D2.2 - EW-Shopp Platform – v1

<table>
<thead>
<tr>
<th>Deliverable n:</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>29 December 2017</td>
</tr>
<tr>
<td>Status:</td>
<td>Final</td>
</tr>
<tr>
<td>Version:</td>
<td>1.0</td>
</tr>
<tr>
<td>Authors:</td>
<td>Nikolay Nikolov, Angelo Marguglio, Aljaž Košmerlj, Matteo Palmonari</td>
</tr>
<tr>
<td>Contributors:</td>
<td>Volker Hoffman (SINTEF), Steffen Dalgard (SINTEF), Renzo Alva Principe (UNIMIB)</td>
</tr>
<tr>
<td>Distribution:</td>
<td>Public</td>
</tr>
</tbody>
</table>
## History of Changes

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>03.11.2017</td>
<td>Definition of Table of Contents</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.2</td>
<td>30.11.2017</td>
<td>Added introduction and platform architecture</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.3</td>
<td>04.12.2017</td>
<td>Added information about ArangoDB</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.4</td>
<td>05.12.2017</td>
<td>Added section for data wrangler component, overall architecture, and PI descriptions for the data hosting platform services</td>
<td>Nikolay Nikolov, Steffen Dalgard</td>
</tr>
<tr>
<td>0.5</td>
<td>06.12.2017</td>
<td>Added section on processing component</td>
<td>Volker Hoffman</td>
</tr>
<tr>
<td>0.6</td>
<td>06.12.2017</td>
<td>Added Platform API for Knowage</td>
<td>Angelo Marguglio</td>
</tr>
<tr>
<td>0.7</td>
<td>08.12.2017</td>
<td>Added sections on the Reporting Component</td>
<td>Angelo Marguglio</td>
</tr>
<tr>
<td>0.8</td>
<td>11.12.2017</td>
<td>Added ArangoDB API sub-section, added references</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.9</td>
<td>13.12.2017</td>
<td>Added improved description of RDF and ArangoDB mapping</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.10</td>
<td>15.12.2017</td>
<td>Added sections about QMiner</td>
<td>Aljaž Košmerlj</td>
</tr>
<tr>
<td>0.11</td>
<td>18.12.2017</td>
<td>Added sections for ABSTAT and ASIA. Ready for QA</td>
<td>Matteo Palmonari</td>
</tr>
<tr>
<td>1.0</td>
<td>22.12.2017</td>
<td>Addressed QA comments. Ready for submission</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>1.1</td>
<td>29.12.2017</td>
<td>Small textual edits to improve the description of ASIA</td>
<td>Matteo Palmonari</td>
</tr>
</tbody>
</table>
Executive Summary

The main technical goal of the EW-Shopp project is to build a platform to support weather and event-based analytics of data collected by companies about sales, consumer behavior, and performance of marketing campaigns. This deliverable provides a detailed description of the first prototype of the EW-Shopp platform by demonstrating how the compounding tools and services are integrated to work together to achieve the goals of the project. Furthermore, it describes the setup of the technical infrastructure and specially developed methodologies for data publishing and consumption that support the platform operations. The goal of the first prototype is to provide support for the project's business cases and, in particular, the implementation of the initial pilots, as well as to set the scope and direction of the platform development process.

Firstly, the deliverable sets the context of the first prototype with relation to the platform architecture described in deliverable D3.1. Furthermore, the document describes the technical implementation of the individual components, which is achieved by integration and extensions of a set of tools and services, which are provided by technical partners of the EW-Shopp project. The tools in question are the DataGraft platform (developed and maintained by SINTEF), ASIA and ABSTAT (developed and maintained by the University of Milano), QMiner (developed and maintained by the Joseph Stephan Institute), and the Knowage suite (developed and maintained by ENG).

- **DataGraft** and its data transformation tool Grafterizer implement the Data Wrangler component of the architecture, providing data management, data cleaning, modelling, preparation and graph transformation functionalities using user-specified transformations.

- **ASIA** is a tool for the semantic enrichment of data available in tabular formats, thus helping users of the EW-Shopp platform in integrating business data with events and weather data. Semantic reconciliation algorithms are integrated into a user interface to help users map the data schema to shared vocabularies and ontologies and link data values to shared systems of identifiers. Data enrichment widgets exploit these links to shared systems of identifiers to ease the extraction of additional data from third-party sources and their fusion into the original tabular data. **ABSTAT** is a tool to profile knowledge graphs represented in RDF based on linked data summarization mechanisms. The profiles extracted by **ABSTAT** describe the content of the knowledge graphs using abstraction (schema-level patterns) and statistics. The profiles help users understand the content of the knowledge graphs used in the platform (e.g., linked product data), support **ASIA**’s semantic reconciliation algorithms, and provide data quality insights.

- The **Data Analyser** component is implemented using **QMiner** data analytics platform and is opened to other solutions. Within the platform, **QMiner** provides functionality of learning models from historic datasets and use them for prediction on new data points. It implements a comprehensive set of techniques for supervised, unsupervised and active learning which support bot structured and unstructured data.
Finally, the Knowage suite implements the Data Reporter component of the architecture by providing tools for producing high-quality reports of the transformed, enriched and analyzed information obtained from the platform.

