Title:

DRR Data Ecosystem - Operational platform - Common information data sharing mechanism

Version 1.0

30th June 2021

Controls

Editors:

Agustin Garcia Pereira, Insight SFI Research Centre for Data Analytics, Ireland.

Tom Flynn, TFC Research and Innovation Limited, Ireland.

Contributors:

Luca Leonardi, Università del Piemonte Orientale, Italy.

Lian Guey Ler, University of Nice – Sophia Antipolis, France.

Rachele Brancaleoni, Poliiclinico Universitario Agostino Gemelli, Italy.

Natasha McCrone, Rinicom Limited, United Kingdom.

Patricia Compard, Ministre de Intérieur and CEN TC391 chairperson, France.

Clive Goodchild, BAE Systems, United Kingdom.

David Crouch, 3M Global Subject Matter Expert – Application Engineering (Defence & Public Safety), United Kingdom.

Arkadiusz Stasiewicz, arekstasiewicz@gmail.com, Ireland.

Espen Kon, EKON Modeling Software Systems, Israel.

Contents

List of Tables

List of Figures

Status of This Document

This section contains the status of this document at the time of its publication (30/06/2021). Other versions of the document may supersede this document. This document has been reviewed by interested parties and it may be used at user discretion as reference material or cited from another document and can be used to promote widespread deployment to enhance functionality and interoperability across related platforms.

Abstract

A wide range of threats and hazards can result in destabilising or disruptive events and escalate towards unpredictable and large-scale consequences for societal and citizen security and responses. Both public and private stakeholders require appropriate solutions in organisations, procedures, practices and technological capabilities to respond effectively across the disaster risk resilience chain. Various private and public organisations have technological capabilities to prevent incidents and to cope with the consequences, but many solutions do not have the technical capability to communicate effectively and share data from operations through to emergency medical services. Thus, there is a need to develop a technical better practice guide that is designed to enable organisations, on local, regional, federal, national and European level to be effectively coordinated and to cooperate with other organisations before, during and after a destabilising or disruptive event. Both semantic and technical interoperability definitions are key for the common information sharing space between platforms and solutions.

Introduction

Scope

This document describes the application programming interface (API) that defines the interaction between three Disaster Risk Resilient (DRR) related platforms and includes NO-FEAR, ENCIRCLE and STAIR4SECURITY. Specifically, the guide focuses on the communities of Chemical, Biological, Radiological, Nuclear and explosives (CBRNe) and Emergency Medical Services (EMS) and addresses:

- **Resource Description** – Vocabularies to describe the different types of resources (i.e. standards, pre-standardisation, better/best practice guidelines, operating manuals, research reports, etc.) to enable different platforms in the target community or ecosystem to exchange resources.

- **Platform Services** – Agree on a minimal set of APIs (Application Programming Interfaces) to be implemented by any platform operating in the target community through which requests for services can be made.

- **Security & Access** - Accessibility Profile for Resources and Services – Set of security specifications to be associated with resources and services maintained in the platforms.

The drafting of this version of the Guide is based on the work of three Horizon 2020 framework projects:

- NO-FEAR: (www.no-fearproject.eu), which brings together a pan-European and beyond network of emergency medical care and practitioners, suppliers, decision and policy makers to collaborate and exchange knowledge, good practices and lessons learned.

- ENCIRCLE: (www.encircle-cbrn.eu), which aim help create the tools and strategies needed to consolidate the EU CBRN communities of suppliers and practitioners in order to strengthen the field of CBRN safety, security and defence in the European Union.

- STAIR4SECURITY: (www.stair4security.eu), which aims to create a collaborative platform as single-entry point of information on the security sector coming mostly from research activities allowing a better governance of standardization needs in the Disaster Resilience and Chemical Biological Radiological Nuclear and explosive (CBRNe) as well as Emergency Medical Services.

The degree of interest across a wide variety of stakeholders (e.g., first responders, emergency medical services, etc.) is significant for Europe and for the strengthening of resilience to threats and hazards to European society and citizens.

Aim

The Better Practice Guide is a resource that defines a set of practices that allow users to integrate and extend the interfaces independently of the implementation and presently involves three-related projects, namely, NO-FEAR, ENCIRCLE and STAIR4SECURITY. The Guide presents the technical specification of the API, applicable standards as well as the data formats that should be use and the conventions to follow.

Background

This Guide was developed under the remit of the NO-FEAR project and is led by TFC Research and Innovation Limited. The undertaking stems from their work into standardisation for the emergency medical services and involved engagement with targeted participants (emergency medical services experts, first responders, etc.) at many of the NO-FEAR foresight workshop events. It also involved engagement with fellow NO-FEAR partners and Advisory Board project members, related EU supported projects, conference participation regards to European standardisation, internal research as well as discussions on the subject with the European Commission and related bodies and experts involved in standardisation. Through their engagement channels and applying a philosophy of 'listen and learn', it was evident how important interoperability was for the operations of emergency medical services and first responders. There is a legacy whereby many software platforms do not communicate or share common information spaces for the benefit of the users. Furthermore, technical outputs of many European operational security research projects follow a similar path. Data flow and information management between related platforms at semantic and technical levels are not well connected. In the market analysis update report Y4 (October, 2020), the ENCIRCLE project reported that with regards to standardisation there needs to be common communication protocols and information management. Likewise, at technical level, there exists the needs to improve standards and information management [Figure 1]. These findings are complemented by the European Commission Directive [COM (2017) 134] for the European Interoperability Framework – Implementation Strategy and more recently (22.2.21), in Article 6 of the Action Plan on synergies between civil, defence and space industries [COM (2020 70 Final], where it states:

Promotion and application of common standards across sectors can contribute to cost savings in terms of production runs and cost management, but also improve operational effectiveness, by enhancing interoperability, particularly in a multinational environment. Better linking standards with public security-related procurement programmes can help EU industry maintain its lead in critical technologies of importance to the EU's technological sovereignty. Overall, common standards can contribute to innovation and synergies.

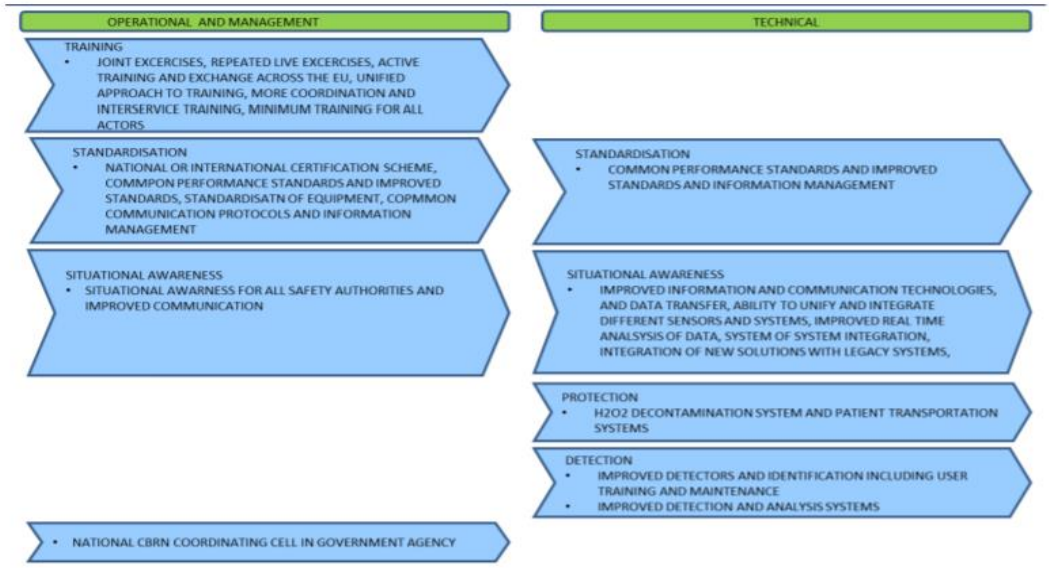

Figure 1. ENCIRCLE Report Update Y4 - Improved Interoperability

The ENCIRCLE market report clearly states that whilst the respondents believe that standards help the uptake of innovations and new market entrants, this has to be tempered with the responses that **Standards are poor and outdated and have little relevance to current capabilities**. A more **flexible approach may be required** here to meet both the Practitioner and Technological community's requirements.

