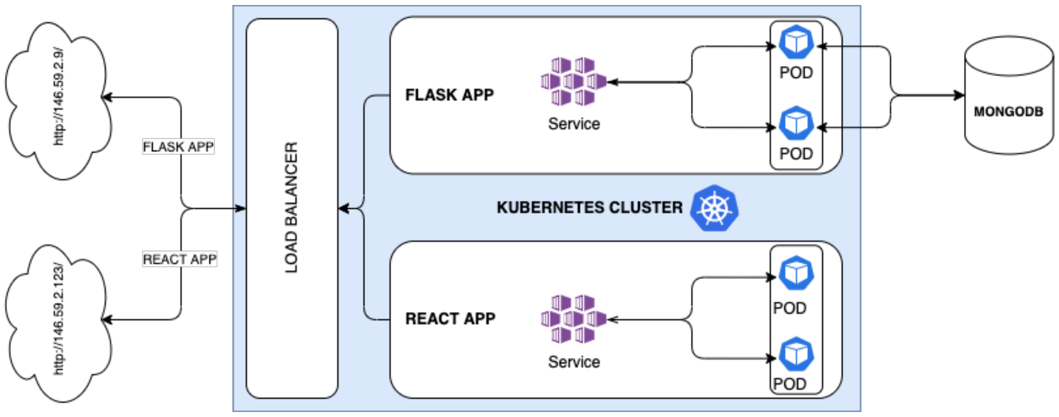


Figure 24: Deployment Diagram

6 Conclusions

The project's conclusion could be divided into different points: The positive and negative aspects and its development impact.

The positive aspects of having developed this project during my academic training have improved my knowledge as a software engineer and especially discovered a new world that most people do not know, but nowadays is extremely useful in data. At the same time, it helped me understand that the development of the web is not trivial, especially in the research field.

I started with almost null knowledge about the topic as a negative point. Therefore, I required a long period of researching it to develop a valuable and versatile solution that reduces the complexity of mapping data.

Likewise, developing this tool was complex because until I found that the solution would be to generate a configuration file that they would use to generate their data, I was focused on developing some way to generalise the form to map the data.

Finally, developing this application using ontologies, linked data, and web semantics increases the possibility of using it in future projects or making it a popular technology.

7 Future Work

After having finalized most of the functionalities and features of the software, there are still requirements to satisfy and develop for future work or upgrades.

In future work, it would be highly recommendable the accomplishment the following functionalities:

- Find and solve errors.
- Make the functionality that allows the user to use multiple ontologies in a single mapping.
- Develop End-to-End Test.
- Develop Unit Testing.
- Develop a process to convert YARRRML automatically to RDF.
- Use URL to create ontologies.
- Make it possible to join multiple ontologies.
- Improve the security of the project.
- Develop functionality to track the percentage of mapping.
- Validate mapping before creating a configuration file.
- Develop functionalities to transform datasets on the fly.
- Make the application more responsive, especially on smartphones.
- Make groups of users share the ontologies and mappings.
- Release the beta.

References

[1] Clean & Energy Efficiency European Union's Horizon 2020 Secure. *Building Information aGGregation, harmonization and analytics platform.* URL: https://cordis.europa.eu/project/id/957047. (accessed: 20.12.2021).

