The Food Safety Market: An SME-powered industrial data platform to boost the competitiveness of European food certification

D2.2. – Data Services

DELIVERABLE NUMBER	D2.2
DELIVERABLE TITLE	Data Services
RESPONSIBLE AUTHOR	Svetla Boytcheva (SAI)

Co-funded by the Horizon 2020
Framework Programme of the European Union

GRANT AGREEMENT N.	871703
PROJECT ACRONYM	TheFSM
PROJECT FULL NAME	The Food Safety Market: An SME-powered industrial data platform to boost the competitiveness of European food certification
STARTING DATE (DUR.)	01/02/2020 (36 months)
ENDING DATE	31/01/2023
PROJECT WEBSITE	www.foodsafetymarket.eu
COORDINATOR	Nikos Manouselis
ADDRESS	110 Pentelis Str., Marousi, GR15126, Greece
REPLY TO	nikosm@agroknow.com
PHONE	+30 210 6897 905
EU PROJECT OFFICER	Stefano Bertolo
WORKPACKAGE N. | TITLE	WP2 | Data
WORKPACKAGE LEADER	Sirma AI
DELIVERABLE N. | TITLE	D2.2 | Data Services
RESPONSIBLE AUTHOR	Svetla Boytcheva (SAI)
REPLY TO	svetla.boytcheva@ontotext.com
DOCUMENT URL	www.foodsafetymarket.eu
DATE OF DELIVERY (CONTRACTUAL)	31 January 2021 (M12)
DATE OF DELIVERY (SUBMITTED)	31 January 2021 (M12)
VERSION | STATUS	V 1.0 | Final
NATURE	Demonstrator (DEM)
DISSEMINATION LEVEL	Public (P)
AUTHORS (PARTNER)	Svetla Boytcheva (SAI), Plamen Tarkalanov (SAI), Nikola Tulechki (SAI), Pavlin Gyurov (SAI), Nikola Rusinov (SAI)
CONTRIBUTORS	
REVIEWER	Dimitris Fotakidis (Agroknow)

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	Table of contents	15/11/2020	Svetla Boytcheva (SAI)
0.2	Initial draft version	15/12/2020	Plamen Tarkalanov (SAI), Pavlin Gyurov (SAI), Nikola Rusinov (SAI)
0.3	Sections contribution, corrections, and observations	21/01/2021	Plamen Tarkalanov (SAI), Nikola Tulechki (SAI), Pavlin Gyurov (SAI), Nikola Rusinov (SAI)
0.8	Document refinement, addition of figures and descriptions	28/01/2021	Svetla Boytcheva (SAI)
0.9	Internal Review	28/01/2021	Dimitris Fotakidis (Agroknow)
1.0	Final Version	31/01/2021	Svetla Boytcheva (SAI)

PARTNERS		CONTACT
Agroknow IKE (Agroknow, Greece)	Agroknow	Nikos Manouselis (Agroknow) nikosm@agroknow.com
SIRMA AI EAD (SAI, Bulgaria)	ontotext	Svetla Boytcheva (SAI) svetla.boytcheva@ontotext.com
GIOUMPITEK MELETI SCHEDIASMOS YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS (UBITECH, Greece)	UBITECH ubiquitous solutions	Danai Vergeti (UBITECH) vergetid@ubitech.eu
AGRIVI DOO ZA PROIZVODNJU, TRGOVINU I USLUGE (Agrivi d.o.o., Croatia)	AGRIVI	Tanja Matosevic (Agrivi d.o.o.) tanja.matosevic@agrivi.com
PROSPEH, POSLOVNE STORITVE IN DIGITALNE RESITVE DOO (PROSPEH DOO, Slovenia)	tracelabs origintrail Core Developers	Ana Bevc (PROSPEH DOO) ana.bevc@tracelabs.io
UNIVERSITAT WIEN (UNIVIE, Austria)	universität wien	Tima Anwana (UNIVIE) tima.anwana@univie.ac.at
STICHTING WAGENINGEN RESEARCH (WFSR, Netherlands)	WAGENINGEN UNIVERSITY & RESEARCH	Yamine Bouzembrak (WFSR) yamine.bouzembrak@wur.nl
TUV- AUSTRIA ELLAS MONOPROSOPI ETAIREIA PERIORISMENIS EUTHYNIS (TUV AU HELLAS, Greece)	TÜV AUSTRIA HELLAS	Kostas Mavropoulos (TUV AU HELLAS) konstantinos.mavropoulos@tuv.at
TUV AUSTRIA ROMANIA SRL (TUV AU ROMANIA, Romania)	TÜV AUSTRIA ROMANIA	George Gheorghiou (TUV AU Romania) george.gheorghiu@tuv.at
VALORITALIA SOCIETA PER LA CERTIFICAZIONE DELLE QUALITA'E DELLE PRODUZIONI VITIVINICOLE ITALIANE SRL (VALORITALIA, Italy)	VALORITALIA	Francesca Romero (Valoritalia) francesca.romero@valoritalia.it
TUV AUSTRIA CYPRUS (TUV AU CYPRUS, Cyprus)	TÜV AUSTRIA CYPRUS	Sousanna Charalambidou (TUV AU CYPRUS) sousanna.charalambidou@tuv.at

ACRONYMS LIST

The Food Safety Market	TheFSM
API	Application Programming Interface
CSV	Comma-separated values
HTTP	Hypertext Transfer Protocol
GREL	Google Refine Expression Language
JSON	JavaScript Object Notation
RDF	Resource Description Framework
RDBMS	Relational Data Base Management System
REST	Representational state transfer
SHACL	Shapes Constraint Language
SKOS	Simple Knowledge Organization System Primer
SOML	Semantic Objects Modelling Language
SPARQL	SPARQL Protocol and RDF Query Language
XML	Extensible Markup Language
WKT	Well-known text

EXECUTIVE SUMMARY

This document presents the initial version of the data services developed within The Food Safety Market (TheFSM) project – data ingestion service; data curation service and data publishing service. There are included brief description of the main functionalitites for each service, example illustrated with screenshots and discussion about the main tools and technologies used in the implementation. The developed services are based on the detailed analysis of business, data and technical requirements of all use cases. The best practices and standards are taken in consideretion as well. This document will be updated twice at M24 and M36 to show the progress in terms of data services fine tuning according to the TheFSM project need.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

The Food Safety Market (TheFSM) project aims building a full stack of data management services in order to equipt the TheFSM platform with full functionalities for data ingestion; data curation, reconciliation and data publishing data. This will allow the TheFSM platform users easily to tackle the problems that tipically araise with heterogenous data integration.

The main puprose of the Data Ingestion service is to implement semantically based services that will make uploading, mapping and data transformation easier and user friendly for different stakeholders.