Led by TFC Research and Innovation Limited (TFC), this SME, brought the issue to the attention of the work package 2 team in STAIR4SECURITY ('S4S') who were focused on the development of the standards, innovation and research platform for the disaster risk resilience including CBRNe and emergency medical services to act at the technology aggregators for the design and implementation of the and approached ENCIRCLE to discuss the issue of their data as part of a share common information space to better support the targeted Disaster Risk Resilience community. A face-to-face meeting was held at the joint NO-FEAR and STAIR4SECURITY event on standardisation, Rome, November 2021 involving each of the three projects. The original intention was to apply the revising CEN workshop agreement (CWA) in STAIR4SCURITY and although a separate workshop meeting, involving the three projects organised by TFC on the revised CWA mechanism was held, it became clear as well as through the STAIR4SECURITY Advisory Board discussions and other means that the proposed revised mechanism was not appropriate for the target community. Costs were cited as a reason. The mechanism is generic for all sectors and does not go far enough to meet the requirements of the community. The STAIR4SECURITY Advisory Board members were representatives of the emergency medical services and CBRNe community. Moreover, there is little evidence to show that a high proportion of CWAs have actually become full CEN EN/TS standards. The revised CWA is probably best suited to other sectors.

To bridge the gap of difficulty that the community has with this pre-standardization approach, it is important that practitioners take steps to develop practices that they want and suit their needs. Subsequently, TFC participated in NO-FEAR foresight meetings and concluded through discussions that practitioners were showing little interest in standardisation but were more interested in knowing about better practices. In fact, willingness to engage in standardization is not usually to the forefront

of their operational and perceived needs. However, practitioners having the freedom to organise themselves in producing harmonised procedures without incurring high costs was of interest to them. Practitioners also having better access to common shared data and information management was of interest. This led to the belief that alternative ways, complementary to formal standardisation, would need to be developed and encouraged across the Disaster Risk Resilience community. Thus, the concept of Better Practice Guide for the target community was born.

The Better Practice Guide approach is treated at an early level of development and as a pre-standardization activity, but it is entirely up to the authors if they wish to progress to a full standard. The approach empowers practitioners to actively contribute to the development of processes and procedures that are needed by them. This is critical. The implementation is built from the following principals:

- "in practice but not necessarily ordained by law"
- "in practice or actuality, but not officially established"

Presently, TFC in the NO-FEAR project is driving the development of a Better Practice ecosystem of which this document will eventually reside within.

Given the ongoing pandemic situation both virtual management and technical 1-to-1 meetings were held to address application programming interface and in particular the individual platform common data type to share in line with GDPR, taxonomies as well as permissions, data management, etc. The technical undertaking was led by the Insight SFI Research Centre for Data Analytics, who developed the STAIR4SECURITY platform, which is the aggregator platform communicating with NO-FEAR and ENCIRCLE platforms.

Application Programming Interface

An API (Application Programming Interface) is defined by the IEEE as a software interface that allows computers to request, retrieve, and exchange data and information in a standardized way. An API will be used by other developers and thus, the Developer eXperience (DX) needs to be addressed since its early design steps. In this context, REST (short for REpresentational State Transfer) is an architectural style defined to help create and organize distributed systems. A RESTfully organized distributed system, will experience improvement in the following aspects:

- *Performance:* The communication style proposed by REST is meant to be efficient and simple, making a more efficient use of the network and enabling higher performance perceived by the users.
- *Scalability:* the simple interaction proposed by REST contributes to this. The stateless communication between server and client guarantees that any request made by the client contains all the information needed (such as credentials for authentication), and that no information needs to be stored.
- *Simplicity of interface:* A simple interface allows for simpler interactions between systems
- *Modifiability of components:* The distributed nature of the system, and the separation of concerns proposed by REST, allows for components to be modified without affecting the others, thus reducing costs and risks.

- *Portability:* REST can be implemented and consumed by any type of technology.
- *Reliability:* The stateless constraint allows for the easier recovery of a system after failure.
- *Visibility:* Supervising the system becomes easy when all the information required is inside the request.

REST architecture treats every content as a resource [1] . Resources define what the services provided by the API will be about. Despite the fact that a resource can be represented in different formats, JSON has been widely used as the standard Data Transfer Format. JSON has many advantages over other data formats, such as XML, previously popular in the domain. It is lightweight since most of the information it transports belongs to the actual data. Another advantage is that it is easily readable by humans, and it can transport many data types other than strings.

A client-server application is an important constraint of a RESTfull API. A server provides services that a client application will consume. The separation of concerns is thus, one of the main benefits of this architecture, since modifications on one side do not have impact on the other, as long as the services communication interface is not modified. This allows for a separation between backend and frontend code. However, this is not the only situation.

Several technologies exist nowadays to support the development of an API ecosystem of this nature. Among the most up to date ones we can name NodeJS JavaScript runtime environment, and PHP programming language. Both of them can be used in server-side applications to serve REST API applications.

NodeJS is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser. It works in an asynchronous way, preventing any blocking and allowing different code to be run at the same time a portion of code is waiting for a call back to be completed. Different libraries can be used to create a RESTful API in a NodeJS environment, such as Express. On the other hand, PHP runs in web servers such as Apache or IIS, and is supported in many popular content management systems such as WordPress, and Joomla, which makes it a convenient choice when working with this type of systems.

Query Language

Data query languages are computer languages used to make queries in databases and information systems. GraphQL[1] is a query language for implementing web service architectures, it was developed by Facebook and open-sourced in 2015. In GraphQL, the clients can define the structure of the data to be requested, and the same structure of the data is returned from the server. Despite the fact that GraphQl has been compared with REST applications and even considered a substitute of REST, both styles can co-exist in the same ecosystem. GraphQL can wrap an existing REST API service, or in the case of the S4S ecosystem, it can encapsulate several REST API services and provide a common standardized query language interface based on graphs and not in endpoints.

The use of GraphQL as a common query language can provide several benefits. Specifically, it solves two of the main problems of REST APIs, underfetching and overfetching. The overfetching is

[1] https://graphql.org/

understood as the problem that arises when a client downloads more data than the data that actually needs, due to the fixed endpoint nature of REST services. Underfetching happens when a client needs to perform more than one request to download the required data.

Graphql uses a type system based on schemas to define the capabilities of an API, and defines its own language to write them down, known as the GraphQL Schema Definition language (SDL) [2].

Data Vocabularies

Data Catalog Vocabulary (DCAT) is an RDF vocabulary designed to facilitate interoperability between data catalogues published on the Web[2]. Its usage to describe datasets in catalogues enhances data discoverability and metadata exchange between systems. Asset Description Metadata Schema (ADMS) is a profile of DCAT used to describe assets in the context of eGovernment system development. Users searching for Assets can have different expectations than someone looking or datasets. Assets are different to datasets since they are expected to be something users can open and read using software tools, in contrast to datasets that need to be processed. Figure 2 presents the ADMS Domain Model using an UML diagram[3].

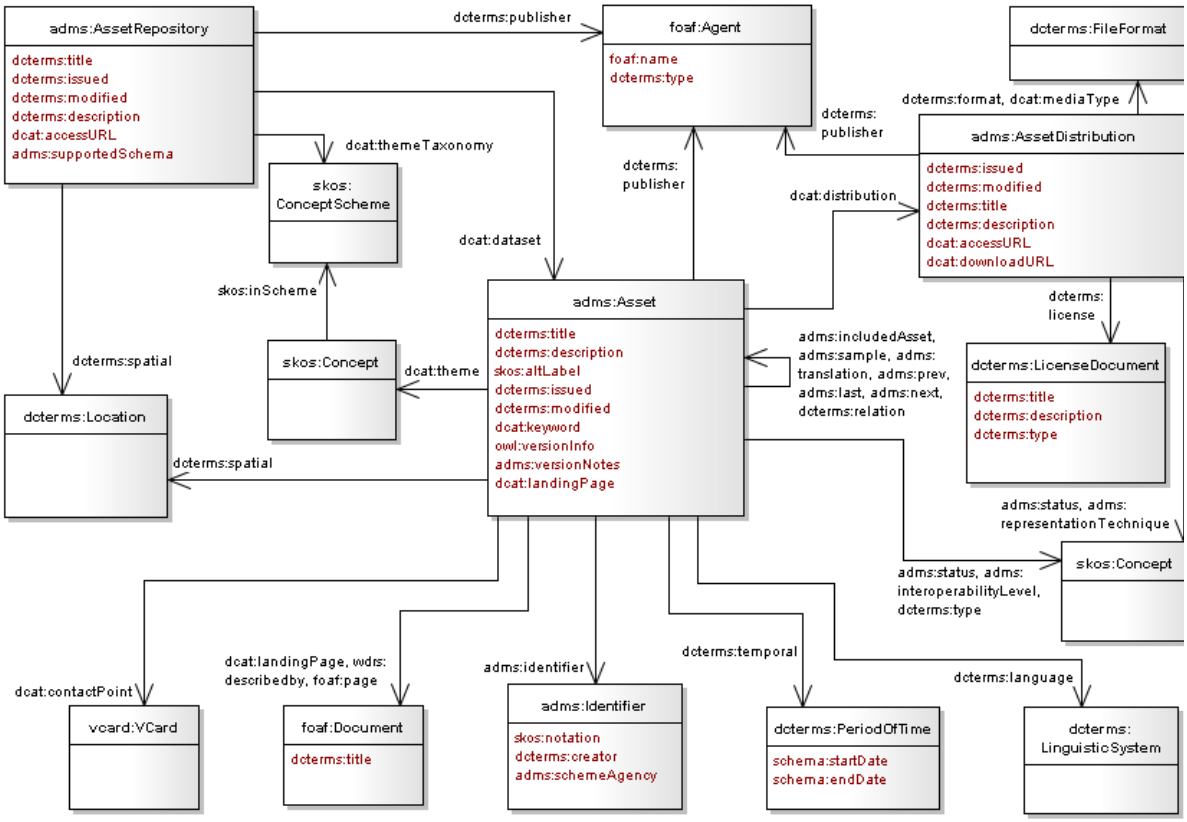

Figure 2. ADMS UML model

[2] https://www.w3.org/TR/vocab-dcat-2/
[3] https://www.w3.org/TR/vocab-adms/

Methodology and Development

Developing a data ecosystem that can satisfy technical interoperability within platforms and, at the same time, focus on users' needs and interests throughout the ecosystem data pipeline requires a flexible methodology that can build upon users' knowledge and expectations.