In terms of the improvements and additional tools and services that were developed for the platform, the document describes the common data warehouse (Enrichment Database) cluster, the newly developed architecture and partial implementation of the Processing component, as well as a set of improvements aimed to support the publication of data in the data warehouse. The Enrichment Database has been set up using ArangoDB – a multi-model database engine. It serves as a point of integration between the tools in the project. Furthermore, the ArangoDB data model (i.e., the additional support for graph and document stores) is used to enable data integration and enrichment, as well as highly optimised querying. Our first implementation of the Processing component provides clustered infrastructure for deployment, management and scaling of data workflows. Finally, the support for ArangoDB in DataGraft, Knowage, and Grafterizer for management, reporting and generation of ArangoDB data has been added.

The deliverable additionally outlines the APIs of the platform and the addressed aspects of platform security, with respect to each of the components and the overall architecture.
# Table of contents

History of Changes.................................................................................................................................................. 2  
Executive Summary .................................................................................................................................................. 3  
List of figures .......................................................................................................................................................... 7  
Chapter 1  Introduction......................................................................................................................................... 8  
  1.1 Objectives and Scope ...................................................................................................................................... 8  
  1.2 Relationship to Other Deliverables ............................................................................................................. 8  
  1.3 Document Structure ...................................................................................................................................... 8  
Chapter 2  Platform Architecture ......................................................................................................................... 9  
  2.1 Overall architecture ....................................................................................................................................... 9  
  2.2 EW-Shopp Platform – v1 – Implementation ................................................................................................. 10  
    2.2.1 Data wrangler component .................................................................................................................... 11  
    2.2.2 Data analyser component .................................................................................................................... 22  
    2.2.3 Data reporting component .................................................................................................................... 24  
    2.2.4 Enrichment database ........................................................................................................................... 24  
    2.2.5 Processing component and data workflows ....................................................................................... 26  
  2.3 EW-Shopp Platform deployment .................................................................................................................. 28  
    2.3.1 Data wrangler component .................................................................................................................... 28  
    2.3.2 Data analyser component .................................................................................................................... 28  
    2.3.3 Data reporting component .................................................................................................................... 29  
    2.3.4 Enrichment database ........................................................................................................................... 29  
    2.3.5 Processing component .......................................................................................................................... 29  
Chapter 3  Platform API ..................................................................................................................................... 30  
  3.1 Data hosting platform services ................................................................................................................... 30  
    3.1.1 DataGraft catalogue ............................................................................................................................ 30  
    3.1.2 ArangoDB API .................................................................................................................................... 30  
  3.2 Data Analysis APIs (QMiner) ..................................................................................................................... 31  
  3.3 Data Reporting APIs (Knowage) ................................................................................................................ 31  
    3.3.1 CRUD operations for datasets .......................................................................................................... 32  
    3.3.2 CRUD operations for a specific dataset ............................................................................................. 32  
    3.3.3 Operations on the dataset content .................................................................................................... 32
3.3.4 Operations on document resources ................................................................. 32
3.4 Security and access control .................................................................................. 33
List of figures

Figure 1 EW-Shopp platform architecture .................................................................9
Figure 2. EW-Shopp platform v1 current implementation ...........................................10
Figure 3. Grafterizer data preparation process ...........................................................12
Figure 4. Data cleaning pipeline in Grafterizer ...........................................................13
Figure 5. RDF mapping for the data in Table 1 ...........................................................14
Figure 6. RDF graph corresponding to the data in Table 1 ...........................................14
Figure 7 DataGraft platform overview ......................................................................19
Figure 8 DataGraft DBMS administration ..................................................................20
Figure 9 DataGraft ArangoDB information - edit .......................................................21
Figure 10 DataGraft AQL query of collection data .....................................................22
Figure 11. Big Data Runtime .......................................................................................26
Chapter 1 Introduction

This deliverable describes the initial release of the EW-Shopp Platform. It describes the initial (partial) implementation of the platform architecture as described in Deliverable D3.1 and the way the components of the architecture interact and share information.

1.1 Objectives and Scope

The objective of the first version of the EW-Shopp platform is to provide a prototype of integration of the tools and services that will eventually implement the full platform architecture and demonstrate how the tools work together. It is based on current implementations (with extensions) of the DataGraft platform\(^1\), QMiner\(^2\) and Knowage\(^3\). Apart from the existing tools, the platform provides proof-of-concept implementations for other components of the architecture. This initial version of the platform has been designed to serve small-scale pilots based on chosen sample data, which is part of the initial pilots of the business cases described in Deliverable D4.1.

1.2 Relationship to Other Deliverables

This deliverable is tightly associated to a number of other deliverables, as it provides the main technical contribution of the EW-Shopp project – the EW-Shopp Platform. Firstly, this first version of the platform implements the main data integration and analytics processes to be supported in EW-Shopp Platform and provides the set of tools needed to enable the aspects of data interoperability/data modelling that were described in Deliverable D1.1. Furthermore, the EW-Shopp Platform aims to support data workflows that have been described in D2.1. As mentioned, the deliverable implements part of the platform architecture that has been described in D3.1. Finally, the first version of the EW-Shopp platform has been designed to support the pilots that were described in deliverable D4.1.