The main puprose of the Data Curation service is to implement an advanced storage environment to make life easier for domain experts, when they perform tasks like deduplication, enrichment, mapping, entity matching, cleaning, etc .

The main puprose of Data Publishing Services is to provide a data publishing gateway that will enhance the TheFSM platform with more sophisticated semantically driven services for data providers and APIs for data consumers.

The developed services are based on the detailed analysis of business, data and technical requirements of all use cases described in D1.1 “Report on Requirements for TheFSM”, in compliance with D1.2 “TheFSM Development Roadmap”, D2.3 “Report on Data Population” and D3.1 “TheFSM Open Reference Architecture”. The best practices, state-of-the-art and standards are taken in consideretion as well.

2 DATA INGESTION SERVICES

2.1 Data ingestion using OntoRefine

GraphDB[1] OntoRefine is an upgraded version of the open source OpenRefine[2] data transformation tool. It allows the quick mapping of any structured data to a locally stored RDF[3] schema in GraphDB. The visual interface is optimized to guide you in choosing the right predicates and types, defining the datatype to RDF mappings, and implementing complex transformation using OpenRefine's GREL[4] language. GREL is the Google Refine Expression Language that helps define complex transformation.

2.1.1 Data ingestion example

The following screen shows a visual mapping of tabular data to RDF. Here the example data is about restaurants, with various properties such as the name (in multiple languages) and location (expressed as WKT litterals) mapped using the relations required by the chosen ontology.

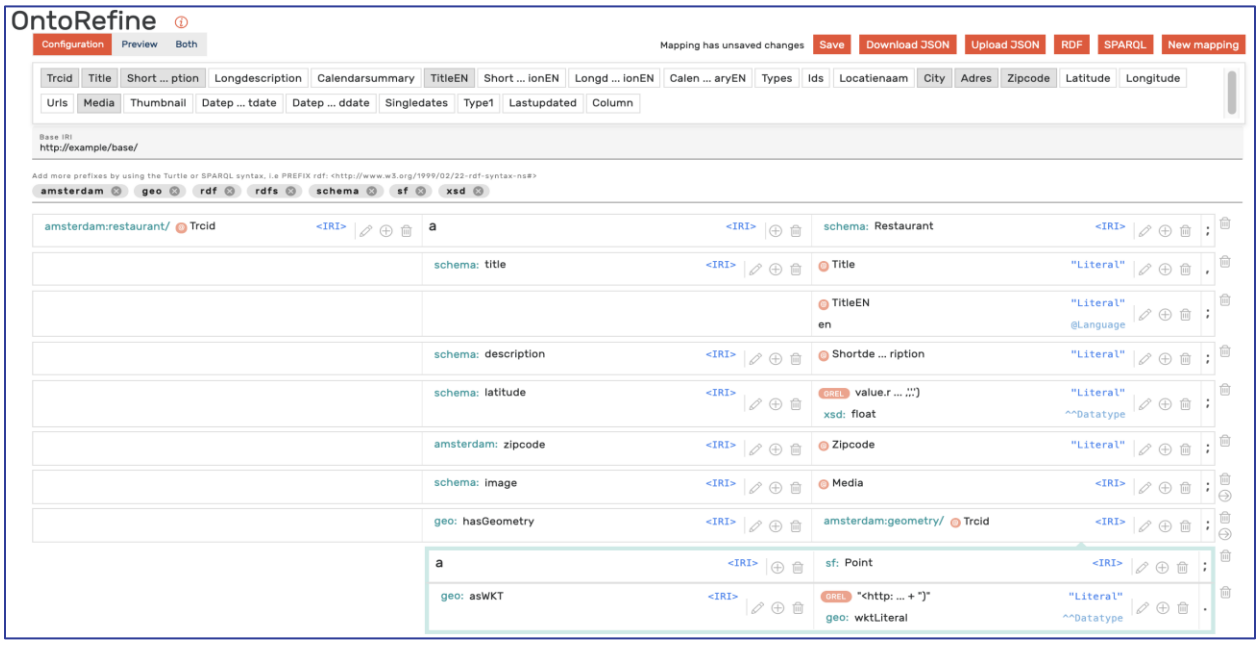

Figure 1 Example for data ingestion – visual mapping of tabular data to RDF

[1] https://www.ontotext.com/products/graphdb/
[2] https://openrefine.org/
[3] https://www.w3.org/RDF/
[4] https://guides.library.illinois.edu/openrefine/grel#:~:text=Google%20Refine%20Expression%20Language%20(GREL)%20is%20to%20OpenRefine%20what%20formulas,column%20based%20on%20another%20column

This visually defined mapping result in the following RDF:

```
@base <http://example/base/> .

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@prefix schema: <http://schema.org/> .

@prefix geo: <http://www.opengis.net/ont/geosparql#> .

@prefix amsterdam: <https://data/amsterdam/nl/resource/> .

@prefix sf: <http://www.opengis.net/ont/sf#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .


<https://data/amsterdam/nl/resource/restaurant/669d7d82-8962-4e88-b2e1-
7b8706633aa0>

  a schema:Restaurant;

  schema:title "Smits Noord-Zuid Hollandsch Koffiehuis", "Smits Noord-Zuid
Hollandsch Koffiehuis"@en;

  schema:description "Het Smits Koffiehuis ontleent haar ontstaan aan de
stoomtram die de verbinding onderhield met Amsterdam naar het noorden van de
provincie en is in 1919 gebouwd. Nu is er een restaurant en een koffiebar. Ook
is hier een informatiekantoor van Amsterdam Marketing gehuisvest.";

  schema:latitude "0"^^xsd:float;

  amsterdam:zipcode "1012 AB";

  schema:image
<https%3A//media.iamsterdam.com/ndtrc/Images/20101122/ec8faec5-5cd5-43d6-
b0fa-eb0dab65e278.jpg>;

  geo:hasGeometry <https://data/amsterdam/nl/resource/geometry/669d7d82-8962-
4e88-b2e1-7b8706633aa0>;

  amsterdam:uniquelocation _:node1em9j7qmhx179149;

  amsterdam:valuelocation _:669d7d82-8962-4e88-b2e1-7b8706633aa0 .


<https://data/amsterdam/nl/resource/geometry/669d7d82-8962-4e88-b2e1-
7b8706633aa0>

  a sf:Point;
```

```
  geo:asWKT "<http://www.opengis.net/def/crs/OGC/1.3/CRS84> POINT (4.9003230
52.3775440)"^^geo:wktLiteral .


_:node1em9j7qmhx179149 amsterdam:address "Stationsplein 10" .


_:669d7d82-8962-4e88-b2e1-7b8706633aa0 amsterdam:city "AMSTERDAM" .
```

3 DATA CURATION SERVICES

3.1 Data curation using OntoRefine

Besides defining and running transformations, OntoRefine is also used for data cleaning and curation. Multiple functionalities such as faceted exploration of tables, bulk editing and a powerful expression language (GREL) allowing complex string manipulation on individual values allows easy finding and fixing of errors in the input data,

3.1.1 Data curation and cleaning examples

The following screenshots show some of the most commonly used data cleaning functionalities of OpenRefine. here are some examples:

3.1.1.1 Facets

Faceted exploration allows to see at a glance all unique values in a column, and to filter based upon them.