During the project we followed a User-Centred Design (UCD) iterative design process, involving users and multidisciplinary stakeholders throughout the design and development process in a number of iterations. UCD is a project-oriented methodology for interactive systems development. ISO 9241-210 (i.e., Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems) describes a number of principles that should be considered in the development of a system centred on users and their needs. The project should be conducted and refined though assessments focusing on the user and in an iterative process, reducing the risk of not reaching the desired requirements [3]. Following this approach, we ensured that any technology developed in the project had users' needs set as the top priority. That design approach was supported by the agile development process, yielding several iterations of technology releases in close collaboration with our stakeholders. Every cycle was initiated with requirements' analysis and finalised with relevant evaluation by our users in the form of dedicated focus groups. In each iteration we evaluated different aspects of the data ecosystem, such as data integration and accessibility from the perspective of the involved projects, and data completeness and understandability from the perspective of the Stairs4Security platform end users.

Proposed Solution

In this section, we will first present the final S4S Data Ecosystem Architecture so that readers can understand the system general concept. Later on, we will explain the technical constraints and decisions that informed this architecture. We will first introduce the external data sources, NO-FEAR and ENCIRCLE, data structures. We will then introduce the available API endpoints to query this data and we will depict the S4S Asset data structure. In this context, we will present the developed S4S taxonomy that enables data categorization and facilitates searching and filtering assets in the domain. Finally, we will explain how data integration is performed, and we will introduce the fundamental requirements for other projects to join the S4S Data Ecosystem.

DRR Data Ecosystem architecture

In this subsection, we will present the DRR Data Ecosystem Architecture components and we will describe how they intercommunicate. Figure 3 presents an architecture diagram of the ecosystem. On the left hand-side we can observe NO-FEAR and ENCIRCLE REST APIs. As described in Introduction section, REST APIs are software instruments that allow computers exchange data and information in a standardized way. In the case of the STAIR4SECURITY project (S4S) a REST API is beneficial for separating concerns between the data providers and the data ecosystem here developed. In this case, the GraphQL Server will act as a server and as a client at the same time. The GraphQL Server will consume (client) the REST API services provided by NO-FEAR and ENCRICLE, while at the same time will serve (server) S4S Platform client data integrated from the different sources. The GraphQL server will be responsible for handling security concerns for accessing external data sources, as well as for transforming the data they provide to comply with the S4S data models.

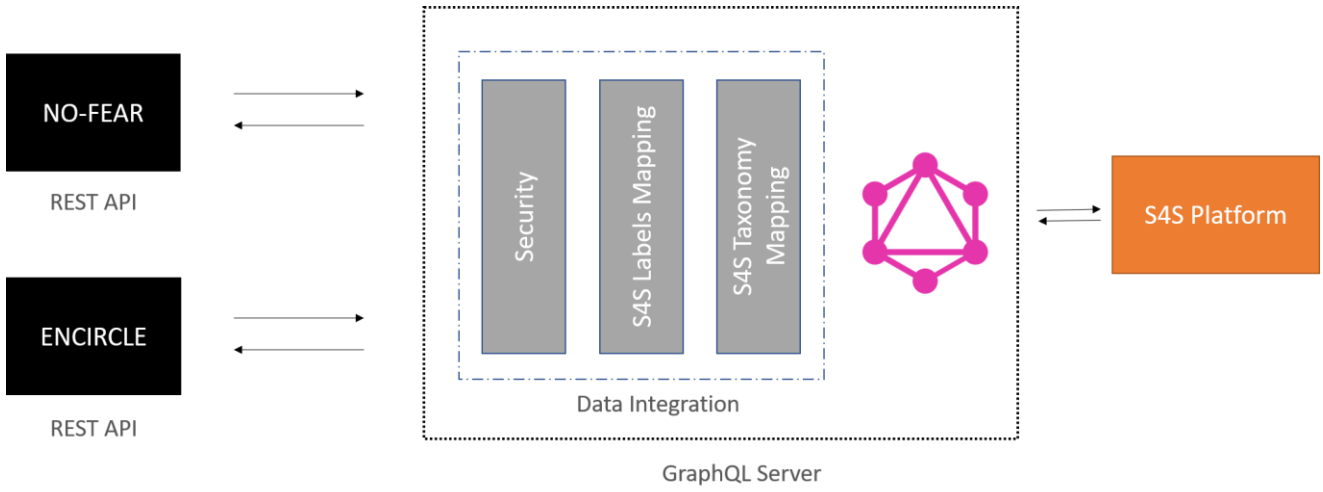

Figure 3. Ecosystem Architecture

External data structures

In this subsection we will describe the data structures of NO-FEAR and ENCIRCLE projects currently integrated in the DRR Data Ecosystem. Figure 4 presents the NO-FEAR data model, while Figure 5 presents Encircle structure. We can observe that despite both models having similarities, the models are different in terms of structure and taxonomy used. Thus, creating a challenge to integrate both data sources. We will address this issue presenting a common data structure later in this chapter.