Appendices

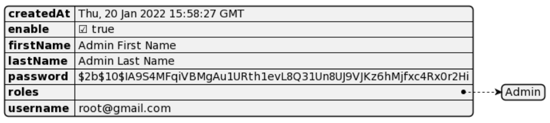
A Project Repositories

- Client
- Server

B Vocabulary

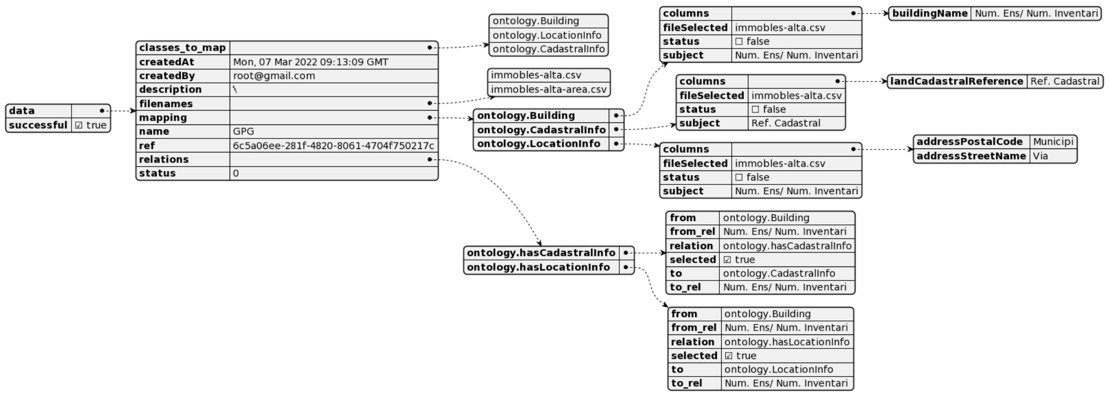
1. **Building**: Anything built on an area of land, having a roof and walls and usually intended to be kept in one place.
2. **Location**: Defines a place.
3. **Area**: a part of a space or a surface.
4. **Person**: the actual self or individual personality of a human being.
5. **Energy Performance Certificate**: are a rating scheme to summarise the energy efficiency of buildings.
6. **Organization**: a group of persons grouped or organized for some purpose or work; an association.
7. **Cadastral**: of or pertaining to a cadastre.
8. **Energy Efficiency Measure**: it refers to the measure that determines the less energy needed to provide an energy service.
9. **Building Space**: limited and defined space inside a building, such as waiting room, main hall etc.
10. **Device**: a thing made for a particular purpose, esp. a mechanical or electric invention.
11. **Measurement**: the number representing the extent, size, etc...
12. **Weather Station**: installation equipped and used for meteorological observation.

C API Data Model



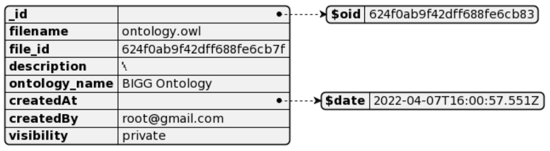
createdAt Thu, 20 Jan 2022 15:58:27 GMT
enable ☑ true
firstName Admin First Name
lastName Admin Last Name
password $2b$10$IA9S4MFqiVBMgAu1URth1evL8Q31Un8UJ9VJKz6hMjfxc4Rx0r2Hi
roles
Admin
username root@gmail.com

Figure 25: User Example



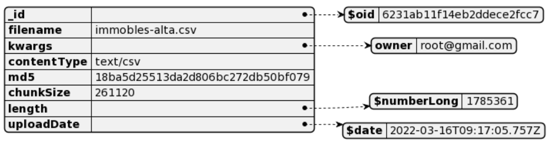
data
successful ☑ true
classes_to_map
createdAt Mon, 07 Mar 2022 09:13:09 GMT
createdBy root@gmail.com
description \
filenames
mapping
name GPG
ref 6c5a06ee-281f-4820-8061-4704f750217c
relations
status 0
ontology.Building
ontology.LocationInfo
ontology.CadastralInfo
immobles-alta.csv
immobles-alta-area.csv
ontology.Building
ontology.CadastralInfo
ontology.LocationInfo
columns
fileSelected immobles-alta.csv
status ☐ false
subject Num. Ens/ Num. Inventari
buildingName Num. Ens/ Num. Inventari
columns
fileSelected immobles-alta.csv
status ☐ false
subject Ref. Cadastral
landCadastralReference Ref. Cadastral
columns
fileSelected immobles-alta.csv
status ☐ false
subject Num. Ens/ Num. Inventari
addressPostalCode Municipi
addressStreetName Via
ontology.hasCadastralInfo
ontology.hasLocationInfo
from ontology.Building
from_rel Num. Ens/ Num. Inventari
relation ontology.hasCadastralInfo
selected ☑ true
to ontology.CadastralInfo
to_rel Num. Ens/ Num. Inventari
from ontology.Building
from_rel Num. Ens/ Num. Inventari
relation ontology.hasLocationInfo
selected ☑ true
to ontology.LocationInfo
to_rel Num. Ens/ Num. Inventari