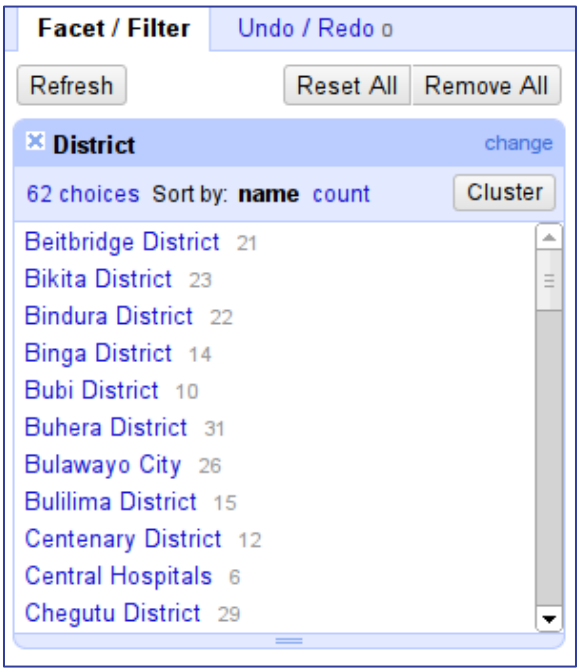

Figure 2 Example for data curation – faceted search

3.1.1.2 Common string transformations

Predefined transformations exists for most of the common tasks, such as whitespace trimming, switching data types, case normalisation etc...

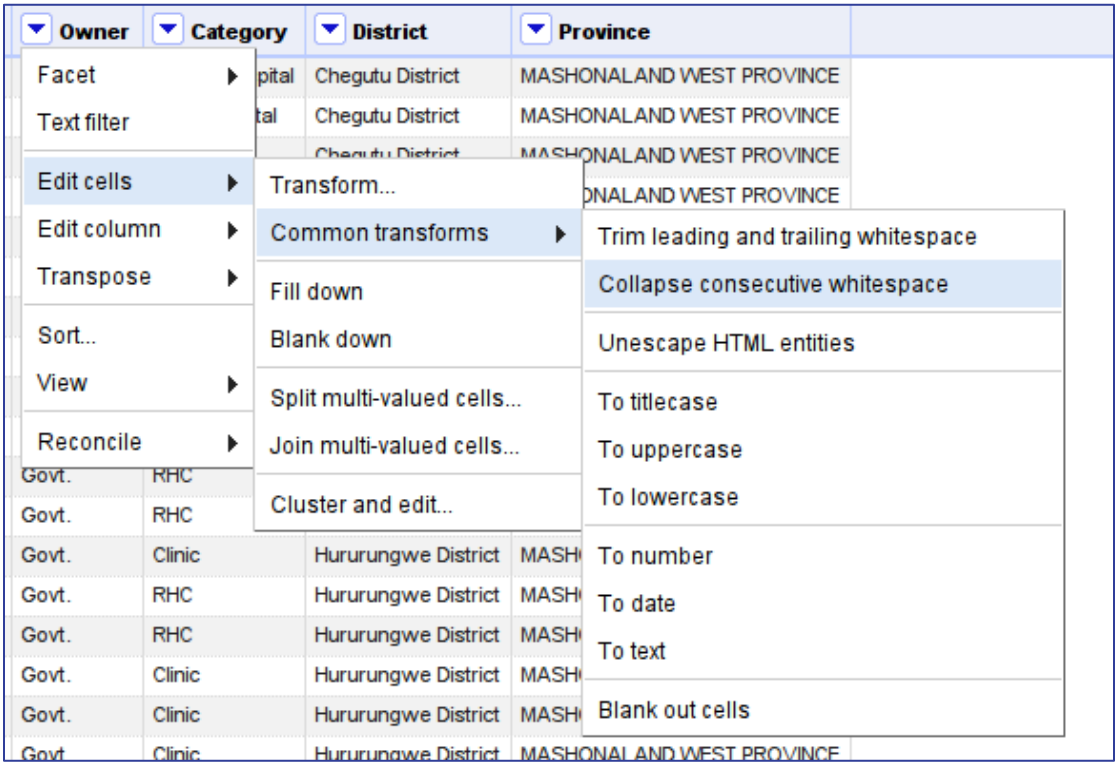

Figure 3 Example for data curation – text transformation

3.1.1.3 Text clustering

Text clustering allows to folding similar values in a supervised manner thus reducing noise in the data.

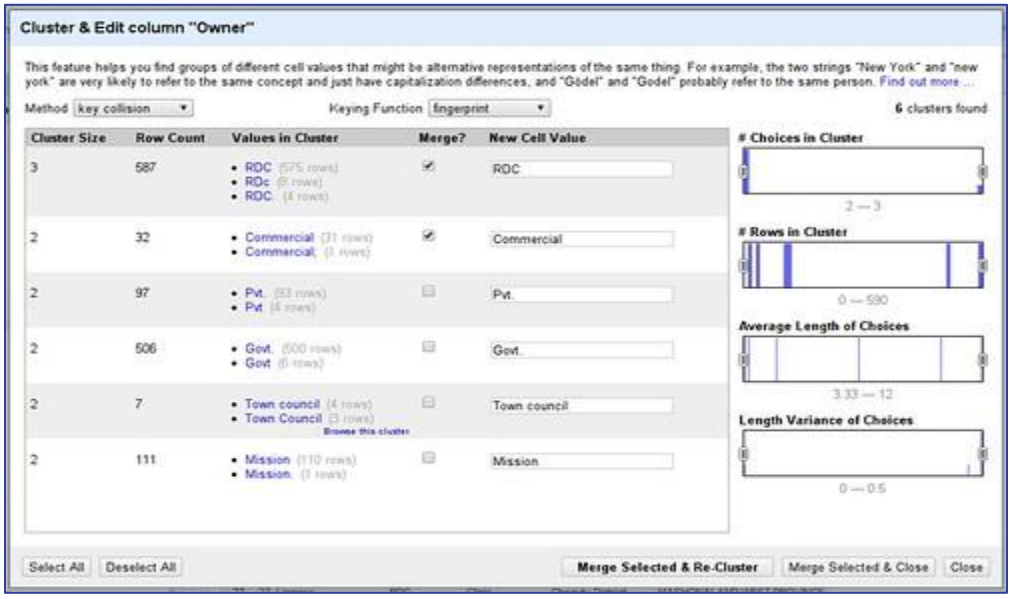

Figure 4 Example for data curation – text clustering

3.2 Reconciliation Service

Besides data cleaning, OntoRefine is also a reconciliation client, following the "Reconciliation API" protocol[5]. Reconciliation or automatic entity matching is a semi-automated process of matching text names to database IDs (keys).