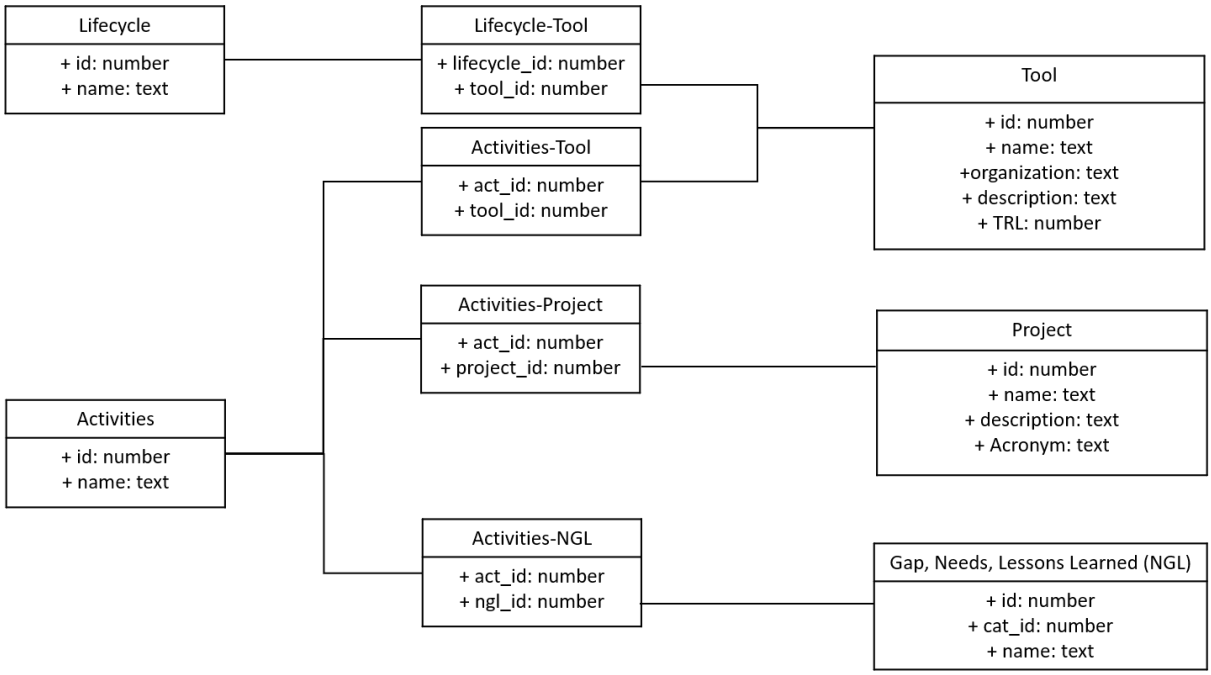

Figure 4. NO-FEAR Data Structure

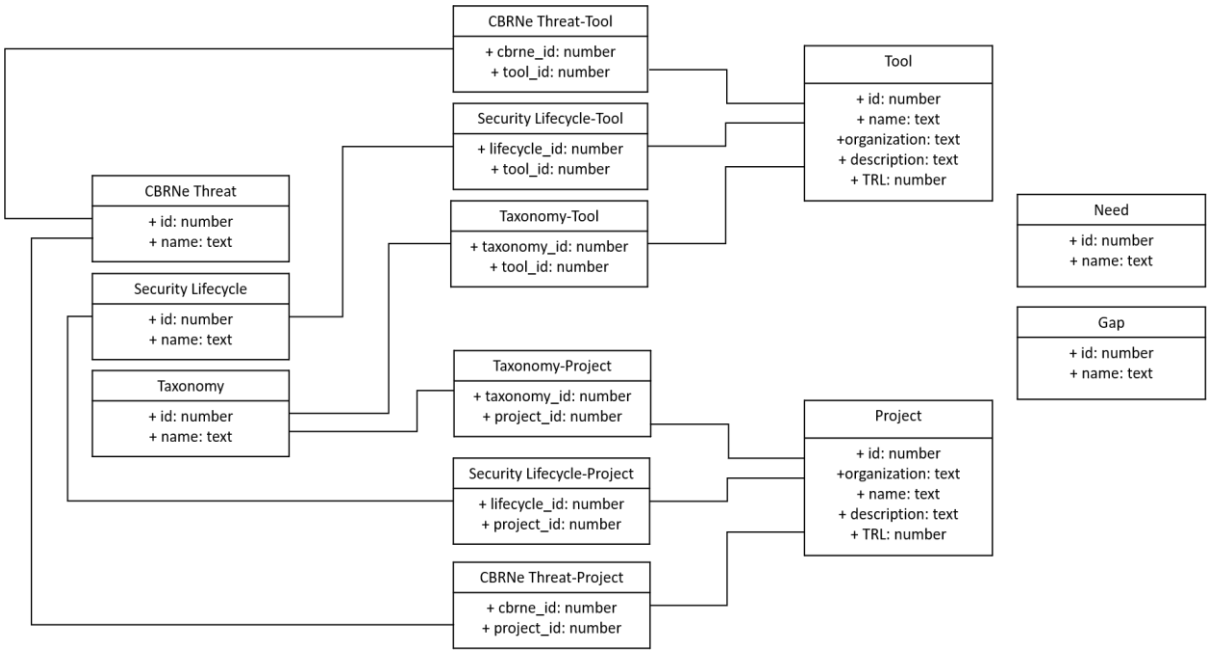

Figure 5. Encircle Data Structure

External API endpoints

Data resources should provide a unique way to be identified and accessed at any time, and a set of operations that can be performed over them. The HTTP protocol that most REST APIs rely on, provides a set of default actions that a client can perform on the different resources named GET, POST, PUT, DELETE, OPTIONS, and HEAD. Despite this fact, other operations over the resources can be defined, such as filtering, getting a subset of resources based on a set of restrictions.

In REST-based APIs, data is exposed by means of endpoints. In this section we present a possible interface definition for the API. Following the data model presented before, we now present the available API endpoints to consume the different resources (Projects, Tools, Gaps, Needs, and Lessons Learnt) in an independent, but also in a collective way. In other words, we present the mechanism to access every single resource by indication of a unique identifier, but also the accessibility of each of the different resources as a collection of resources. The later can be used to obtain the available IDs list.

Resource	Projects
HTTP Action	GET
Collection URL	http://api.no-fearproject-portal.eu:8001/api/s4s/projects http://api.no-fearproject-portal.eu:8001/api/s4s/activities_projects
Single URL	http://api.no-fearproject-portal.eu:8001/api/s4s/projects/id

Resource	Tools
HTTP Action	GET
Collection URL	http://api.no-fearproject-portal.eu:8001/api/s4s/tools http://api.no-fearproject-portal.eu:8001/api/s4s/lifecycles_tools http://api.no-fearproject-portal.eu:8001/api/s4s/activities_tools
Single URL	http://api.no-fearproject-portal.eu:8001/api/s4s/tools/id

Resource	Tools
HTTP Action	GET
Collection URL	http://api.no-fearproject-portal.eu:8001/api/s4s/tools http://api.no-fearproject-portal.eu:8001/api/s4s/lifecycles_tools http://api.no-fearproject-portal.eu:8001/api/s4s/activities_tools
Single URL	http://api.no-fearproject-portal.eu:8001/api/s4s/tools/id

Resource	Needs, Gaps, Lessons Learnt
HTTP Action	GET
Collection URL	http://api.no-fearproject-portal.eu:8001/api/s4s/ngl

	http://api.no-fearproject-portal.eu:8001/api/s4s/activities_ngl
Single URL	http://api.no-fearproject-portal.eu:8001/api/s4s/ngl/id

Resource	Others tables
HTTP Action	GET
Collection URL	http://api.no-fearproject-portal.eu:8001/api/s4s/lifecycles http://api.no-fearproject-portal.eu:8001/api/s4s/activities http://api.no-fearproject-portal.eu:8001/api/s4s/nglcat

Table 1. NO-FEAR API Endpoints

Resource	Projects	Tools
HTTP Action	GET	GET
Collection URL	http://api.encircle.eu:8000/api/s4s/projects http://api.encircle.eu:8000/api/s4s/cbrne_project http://api.encircle.eu:8000/api/s4s/project_taxonomy_parent	http://api.encircle.eu:8000/api/s4s/tools http://api.encircle.eu:8000/api/s4s/cbrne_tool http://api.encircle.eu:8000/api/s4s/tool_taxonomy_parent
Single URL	http://api.encircle.eu:8000/api/s4s/projects/id	http://api.encircle.eu:8000/api/s4s/tools/id

Resource	Needs	Gaps
HTTP Action	GET	GET
Collection URL	http://api.encircle.eu:8000/api/s4s/needs	http://api.encircle.eu:8000/api/s4s/gaps
Single URL	http://api.encircle.eu:8000/api/s4s/needs/id	http://api.encircle.eu:8000/api/s4s/gaps/id

Table 2. Encircle API Endpoints

A common data structure

Based on the data integration challenge previously described, we present in Figure 6 a common data structure amalgamating both projects' structures and taxonomies. We inspired this data structure in the ADMS vocabulary presented in the Introduction section.

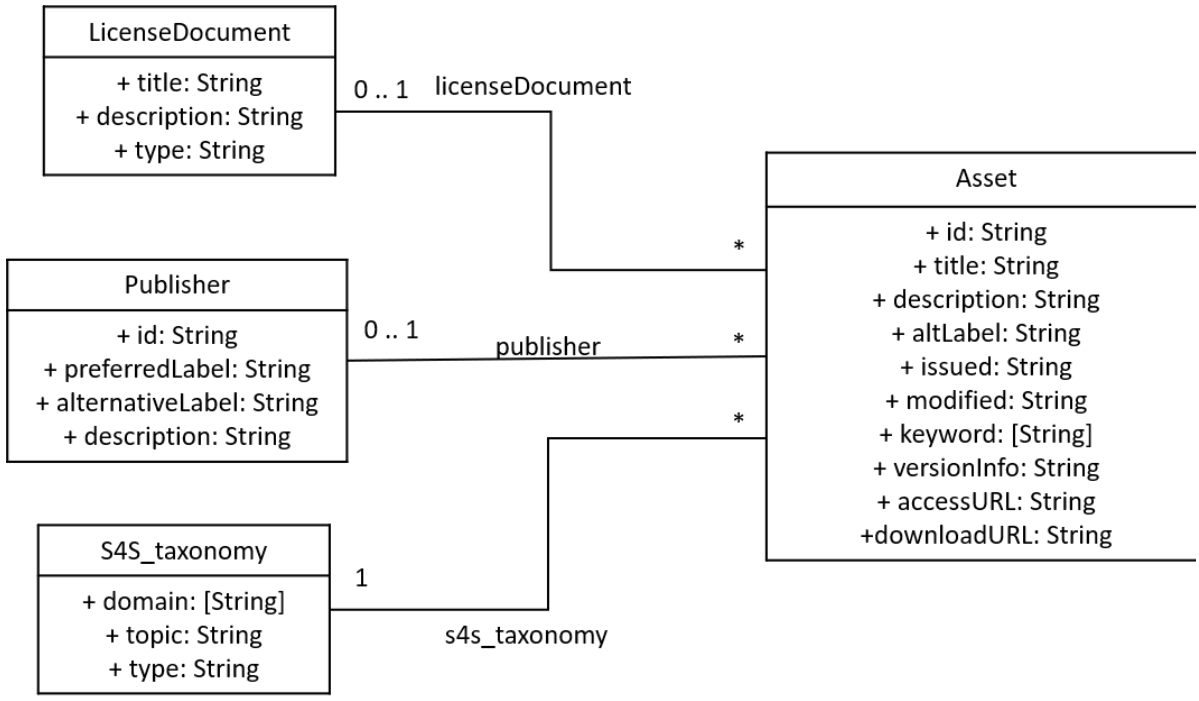

Figure 6. S4S Asset Data Structure

Next, we describe each concept of the Asset Data Structure presented before.