Figure 26: Instance Simplified Example



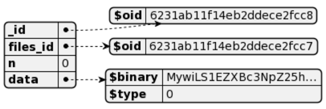
_id
$oid 624f0ab9f42dff688fe6cb83
filename ontology.owl
file_id 624f0ab9f42dff688fe6cb7f
description \
ontology_name BIGG Ontology
createdAt
$date 2022-04-07T16:00:57.551Z
createdBy root@gmail.com
visibility private

Figure 27: Ontology Example



_id
$oid 6231ab11f14eb2ddece2fcc7
filename immobles-alta.csv
kwargs
owner root@gmail.com
contentType text/csv
md5 18ba5d25513da2d806bc272db50bf079
chunkSize 261120
length
$numberLong 1785361
uploadDate
$date 2022-03-16T09:17:05.757Z

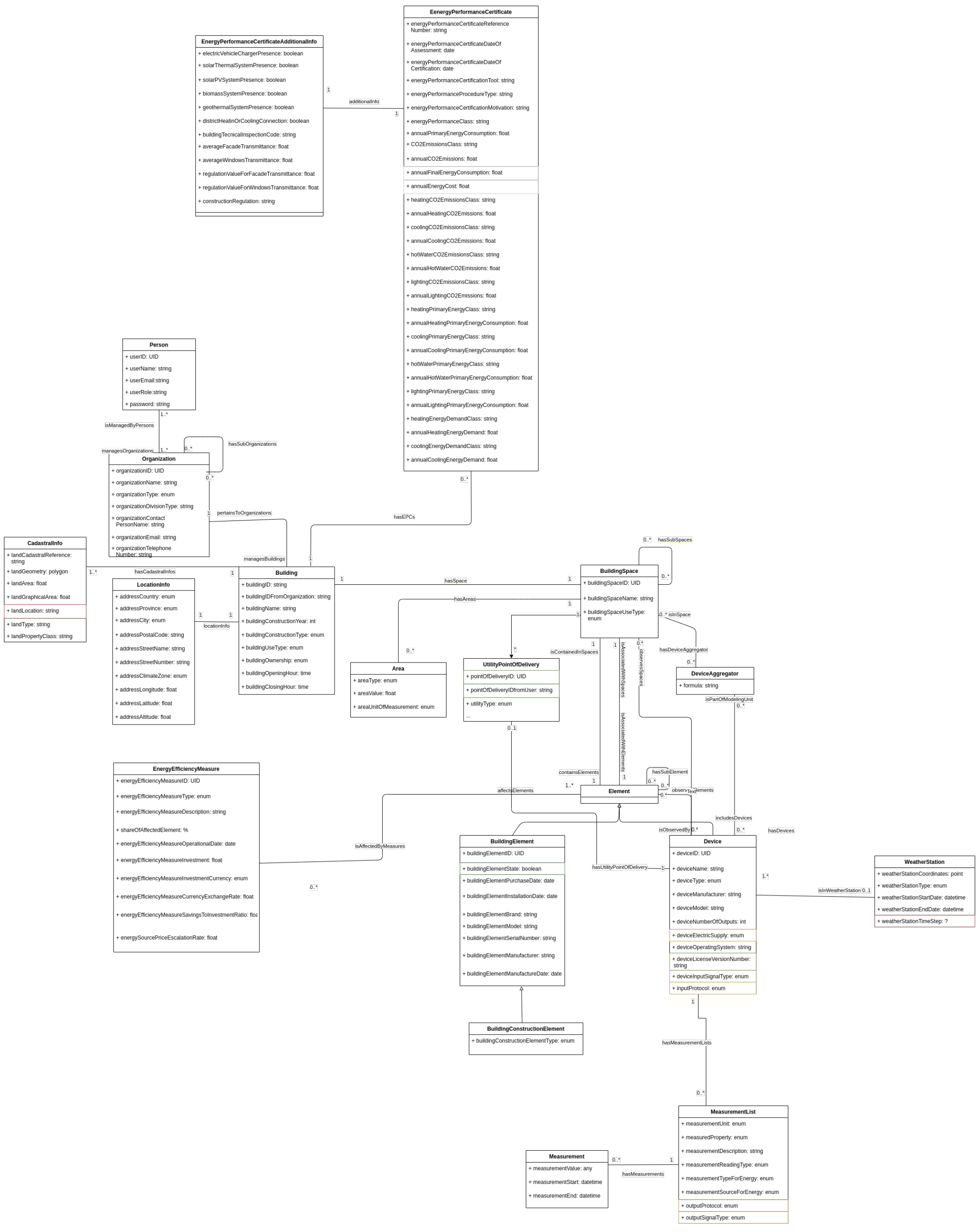
Figure 28: Fs.file Example



_id
$oid 6231ab11f14eb2ddece2fcc8
files_id
$oid 6231ab11f14eb2ddece2fcc7
n 0
data
$binary MywiLS1EZXBc3NpZ25h...
$type 0