The following screenshot shows the process applied to Entities of type "person". The "itemLabel'' column initially contains names of people as strings. The reconciliation service has searched for these names on a remote endpoint (in this case Wikidata[6]) and proposes a number of candidates for each string.

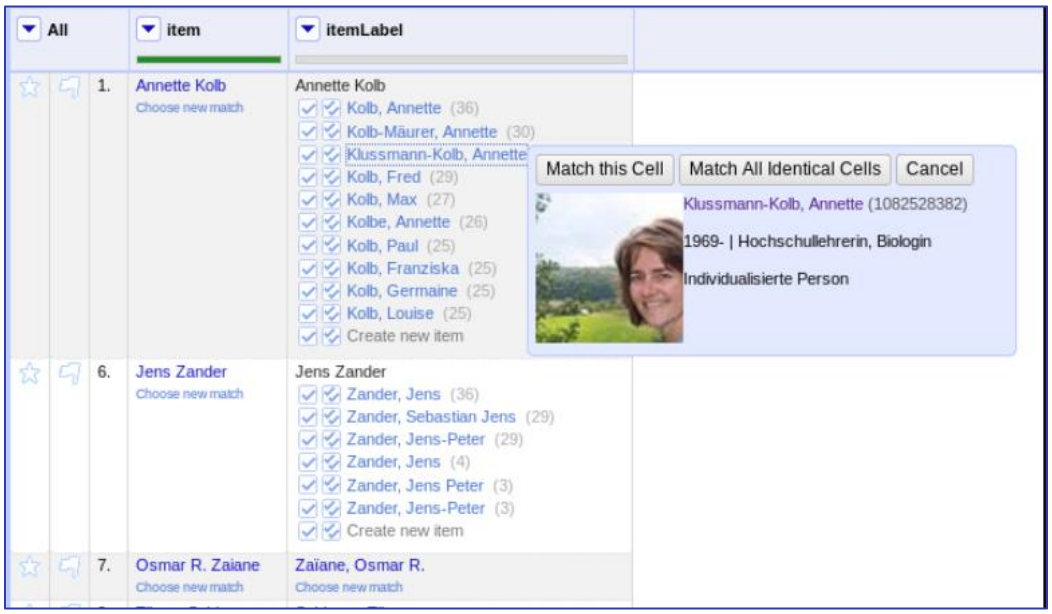

Figure 5 Example for reconciliation service

3.2.1 TheFSM Reconciliation Service

Besides using readily available public reconciliation endpoints (such as Wikidata) we will also develop a framework for setting up and running project specific endpoints corresponding to Food Safety relevant objects. These will be entities such as products, fertilizers, incident categories, and certificate types and families. The service will run on top of the public subset of the semantic data collected and will streamline the process of mapping and submitting new datasets to the TheFSM.

[5] https://reconciliation-api.github.io/specs/latest/
[6] https://www.wikidata.org/wiki/Wikidata:Main_Page

4 DATA PUBLISHING SERVICES

4.1 GraphQL API

GraphQL[7] is a set of a query-by-example type language allowing users to make queries and server-side tools providing data or executing data mutations in correspondence with the user queries. The queries are passed to the services (API) as plain texts. The responses are JSON structures, following the query structure. The field names and data types are as per preliminary defined data description (also known as schema) and as a rule the field names are quite informative making the queries and results also human readable.

More detailed information about the language, its syntax and semantics, etc. can be found at http://graphql.org

4.1.1 Why GraphQL

GraphQL is not closely tied to any specific data storage, data source or programming language. It provides the abstraction of data source moving the focus from databases, web services, data types, class inter-relations, indices and any other specifics and details to the data themselves and provides users with an unified interface for dealing with data. The only thing the user has to know is that there are such data classes or concepts with their hierarchy, maybe relations to some other concepts and what are the names of the fields he/she is interested in. Where the data (objects) of such classes are stored and how they can be accessed is up to the system and is generally completely hidden from the users.

Each GraphQL service or also GraphQL endpoint has a description of data classes (concepts) which it manipulates. It is also known as schema. The schema consists of description of fields, their types and the names of the functions which can be called in order the field (property) values to be accessed. Additionally, behind the scene, there are also implementations of these functions in some programming languages, packed as libraries or services and closely connected to the specific data storage and its functionality.

The specific implementation of a GraphQL service or also GraphQL endpoint depends on the storage system of the data which it manipulates, however, the schema is public and the user can see the available data objects description so that to be able to construct the appropriate GraphQL queries.

[7] https://graphql.org/

Since the real organization of data is hidden, it is completely valid for some predefined queries to be delivered as data classes and data to be accessed by the users in a straightforward manner as in the case they exist as physical data objects.

This way the GraphQL endpoint becomes an abstraction layer between the data and business logic providing consistency, isolation and standardization.

4.2 Data Federation

The proposed Apollo Federation Service[8] as a part of the Data Services module of TheFSM Platform behaves as a GraphQL endpoint.

It allows gathering data from different GraphQL endpoints as per definitions in its schema, combining their schemes and providing users with a solid holistic data source.

Specific predefined queries can be defined and added to its combined data scheme so that the users can be supported in their everyday activities without the need of using very complex queries.

In the context of TheFSM Data Services the usage of GraphQL allows development of a standardized way for requesting data and data publishing.

The main GraphQL endpoint is the semantic-service which uses the data stored in the GraphDB semantic repository. Its schema is defined in terms of Semantic Objects Modelling Language[9] (SOML). The service can be queried using the classes and field names from the schema. It is provided as a web service providing REST API and can be called separately or via the Apollo Federation Service.

4.3 Ontotext Platform Workbench

This component is a user interface and is a kind of playground where users can explore the semantic schema and view different data classes provided by TheFSM platform.