Concept	Type	Description
id	String	A unique string that will identify each Asset independently.
title	String	A name given to the Asset
description	String	A description of the Asset
altLabel	String	An alternative name for the Asset
issued	String	Date of formal issuance (e.g., publication) of the Asset
modified	String	Date of the latest update of the Asset

keyword	[] String	A set of words, phrases or tags to describe the Asset
verstionInfo	String	A version number or other designation of the Asset
accessURL	String	Any kind of URL that gives access to an Asset e.g. a landing page, download, feed URL, SPARQL endpoint etc.
downloadURL	String	This is a direct link from an Asset to a downloadable file in a given format, e.g. CSV file or RDF file
language	String	The language in which the Asset is available

Table 3. Asset Concepts

Concept	Type	Description
title	String	A name given to the license
description	String	A short description of the License terms
type	String	A string unique identifier for the license

Table 4. License Document Concepts

Concept	Type	Description
id	String	A string unique identifier for the Publisher
preferredLabel	String	A name for the Publisher
alternativeLabel	String	An alternative name for the Publisher
description	String	A short description of the Publisher

Table 5. Publisher Concepts

Concept	Type	Description
domain	[String]	A set of predefined options. See S4S Taxonomy Section.
topic	String	One of a predefined list of options. See S4S Taxonomy Section.
type	String	One of a predefined list of options. See S4S Taxonomy Section.

Table 6. S4S Taxonomy Concepts

S4S Taxonomy

Users of the S4S Platform pointed out the need for a unified taxonomy that represented the key concepts of the domain and allowed them to filter the resources available on the platform following

an intuitive representation. Taking this requirement into consideration, the team conducted a series of iterations where taxonomies options were proposed, discussed, and refined with users and stakeholders. After five iterations, we converged on a taxonomy encompassing three levels, named Domain, Topic, and Type. Each level contained nine, eight, and twelve options, respectively. Table 7. S4S Taxonomy – DomainTable 7, Table 8, and Table 9 present each taxonomy level with their associated options. Finally, Figure 7 shows how this taxonomy is used in the STAIR4SECURITY (S4S) Platform.

Domain	
id	**label**
1	Risk Assessment and Reduction
2	Protection
3	Training and Simulation
4	Detection, Identification and Monitoring
5	Situation Awareness and Information Management
6	Crisis Operations and Management
7	All-Hazard Management
8	Medical Intervention and Counter-measures
9	Communication/Community Involvement

Table 7. S4S Taxonomy – Domain

Topic	
id	**label**
1	Chemical
2	Radiological/Nuclear
3	Biological
4	Explosive
5	CBRNe Combined
6	Natural Disaster
7	Man-Made Disaster
8	Undefined

Table 8. S4S Taxonomy - Topic

Type	
id	**label**
1	Standards
2	Pre-standardization
3	Best Practices
4	Lessons Learnt
5	EU Regulation
6	EU Research Projects

7	Needs
8	Gaps
9	Standard Operating Procedures
10	Tools and Services
11	Projects
12	Other Outcomes

Table 9. S4S Taxonomy – Type

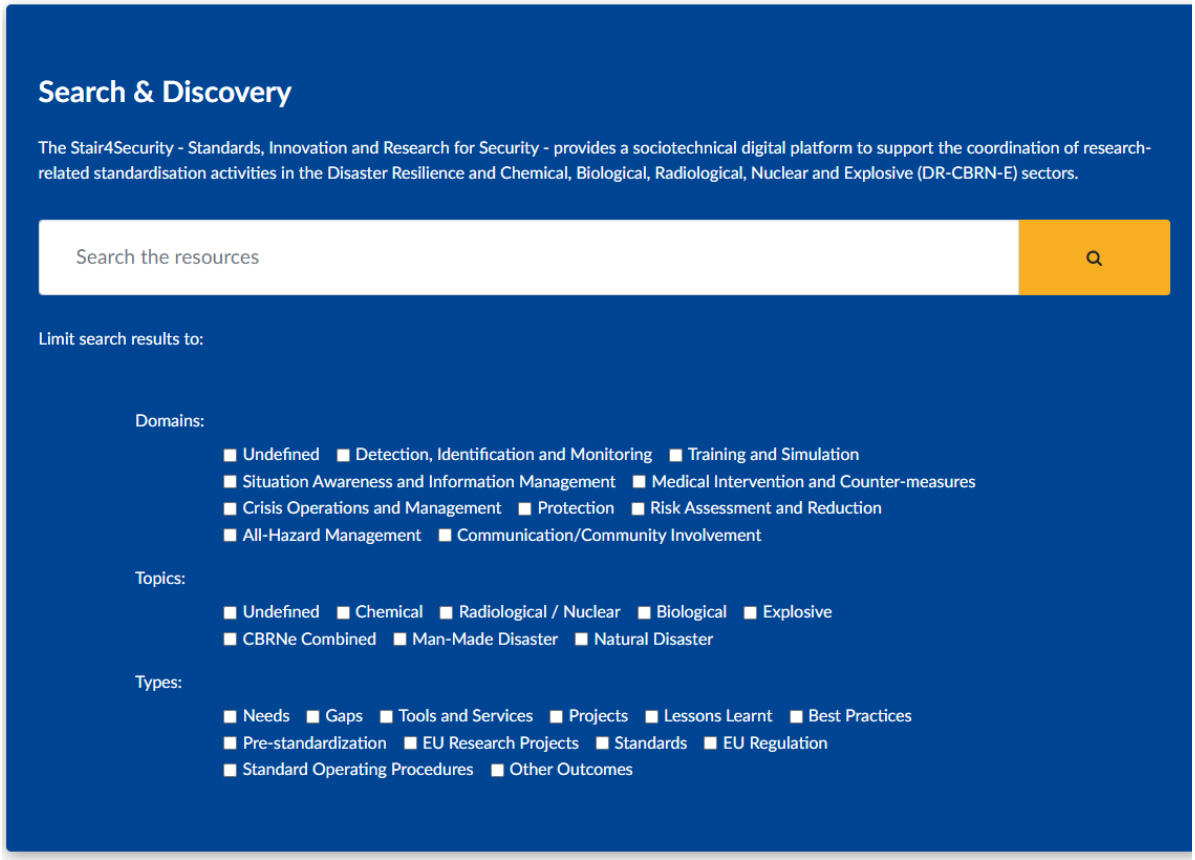

Figure 7. S4S Taxonomy in the Platform

Data integration

From the perspective of the users of the DRR Data Ecosystem, the need for a common query language that can facilitate data consumption from different platforms was identified. Despite the fact that REST APIs would satisfy data communication requirements, an outstanding solution is proposed in this project. GraphQL, a common query language has many advantages over REST APIs, as explained in Introduction section. This technology brings together several benefits such as more elegant data retrieval, more efficient, highly compatible with other systems independently of the languages or data types, and more importantly has the potential to increase data integration across several systems.

We implemented a GraphQL Server Framework using Node.JS and ApolloServer[4] in javascript language. Figure 8 depicts a static view of the server. Four main components can be identified:

[4] https://www.apollographql.com/

Application, Types, Sources, Mappings and Modules. Each component has different responsibilities and we will describe them next.