Figure 29: Fs.chunk Example

D BIGG - Data Model



EnergyPerformanceCertificateAdditionalInfo
+ electricVehicleChargerPresence: boolean
+ solarThermalSystemPresence: boolean
+ solarPVSystemPresence: boolean
+ biomassSystemPresence: boolean
+ geothermalSystemPresence: boolean
+ districtHeatinOrCoolingConnection: boolean
+ buildingTecnicalInspectionCode: string
+ averageFacadeTransmittance: float
+ averageWindowsTransmittance: float
+ regulationValueForFacadeTransmittance: float
+ regulationValueForWindowsTransmittance: float
+ constructionRegulation: string
additionalInfo
1
1
EenergyPerformanceCertificate
+ energyPerformanceCertificateReference Number: string
+ energyPerformanceCertificateDateOf Assessment: date
+ energyPerformanceCertificateDateOf Certification: date
+ energyPerformanceCertificationTool: string
+ energyPerformanceProcedureType: string
+ energyPerformanceCertificationMotivation: string
+ energyPerformanceClass: string
+ annualPrimaryEnergyConsumption: float
+ CO2EmissionsClass: string
+ annualCO2Emissions: float
+ annualFinalEnergyConsumption: float
+ annualEnergyCost: float
+ heatingCO2EmissionsClass: string
+ annualHeatingCO2Emissions: float
+ coolingCO2EmissionsClass: string
+ annualCoolingCO2Emissions: float
+ hotWaterCO2EmissionsClass: string
+ annualHotWaterCO2Emissions: float
+ lightingCO2EmissionsClass: string
+ annualLightingCO2Emissions: float
+ heatingPrimaryEnergyClass: string
+ annualHeatingPrimaryEnergyConsumption: float
+ coolingPrimaryEnergyClass: string
+ annualCoolingPrimaryEnergyConsumption: float
+ hotWaterPrimaryEnergyClass: string
+ annualHotWaterPrimaryEnergyConsumption: float
+ lightingPrimaryEnergyClass: string
+ annualLightingPrimaryEnergyConsumption: float
+ heatingEnergyDemandClass: string
+ annualHeatingEnergyDemand: float
+ coolingEnergyDemandClass: string
+ annualCoolingEnergyDemand: float
0..*
hasEPCs
Person
+ userID: UID
+ userName: string
+ userEmail:string
+ userRole:string
+ password: string
1..*
isManagedByPersons
managesOrganizations
1..*
0..*
hasSubOrganizations
Organization
+ organizationID: UID
+ organizationName: string
+ organizationType: enum
+ organizationDivisionType: string
+ organizationContact PersonName: string
+ organizationEmail: string
+ organizationTelephone Number: string
0..*
1
pertainsToOrganizations
managesBuildings
1
CadastralInfo
+ landCadastralReference: string
+ landGeometry: polygon
+ landArea: float
+ landGraphicalArea: float
+ landLocation: string
+ landType: string
+ landPropertyClass: string
1..*
hasCadastralInfos
1
LocationInfo
+ addressCountry: enum
+ addressProvince: enum
+ addressCity: enum
+ addressPostalCode: string
+ addressStreetName: string
+ addressStreetNumber: string
+ addressClimateZone: enum
+ addressLongitude: float
+ addressLatitude: float