[8] https://www.apollographql.com/docs/federation/
[9] https://platform.ontotext.com/3.0/soml/index.html

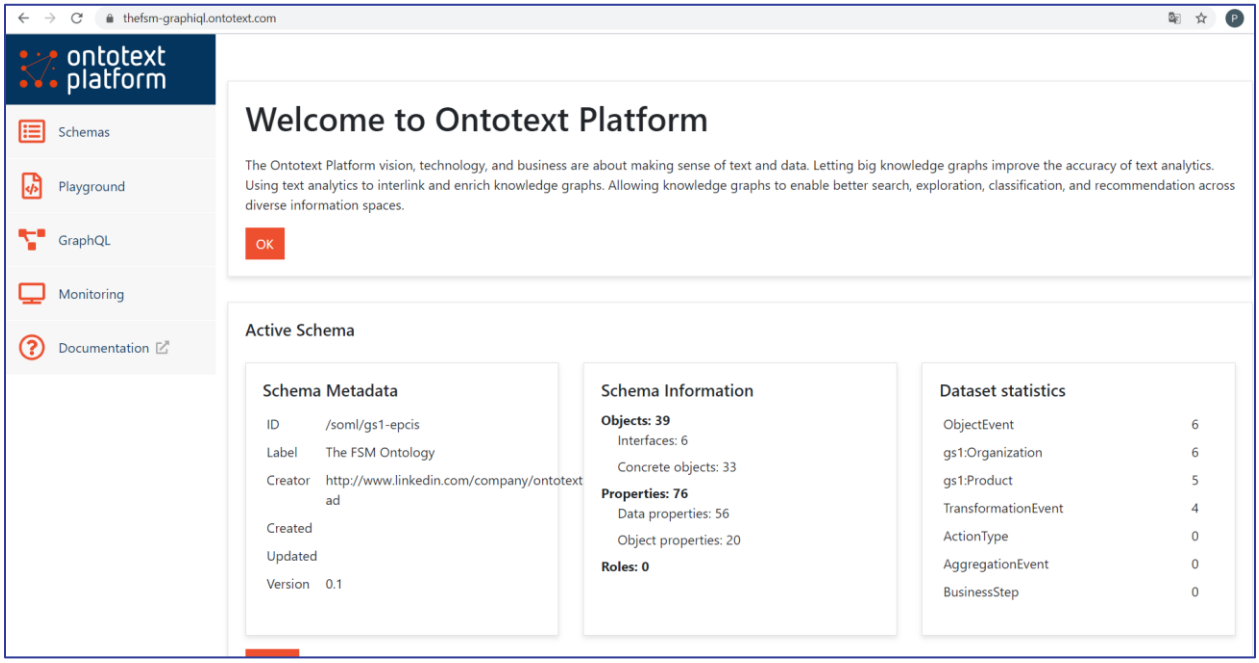

Figure 6 Ontotext Platform – Welcome view

It is designed as an entry point for schema management, monitoring and an interface where users can try queries and explore their results.

The Schemas tab allows users to switch between loaded schemas, to load new ones and in general to manage the used schemas by the workbench.

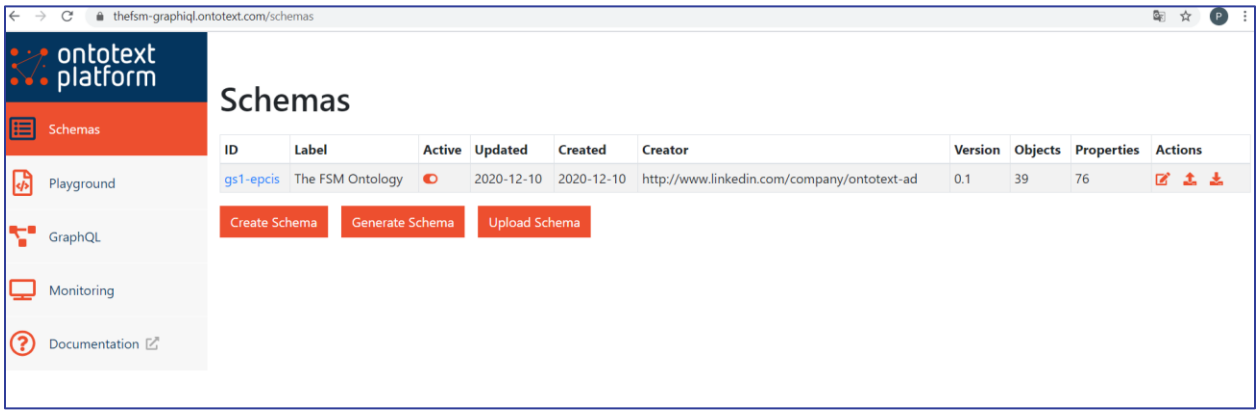

Figure 7 Ontotext Platform - Schemas

The Playground tab provides functionality for schema modification and validation. It acts as a regular text editor for usage with codes.

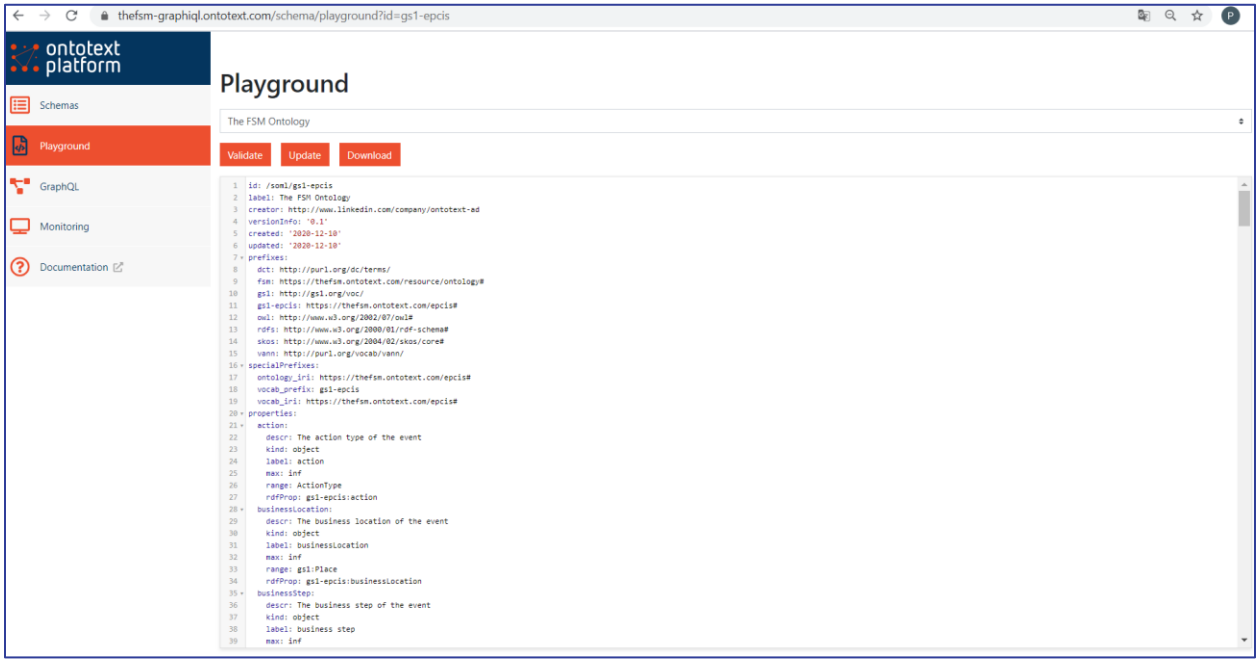

Figure 8 Ontotext Platform - Playground

The Monitoring tab shows the current state of the platform components, A status line is present on the top of the screen just below the screen title. Its background color can be green, yellow or red depending on the state of the platform as a whole. This allows one sight checks - if the colour is green - all is OK and there is no need of further elaboration.