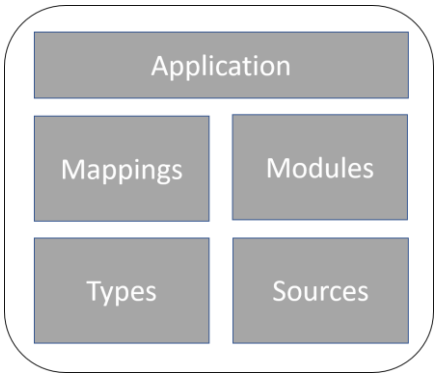

Figure 8. GraphQL Server Framework - Static View

The Types component defines the different data models presented in A common data structure section using the GraphQL Schema Definition language.

```
module.exports =   `type Asset {
    "A unique string that will identify each Asset independently."
    id: String,
    "A name given to the Asset"
    title: String,
    "A description of the Asset"
    description: String,
    "An alternative name for the Asset"
    altLabel: String,
    "Date of formal issuance (e.g., publication) of the Asset"
    issued: String,
    "Date of the latest update of the Asset"
    modified: String,
    "A set of words, phrases or tags to describe the Asset"
    keyword: [String],
    "A version number or other designation of the Asset"
    versionInfo: String,
    licenseDocument: LicenseDocument,
    publisher: Publisher,
    "Any kind of URL that gives access to an Asset e.g. a landing page, down
load, feed URL, SPARQL endpoint etc."
    accessURL: String,
    "This is a direct link from an Asset to a downloadable file in a given f
ormat, e.g. CSV file or RDF file"
    downloadURL: String,
    s4s_taxonomy: S4S_taxonomy,
    language: String
}
`;
```

Figure 9. Asset Type Definition - Example

The Mappings component is responsible for defining, for each external data source, how data will be transformed to fit the common data structure proposed, and how will the S4S Taxonomy be mapped. This last point is very important. Here the framework developed presents a flexible mechanism to automate the taxonomic mapping. It lets the developer build complex rules based on available data coming from the REST APIs to map the S4S Taxonomy.

```
module.exports.projects = (assets) => {

    let mapTopic = (topic)=> {
        return topic== 'unknown' ? 'undefined' : topic
    }

    let mapDomain = (project) => {
        let id_tax_mapping = csvToJson.fieldDelimiter(',').getJsonFromCsv(pa
th.resolve(__dirname, 'csv/encircle_projects_domain.csv'));
        let taxonomy_id = id_tax_mapping.filter(id=> id.project_id == projec
t.project_id);
        taxonomy_id = taxonomy_id.map(tax=>tax.taxonomy_id);
        let mapping= require('./mapping').mapping;
        taxonomy_id = taxonomy_id.map(tax=>mapping[parseInt(tax)]);
        taxonomy_id = [...new Set(taxonomy_id)];
        console.log("Taxonomy id", taxonomy_id)
        return taxonomy_id.length>0 ? taxonomy_id : ['undefined']
    }

    return assets.map(asset => {
        return {
            id: asset.project_id,
            title: asset.project_acronym,
            description: asset.project_signification,
            altLabel: null,
            issued: null,
            modified: Date.now(),
            keyword: [],
            versionInfo: null,
            licenseDocument: constants.LICENSE.UNKNOWN,
            publisher:{
                id: null,
                preferredLabel: asset.organisation_name,
                alternativeLabel: asset.organisation_name,
                description: null
            },
            accessURL: "https://www.encircle.eu/project/show/"+ asset.projec
t_id,
            downloadURL: "https://www.encircle.eu/project/show/"+ asset.proj
ect_id,
```

```
            s4s_taxonomy: {
                domain: mapDomain(asset),
                topic: mapTopic(asset.CBRNE_slug),
                type: constants.TYPE.PROJECTS
            },
        }
    })
    }
```

Figure 10. Encircle Projects Mapping Example

The Sources component is responsible for defining how to access the external data sources. Here we define API endpoints or data files, and we store security credentials. It interacts with APIs to get security tokens and stores them for subsequent queries.

The Modules component hosts external libraries needed to process the data, such as json files management, HTTP requests, among others.

Finally, Application component is responsible for serving the GraphQL requests, filtering the data, and defining how each data type query will be solved.

The GraphQL allows for two types of queries: assets and filterAssets, as shown in Figure 11 and Figure 12 respectively. In the first one, a user can query for all the assets available indicating the fields that it would like to retrieve, as in example shown in Figure 13. In the second case, a user can filter the assets based on the S4S Taxonomy developed. The suer can specify a domain, topic or type category of its interest, or a set of them, and the GraphQl API will filter the resources coming from the external data sources based on these parameters. Figure 14, Figure 15, and Figure 16 show examples of this queries.