+ addressAltitude: float
1
1
locationInfo
Building
+ buildingID: string
+ buildingIDFromOrganization: string
+ buildingName: string
+ buildingConstructionYear: int
+ buildingConstructionType: enum
+ buildingUseType: enum
+ buildingOwnership: enum
+ buildingOpeningHour: time
+ buildingClosingHour: time
1
hasSpace
1
hasAreas
1
0..*
hasSubSpaces
0..*
BuildingSpace
+ buildingSpaceID: UID
+ buildingSpaceName: string
+ buildingSpaceUseType: enum
1
0..*
isInSpace
Area
+ areaType: enum
+ areaValue: float
+ areaUnitOfMeasurement: enum
0..*
*
UtilityPointOfDelivery
+ pointOfDeliveryID: UID
+ pointOfDeliveryIDfromUser: string
+ utilityType: enum
...
0..1
1
isContainedInSpaces
1
isAssociatedWithSpaces
0..*
observesSpaces
hasDeviceAggregator
0..*
DeviceAggregator
+ formula: string
isPartOfModelingUnit
0..*
isAssociatedWithElements
containsElements
1
1..*
1
hasSubElement
0..*
0..*
observ Text lements
0..*
affectsElements
Element
EnergyEfficiencyMeasure
+ energyEfficiencyMeasureID: UID
+ energyEfficiencyMeasureType: enum
+ energyEfficiencyMeasureDescription: string
+ shareOfAffectedElement: %
+ energyEfficiencyMeasureOperationalDate: date
+ energyEfficiencyMeasureInvestment: float
+ energyEfficiencyMeasureInvestmentCurrency: enum
+ energyEfficiencyMeasureCurrencyExchangeRate: float
+ energyEfficiencyMeasureSavingsToInvestmentRatio: floa
+ energySourcePriceEscalationRate: float
isAffectedByMeasures
0..*
BuildingElement
+ buildingElementID: UID
+ buildingElementState: boolean
+ buildingElementPurchaseDate: date
+ buildingElementInstallationDate: date
+ buildingElementBrand: string
+ buildingElementModel: string
+ buildingElementSerialNumber: string
+ buildingElementManufacturer: string
+ buildingElementManufactureDate: date
BuildingConstructionElement
+ buildingConstructionElementType: enum
hasUtilityPointOfDelivery
1
isObservedBy
0.*
includesDevices
0..*
hasDevices
Device
+ deviceID: UID
+ deviceName: string
+ deviceType: enum
+ deviceManufacturer: string
+ deviceModel: string
+ deviceNumberOfOutputs: int
+ deviceElectricSupply: enum
+ deviceOperatingSystem: string
+ deviceLicenseVersionNumber: string
+ deviceInputSignalType: enum
+ inputProtocol: enum
1.*
isInWeatherStation
0..1
WeatherStation
+ weatherStationCoordinates: point
+ weatherStationType: enum
+ weatherStationStartDate: datetime
+ weatherStationEndDate: datetime
+ weatherStationTimeStep: ?
1
hasMeasurementLists
0..*
MeasurementList
+ measurementUnit: enum
+ measuredProperty: enum
+ measurementDescription: string
+ measurementReadingType: enum
+ measurementTypeForEnergy: enum
+ measurementSourceForEnergy: enum
+ outputProtocol: enum
+ outputSignalType: enum
Measurement
+ measurementValue: any
+ measurementStart: datetime
+ measurementEnd: datetime
0..*
1
hasMeasurements