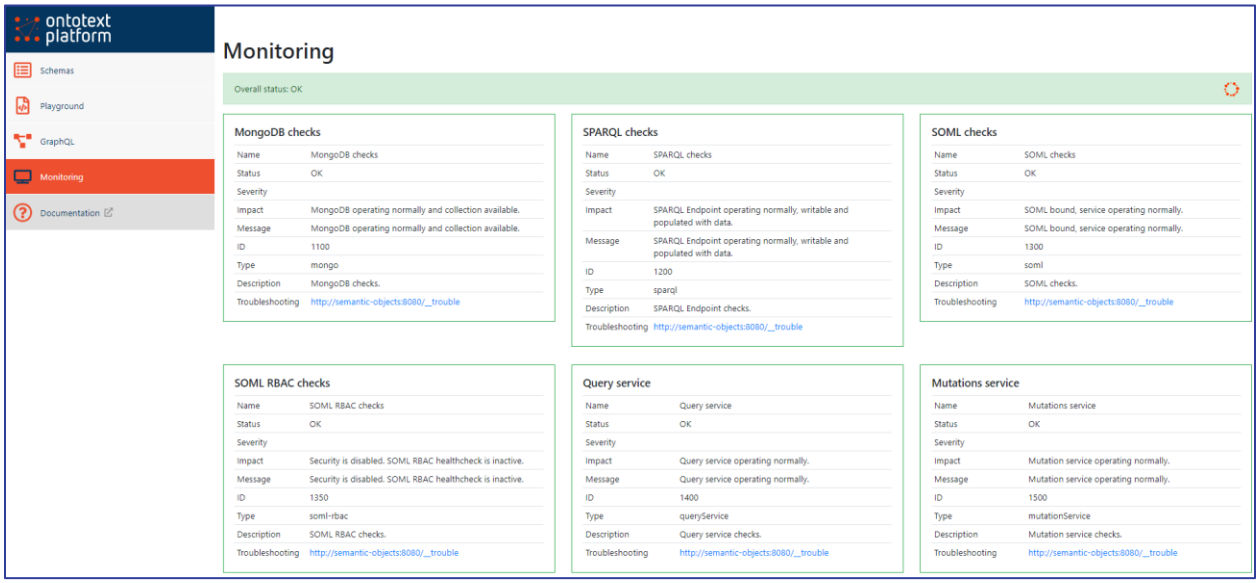

Figure 9 Ontotext Platform - Monitoring

One of the most important features of the platform workbench is in the tab GraphQL. It provides a user interface for writing and execution of GraphQL queries. That is useful for creation and testing of queries which to be included in the user code for querying the services programmatically.

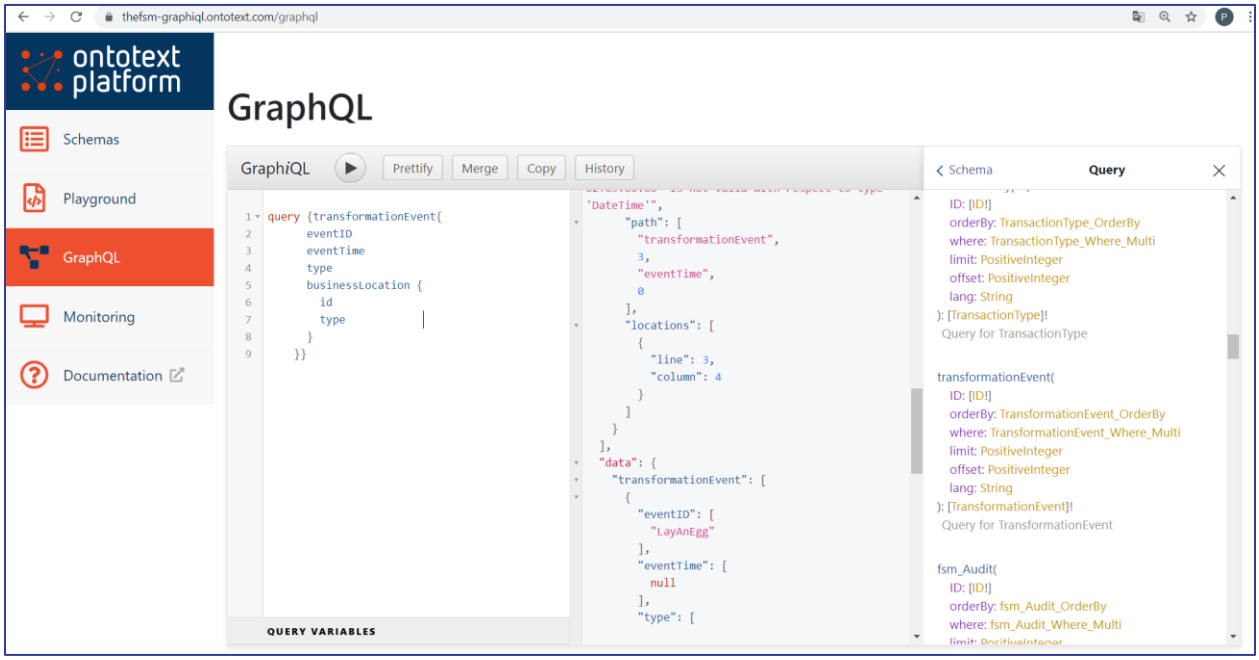

Figure 10 GraphQL

There are three user interface panes. The user can write a GraphQL query in the left pane, using the data definitions in the rightmost pane. There are also context related hints about data classes and property (field) names. The results of running the query are displayed in the central area.

The query from the above example returns the list of transformation events from the loaded sample data objects in the semantic repository. As one can see, the results are returned in JSON format following the structure of the query.

There is also rich documentation content in the Documentation tab. Plenty of information about the modules of the platform, their usage and interconnections is available in there.