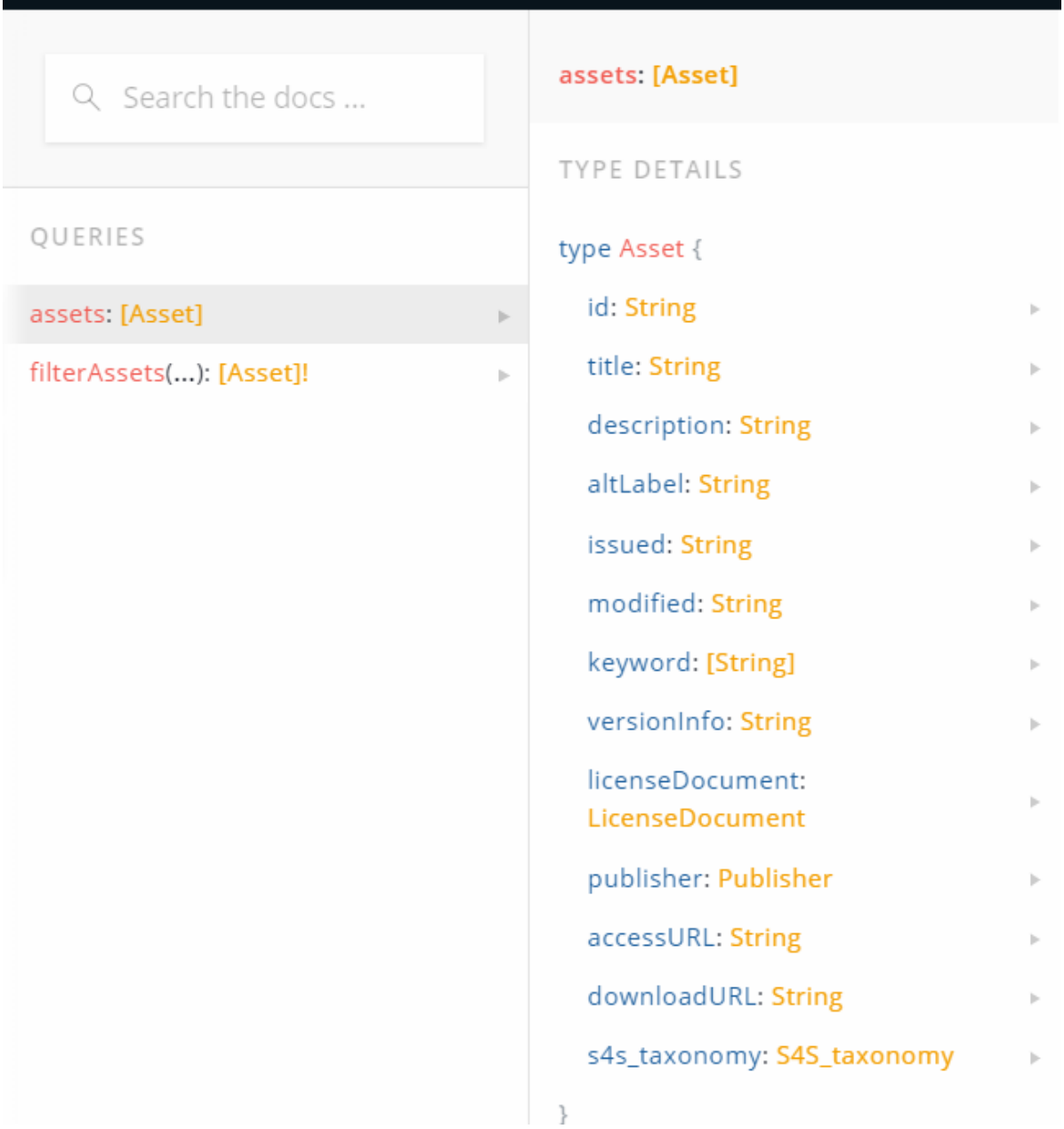

Figure 11. Assets Query

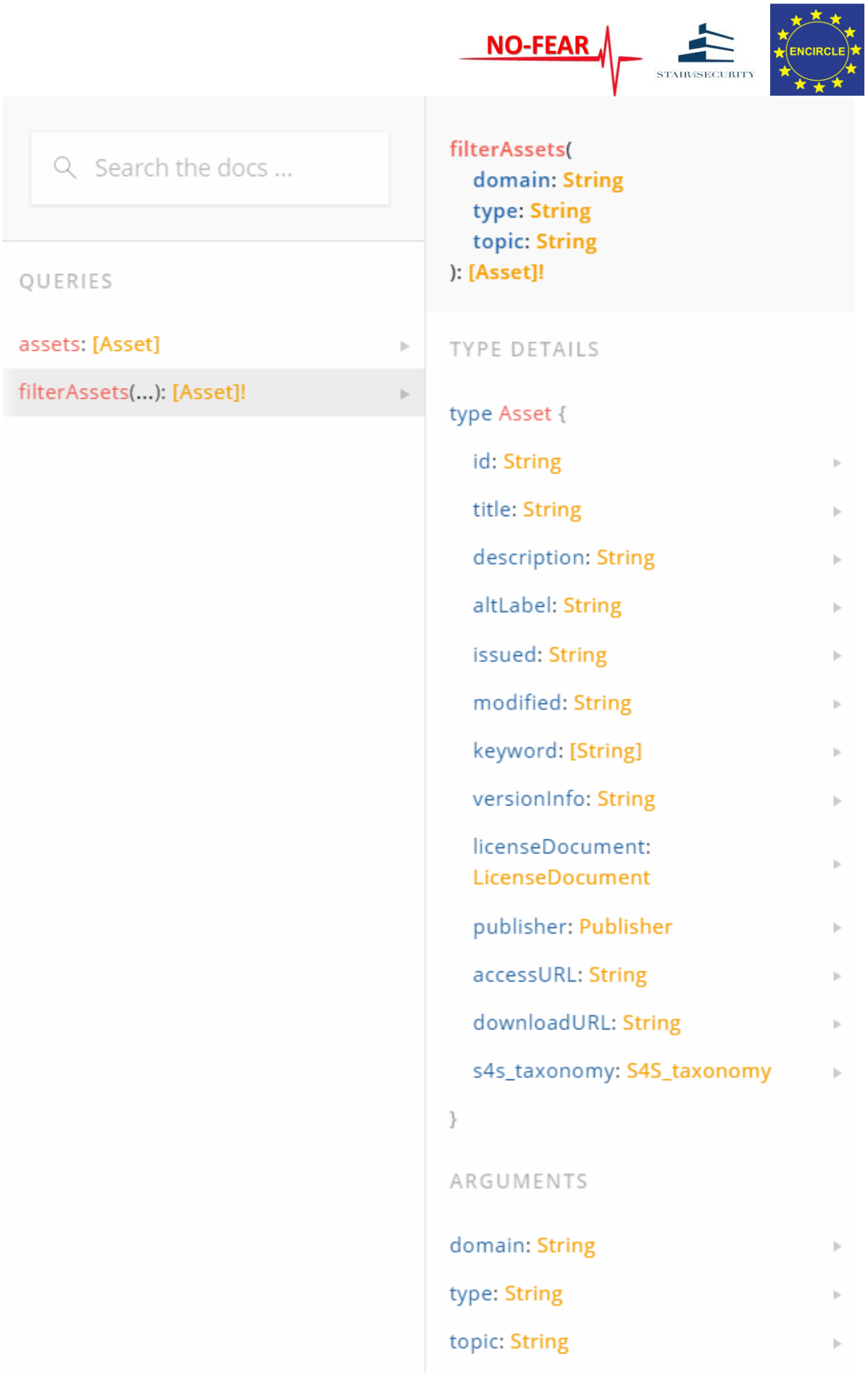

Figure 12. filterAssets Query

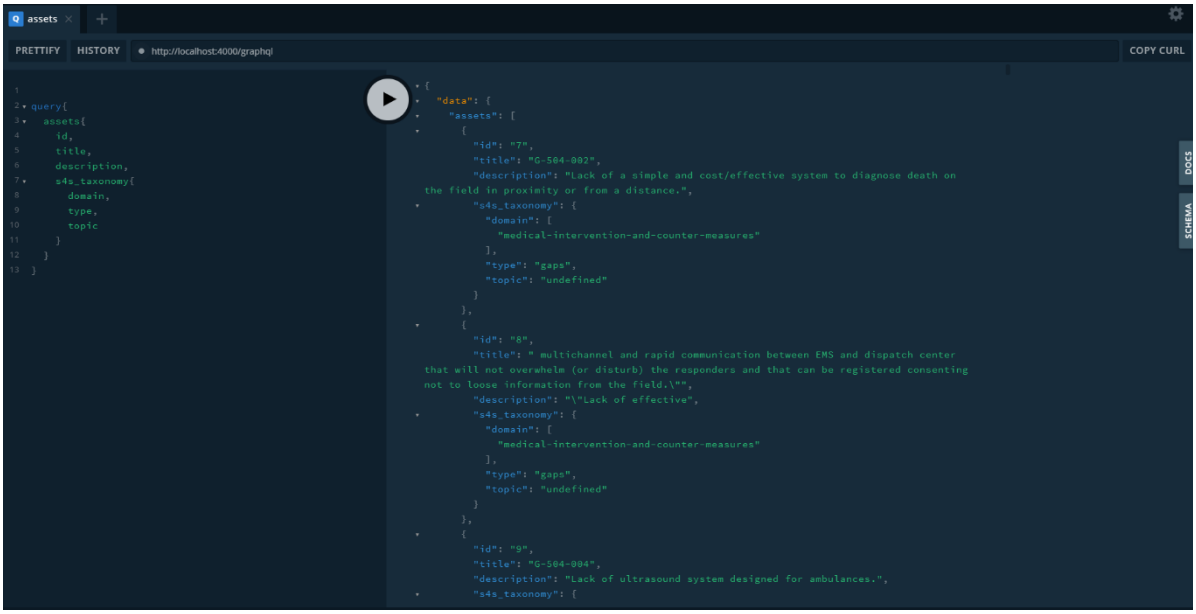

Figure 13. Assets Query Example

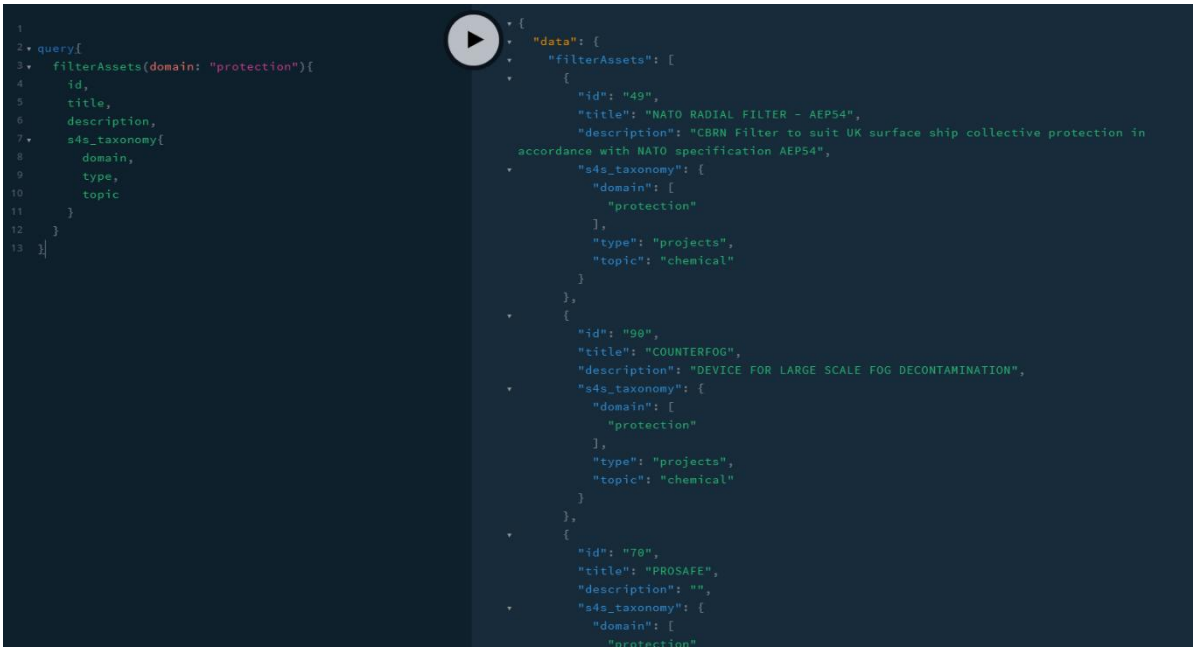

Figure 14. filterAssets Query Example - Simple

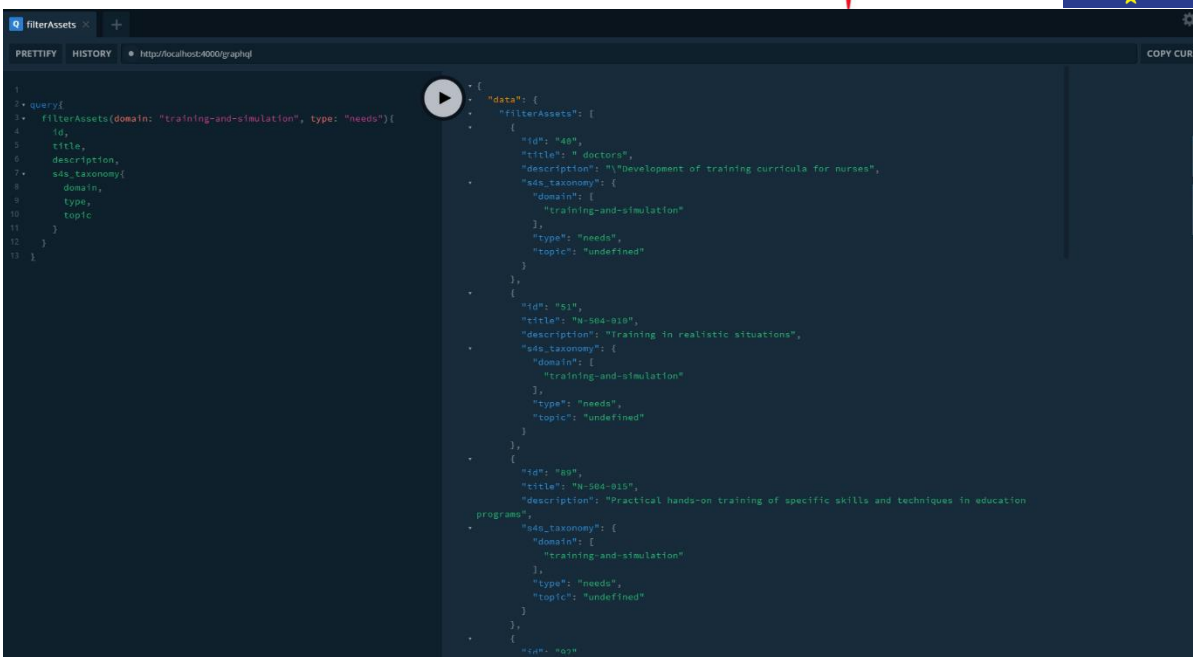

Figure 15. filterAssets Query Example - Complex

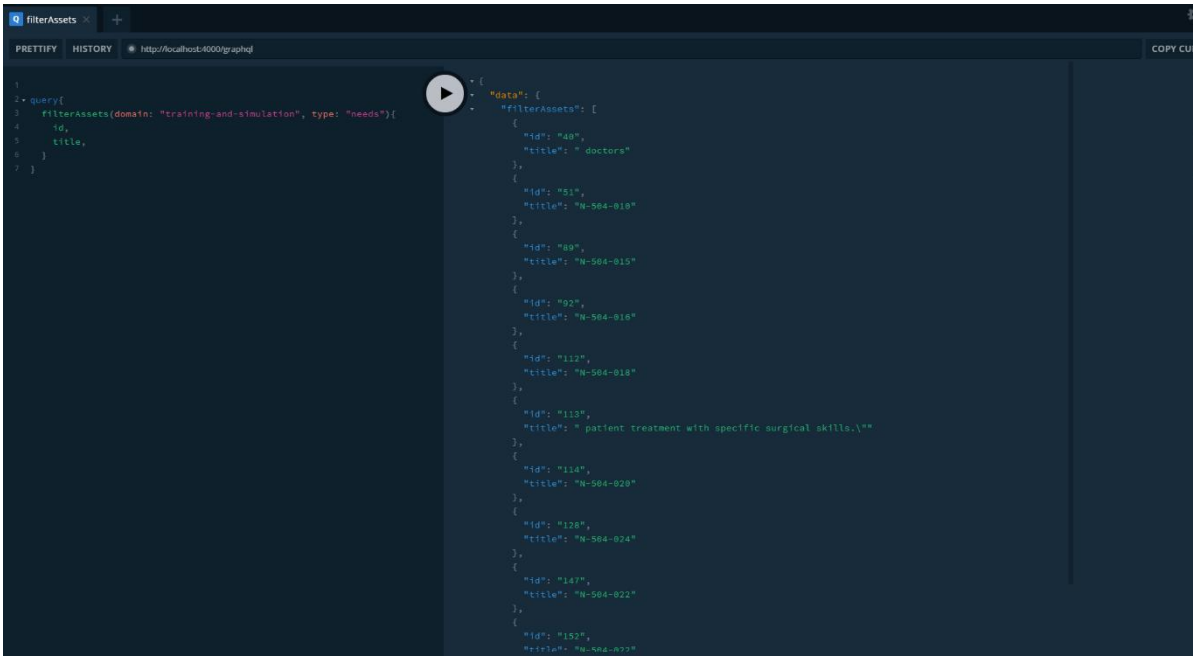

Figure 16. filterAssets Query Example – Reduced Fields

Security

Regarding the security of the API, JSON Web Tokens (JWT) are used for the communication between the REST APIs and the S4S GraphQL server, or client. JWT is an internet standard that provides a

method to encapsulate and share claims between peers in a secure way by using JSON objects. The content encoded inside the token can provide information about the user role and permissions.

Once the client provides the correct credentials, the server replies with a valid JWT (special string). This token is stored in the local memory of the client, hence, letting the perform different queries to the server. Every query done to the API must be sent including the Authorization header containing the token. On the server side, the token is decoded using a secret key. Thus, checking its correctitude, and denying the service to any query that does not match this requirement, and eventually, limiting the query results to the specific role assigned to that token during its creation. Every JWT has a time to live that is configured at the server side. After this time, the token is not valid anymore and the user will have to resend its credentials to the server to obtain a new valid one. Communication with the API is done over HTTPS protocol. The framework developed is flexible enough to support other security mechanisms.

In addition to this, a list of allowed IPs is implemented. Only listed IP addresses will be allowed by the API to obtain a token, and any other IP addresses will not be able to communicate with the API in any way.

Multi-linguality

The DRR Data Ecosystem services can be scaled to a global user base supporting access to resources in different languages. Despite the fact that current data being used in the S4S platform is available as English as the main Language, the framework is developed with flexibility to support multilingual accessibility of future multilingual resources. As we can observe from Figure 9, a language field is added to the Assets of the ecosystem, thus allowing the client systems querying data in a specific language.

Other data sources support

During the iterative development of the Data Ecosystem, REST APIs from the external sources were developed in parallel to enable data pipeline automation. Until these APIs were developed and deployed, data was provided in other different formats, such as spreadsheets and JSON files.

We believe this is a key aspect to highlight in this document, since other projects with different IT maturity levels might be interested in sharing data within the data ecosystem, but might not have a web data interface in place, such as REST APIs. In this case, the framework developed allows for integration of this datasets, without affecting the already connected data sources.