Figure 30: BIGG - **U**nified **M**odeling **L**anguage (Simplified Version)

E BIGG - VOWL

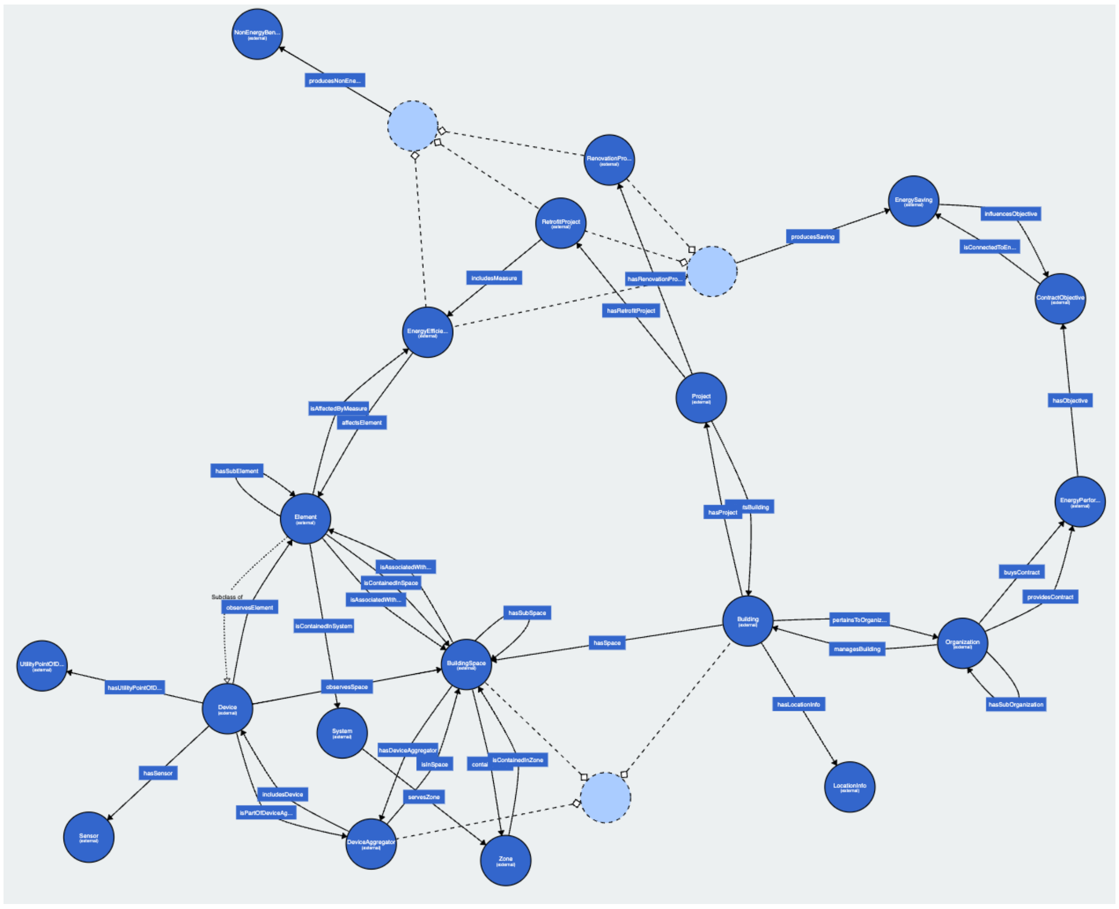