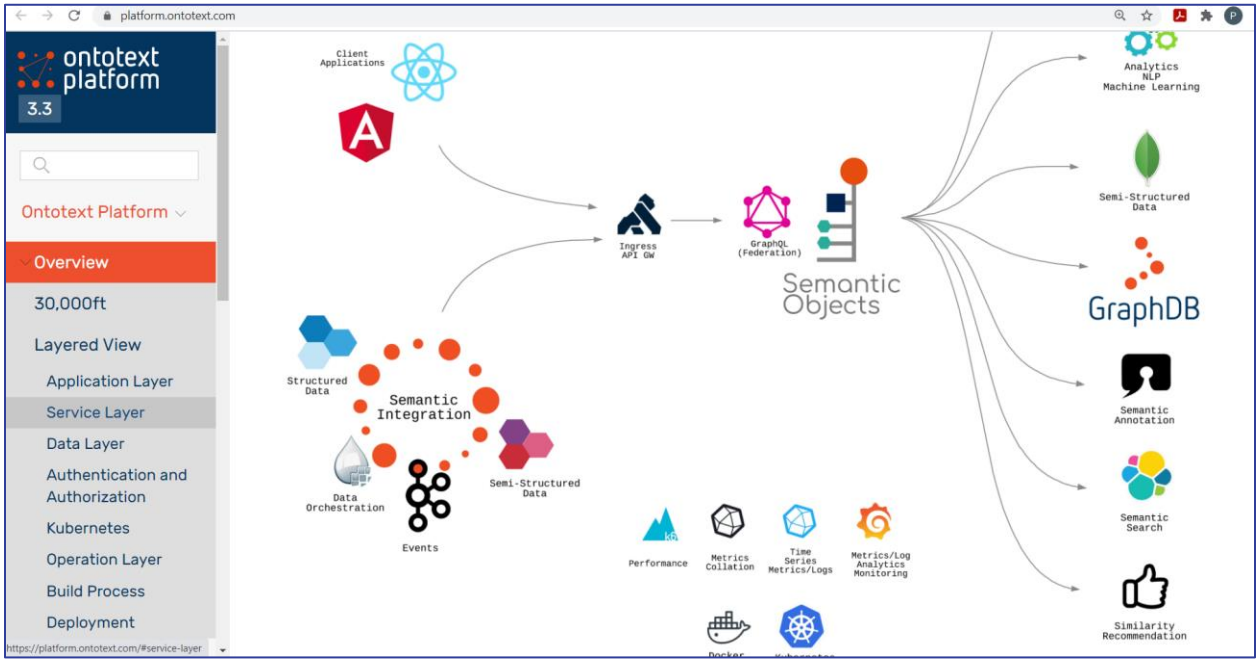

Figure 11 Ontotext Platform - overview

4.4 GraphQL endpoint

The same result can be obtained by an application when sending the following HTTP request to the service.

```
curl -X POST https://thefsm-graphql.ontotext.com/graphql    \
      -H "Content-type: application/json"                        \
      --data-binary '{"query":                                   \
            "query {transformationEvent{                        \
                  eventID                                       \
                  eventTime                                     \
                  type                                          \
                  businessLocation {                            \
                        id                                      \
                        type                                    \
                  }                                             \
            }}"}'                                               \
```

The above example is provided for the `cUrl` command line tool. Every developer can construct the corresponding HTTP requests for the program language he uses following the above pattern,

In other words, when a developer have his desired query tested using the GraphQL tab of the Platform workbench, the of the query is placed in the "query" field of the JSON, which the program will send as a body of the http request to the GraphQL endpoint.

Important remark: Since the currently provided data services are running on Sirma AI internal infrastructure, which will not be the final one, the server address will be changed when Data Services are moved to AWS infrastructure[10].

4.4.1 Other usable API calls

`curl -X GET https://thefsm-graphql.ontotext.com/soml` – returns the schema of the GraphQL endpoint

`curl -X GET https://thefsm-graphql.ontotext.com/ gtg` – returns "Good to Go" status, e.g. `{"gtg":"OK"}`

`curl -X GET https://thefsm-graphql.ontotext.com/ health` returns more detail status information of the Ontotext Platform and its components

4.5 Data Publishing Workflow

The TheFSM project is data centric. All the user stories include data transfers from one user to another and most of them require data enrichment and disambiguation against Linked Open Data[11] sources, datasets from other users and mapping to the TheFSM data model (described in D2.1).

The Data Services provide unified data access via GraphQL queries, federating among the various data sources.

From the viewpoint of data persistence, there are two major super-types of data:

- **Stored static data** - they are not expected to be changed during the lifecycle of the project and are used frequently. They will be stored in the Secure data storage. All mappings, enrichments, etc. will be performed on the stage of initial data transformation before storing. uch data are:
  - ontologies, vocabularies, etc.
  - public static datasets which do not require frequent updates.
  - Static published datasets from specific users.
- **Realtime data**. These data will be achieved, enriched, and mapped at the realtime when requested by customer queries. Such data types are:
  - Data provided by various applications,
  - LOD,
  - Data with restricted access, which depends on the querying end user credentials. Such data cannot be imported in the system.
  - Published data from users via providing access to their internal systems

[10] https://aws.amazon.com/about-aws/global-infrastructure/#:~:text=The%20AWS%20Global%20Infrastructure%20gives,the%20AWS%20Regions%20and%20AZ's.
[11] https://lod-cloud.net/

Data Services also give a standardized way for publishing data. The users of the platform can be broadly divided to two major types:

- Data consumers - users who consume data from the platform using provided functionality
- Data providers - parties that provide their data to be used by the consumers via the platform functionality

Any provided dataset must be described and its structure must be added to the platform data schema in order to be accessible by Semantic Objects service. This means that any single data source requires some efforts in order to be integrated with the other part of the platform.

As already mentioned in the case of static data, which will be stored inside the platform all necessary enrichments, mappings into the ontology concepts, disambiguation, etc is performed at the initial processing time. The data structure is evaluated and data description is prepared. The specific data schema descriptors are created and added to the semantic schema of the platform. The data itself is processed, mapped and linked to the concepts of the ontology, LOD, etc. The result of the process is rdf representation of data, subsequently added to the repository

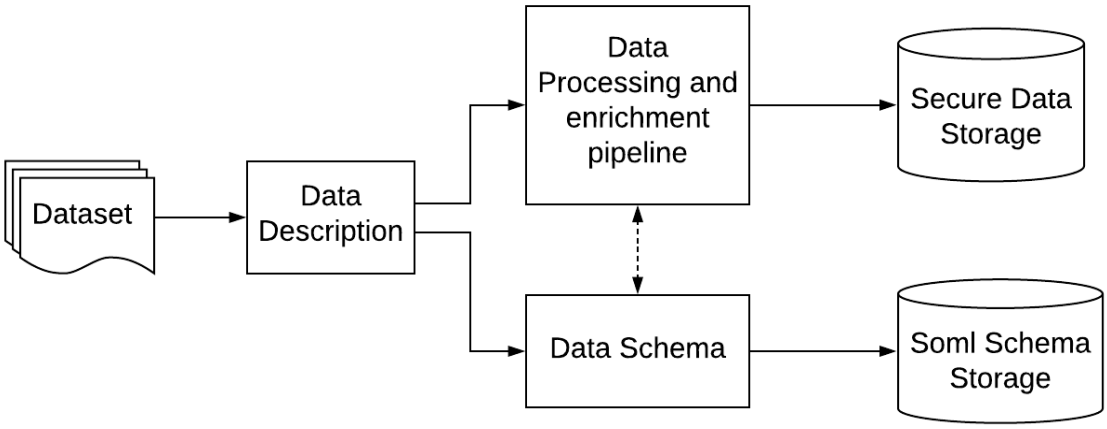

Figure 12 Static data publishing workflow

In case of a real-time data source, there is no data, but the description of the provided API (if it is not a federation-ready GraphQL endpoint) is studied. Data Schema extensions are created and added to the federation service schema. If the provided API is from a GraphQL federation-ready endpoint, only its data description is added to the federation service schema.