Conclusion and Discussion

We have presented the DRR Data Ecosystem developed, providing technical and non-technical descriptions of the development process and the solution. We have provided a series of examples depicting the main features of the proposed solution and its benefits.

Future work can include the addition of a module of Artificial Intelligence that will allow for automatic mapping of the assets to the S4S Taxonomy in the cases where the lack of metadata around the resources is scarce. The module can utilize natural language processing techniques to learn from already mapped resources and their content, and use this knowledge to map other resources or assets.

Acknowledgement

The authors would like to thank each of the contributors and their respective projects. The undertaking represents a significant step forward to how EU operational security projects are delivered and believe that it will act as a beacon for future Horizon Europe proposals and projects.

Change history

Issue	Author	Organisation	Issue Date
1.0	Agustin Garcia Pereira	Insight SFI Research Centre for Data Analytics, Ireland.	30th June 2021
	Tom Flynn	TFC Research and Innovation Limited, Ireland.	

References

[1] F. Doglio, *REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development*. 2018.

[2] G. Brito and M. T. Valente, "REST vs GraphQL: A controlled experiment," *Proc. - IEEE 17th Int. Conf. Softw. Archit. ICSA 2020*, no. Dcc, pp. 81–91, 2020, doi: 10.1109/ICSA47634.2020.00016.

[3] A. Chammas, M. Quaresma, and C. Mont'Alvão, "A Closer Look on the User Centred Design," *Procedia Manuf.*, vol. 3, no. December, pp. 5397–5404, 2015, doi: 10.1016/j.promfg.2015.07.656.

Errata

None at this point.

Appendix

GraphQL Schema

```
directive @cacheControl(
  maxAge: Int
  scope: CacheControlScope
) on FIELD_DEFINITION | OBJECT | INTERFACE

type Query {
  assets: [Asset]
  filterAssets(domain: String, type: String, topic: String): [Asset]!
}

type Asset {
  # A unique string that will identify each Asset independently.
  id: String

  # A name given to the Asset
  title: String

  # A description of the Asset
  description: String

  # An alternative name for the Asset
  altLabel: String

  # Date of formal issuance (e.g., publication) of the Asset
  issued: String

  # Date of the latest update of the Asset
  modified: String

  # A set of words, phrases or tags to describe the Asset
  keyword: [String]

  # A version number or other designation of the Asset
  versionInfo: String
  licenseDocument: LicenseDocument
  publisher: Publisher

  # Any kind of URL that gives access to an Asset e.g. a landing page, download, feed URL, SPARQL
endpoint etc.
  accessURL: String

  # This is a direct link from an Asset to a downloadable file in a given format, e.g. CSV file or RDF
file
  downloadURL: String
  s4s_taxonomy: S4S_taxonomy
}
```

```
type S4S_taxonomy {
  # Stairs4Security Taxonomy. Domain represents one of the following categories: Detection, etc.,
  domain: [String]
  topic: String
  type: String
}

type LicenseDocument {
  title: String
  description: String
  type: String
}

type Publisher {
  id: String
  preferredLabel: String
  alternativeLabel: String
  description: String
}
```