Figure 31: WebVOWL - BIGG Ontology

F React Roadmap

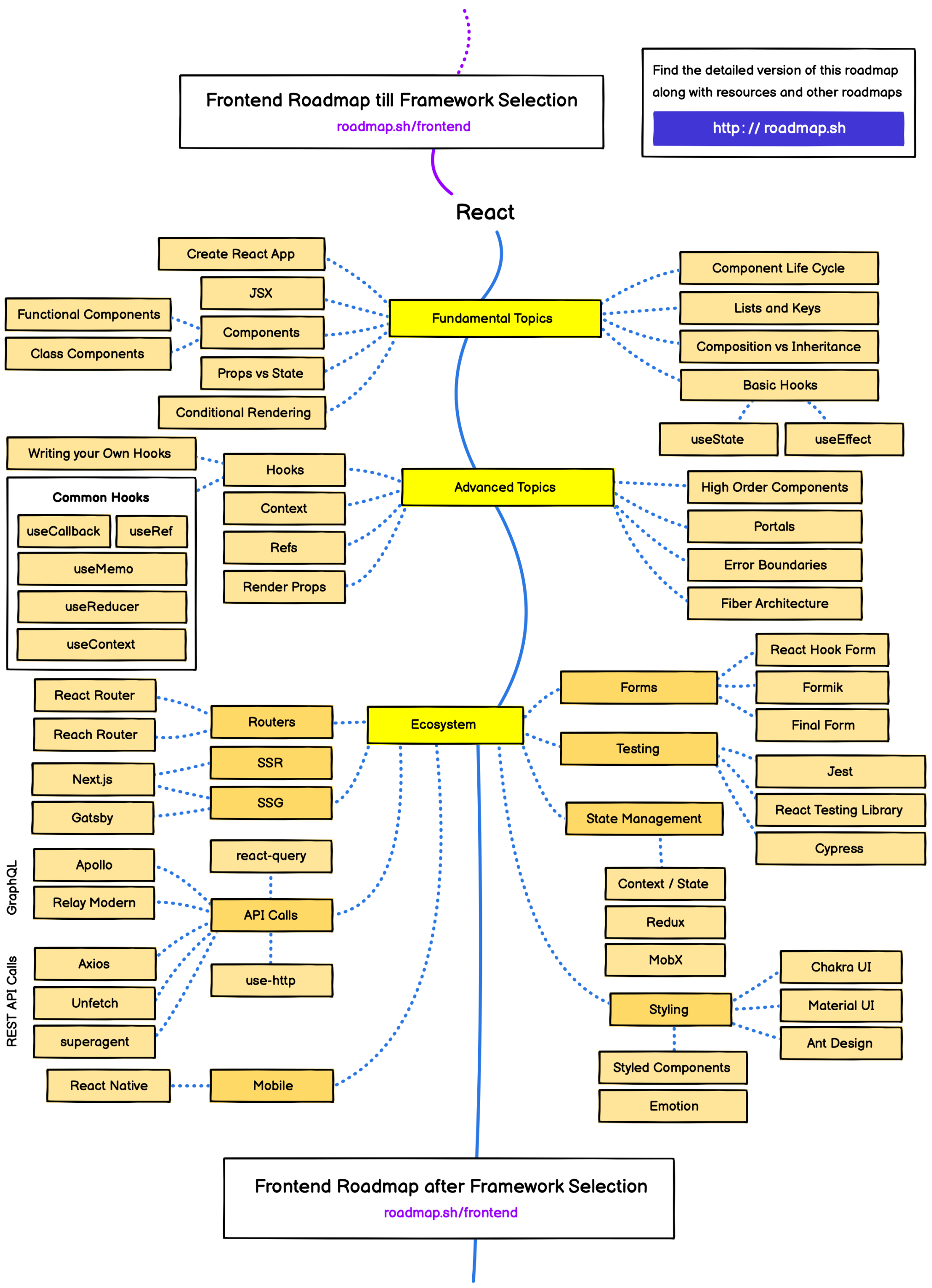


Figure 32: React Roadmap