However, in the common case a GraphQL endpoint must be implemented as a data source wrapper in order the functionalities for accessing data from the external data source to be presented in a standard GraphQL manner and ready for use by the Apollo Federation service. It is added to the set of microservices running on the infrastructure.

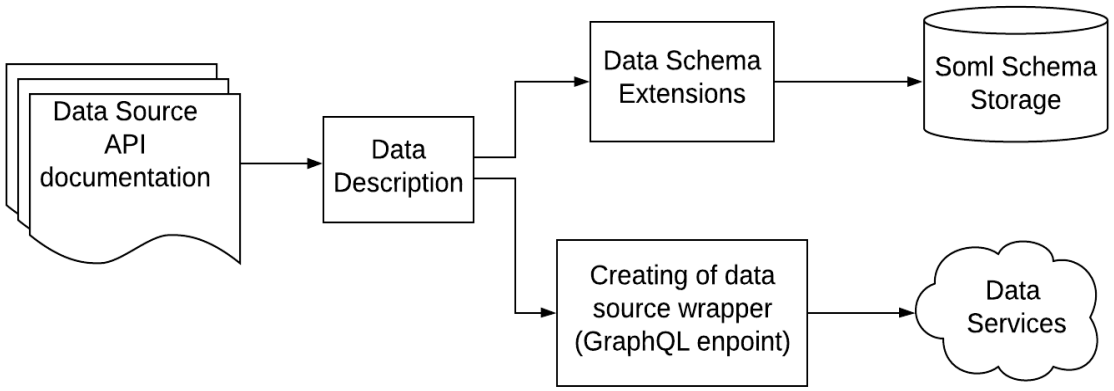

Figure 13 Realtime data publishing workflow

As it can be seen, adding a new data source, especially a data source with dynamic content implies some efforts - data analysis, writing schema extensions and some programming - creating the corresponding wrapper microservice or the processing pipeline, accordingly.

As it can be seen below, adding a static dataset can be semi-automated reducing the efforts to a few hours.

4.6 Dynamic publishing of domain specific ontologies

The principle is as follows. The implementation resides in the project's github[12]. Master vocabulary data resides in a shared Google Sheet. Google Sheets is a wonderful tool perfect for handling many of the difficulties of remote collaboration:

- It relieves us of the responsibility of monitoring data integrity and data versioning.
- it maintains a complete log of all user actions.
- It's simple to revert to the previous version or assign an error to a user.
- This allows us to be sufficiently agile while ensuring that the final data meets the rigorous consistency required to generate RDF.
- it has a collaboration mechanism with comments and tasks on specific cells allowing fast and efficient remote communication on specific data elements.
- linking with cells and ranges also facilitates communication.

Google sheet data is consumed over HTTP in a simple CSV tabular format and is converted to RDF using a custom SPARQL query and the TARQL[13] tool. TARQL (or tabular sparql) allows us to define a tabular to RDF mapping in a SPAQRL query. This allows us to maintain the mappings in a fully

[12] https://github.com/OriginTrail/epcis-erm/tree/food-auth/vocab
[13] https://tarql.github.io/

declarative way by having the various elements independent of each other and in version control (github)

The generated ontology in RDF is also committed to the github repository and can be directly consumed via HTTP.

4.7 Data validation using RDF Shapes

Another aspect of publishing data through the TheFSM platform is assuring their consistency before they are available to data consumers.

Since the problem of data validation before integration is very common worldwide, a special experts group of W3C was constituted. Its name is RDF Data Shapes Working Group[14]. It created Shapes Constraint Language[15] (SHACL) as a recommendation for description of data integration and validity rules (called also *shapes*). More details about the language and ways of description of constraints can be found here[16].

SHACL validation is incorporated in the latest version of GraphDB. Detailed information about setup and usage is available under Documentation Tab, Usage section, SHACL validation.

In a very few words, this functionality provides a tool for validating graphs against predefined set of rules, in the form of shapes or other constructs and loaded in as separate RDF graphs. They are also called *shape graphs*.

Formally, each shape is an IRI or a blank node `s` that fulfills at least one of the following conditions in the shapes graph:

- s type is one of `sh:NodeShape` or `sh:PropertyShape`.
- s is subject of a triple that has one of the following predicates: `sh:targetNode`, `sh:targetClass`, `sh:targetObjectsOf` or `sh:targetSubjectsOf`.
- `s` is the subject of a triple that has a parametric predicate.
- `s` is a value of a shape-expecting, non-list-taking parameter such as `sh:node`, or a member of a SHACL list that is a value of a shape-expecting and list-taking parameter such as `sh:or`.

Every SHACL repository contains a named graph with the reserved name `http://rdf4j.org/schema/rdf4j#SHACLShapeGraph`,
where the validation rules are inserted.

If one intends to use SHACL validation in some repository he/she must specify it on the creation stage. This cannot be added afterwards. If the workbench user interface is used some additional

[14] https://www.w3.org/2014/data-shapes/
[15] https://www.w3.org/TR/shacl/
[16] https://www.w3.org/TR/shacl/

checkboxes appear. Their detailed description and recommended values are given in the online documentation.

The SHACL rules must be loaded to the `http://rdf4j.org/schema/rdf4j#SHACLShapeGraph` named graph using one of the following methods:

- Workbench Import - as a file or RDF triples in the immediate window
- Import Statements in SPARQL window
- The GraphDB REST API
- When new data is inserted into the repository and SHACL validation is on, the RDF triples compliances to the rules are evaluated and if there is violation, an exception is thrown. Information about the violation is provided. More detailed information can be logged, depending on the repository settings.
- The supported SHACL constraints are listed in details in the official documentation[17] of GraphDB, also available online under the Documentation tab - Usage->SHACL validation

[17] https://graphdb.ontotext.com/documentation/9.5/enterprise/shacl-validation.html

5 CONCLUSION AND NEXT STEPS

In this document were presented TheFSM data services: data ingestion service; data curation service and data publishing service. The initial version was based on the state-of-the-art for such tools and services for data managementa and data integration. In addition the analyasis and prioritization of the functional and non-functional requirements served as a basis in the TheFSM data services development. More sophistacted version of these services is planned for v2.00 on M24 and v3.00 on M36, where some additional functionalities and fine-tuning of the current functionalities will be included, based on the business, data and technical requirement update.