1.3 Document Structure

The rest of the document has been divided as follows:

- Chapter 2 briefly discusses the platform architecture (from D3.1) and describes the relation of the implementation and its individual components to the main parts of the architecture.

---

\(^1\) [https://datagraft.io/](https://datagraft.io/)
\(^2\) [http://qminer.ijs.si/](http://qminer.ijs.si/)
\(^3\) [https://www.knowage-suite.com/site/home/](https://www.knowage-suite.com/site/home/)
Chapter 2  Platform Architecture

This section contains a brief description of the overall architecture of the EW-Shopp platform. Furthermore, it provides details on the implementation of the architecture that is provided in the first version of the EW-Shopp platform.

2.1 Overall architecture

The overall architecture of the EW-Shopp platform is shown on Figure 1 and has been described in detail in deliverable D3.1.

![Figure 1 EW-Shopp platform architecture](image)

The initial implementation of the EW-Shopp platform focuses on the key components of the platform – the Enrichment Database, Processing component, Data Wrangler, Data Analyser, Data Reporter. The goal of the initial implementation is to demonstrate the capabilities of the components and services as well as their integration with respect to sample data and support the pilots of the business cases described in deliverable D4.1. Section 2.2 provides details on the roles of the individual components and their implementation in this version of the EW-Shopp platform.
2.2 EW-Shopp Platform – v1 – Implementation

Figure 2 shows an overview of the current implementation of the EW-Shopp platform.

In the first implementation of the EW-Shopp platform, the *Enrichment Database* is implemented using the *ArangoDB*\(^4\) multi-model store. It serves as the staging area of the integrated and enriched data and is the central data space, from where the data is consumed for analysis and reporting. The data in the *Enrichment Database* is imported from the *Processing* component, which is responsible for executing data workflows to clean up, model, transform, integrate and enrich raw input data. The *Processing* component is the infrastructural environment where these data workflows are executed and is implemented using Docker, GlusterFS and Rancher (details in the following chapters).

The definition and management of the data workflows in this architecture is done by the *System Orchestration* component (not yet implemented; preliminary experiments with Apache NiFi described in section 2.2.5.3). It will define high-level data workflows based on transformation models provided by the *Data Wrangler* component, as well as other proprietary dataset-specific steps.

The *Data Wrangler* component is implemented by making use of and extending the DataGraft platform, along with the ASIA and ABSTAT tools. DataGraft provides secure access for administration and access of data. It provides upload and query of data stored in the *Enrichment Database*. *Grafterizer* provides cleaning and transformation of tabular data into graph format. It runs as a cloud service using test data for interactive development and test, and the result can be downloaded for

\(^4\) [https://www.arangodb.com/](https://www.arangodb.com/)
execution on Big Data Engine/Processing component. ASIA is a tool for the semantic enrichment of data available in tabular formats, thus helping users of the EW-Shopp platform in integrating business data with events and weather data. Semantic reconciliation algorithms are integrated into a user interface to help users map the data schema to shared vocabularies and ontologies and link data values to shared systems of identifiers. Data enrichment widgets exploit these links to shared systems of identifiers to ease the extraction of additional data from third-party sources and their fusion into the original tabular data. Additional support for semantic data enrichment is provided by ABSTAT. ABSTAT is a tool to profile knowledge graphs represented in RDF based on linked data summarization mechanisms. The profiles extracted by ABSTAT describe the content of knowledge graphs, using abstraction (schema-level patterns) and statistics. The profiles help users understand the content of the knowledge graphs used in the platform (e.g., the product knowledge graph), support ASIA’s semantic reconciliation algorithms, and provide data quality insights.

The Data Analyser component is implemented using QMiner data analytics platform. QMiner combines a robust C++ core with a JavaScript wrapper and is released as part of the Node.js runtime environment package system. Within the platform it provides functionality of learning models from historic datasets and use them for prediction on new data points. It implements a comprehensive set of techniques for supervised, unsupervised and active learning which support both structured and unstructured data. As it is primarily used as a library, it does not have a native front-end graphical user interface. For interactive work, it can be used through Jupyter, an open-source interactive computing platform, using the Node.js kernel.

Finally, the Data Reporter component of the architecture is implemented using the Knowage tool suite. Knowage is a professional open source business analytics suite that combines traditional data and big data sources into valuable and meaningful information. It merges the innovation coming from the community with the experience and practices of enterprise-level solutions. In the EW-Shopp project Knowage is responsible for producing high-quality reports of the transformed, enriched and analyzed information obtained from the platform. This reporting will be made possible both in a web graphical interface in standard reporting formats.

The following sub-sections provide more details about the implementation, deployment and role of the individual components.

### 2.2.1 Data wrangler component

The data wrangler component is a composite component that consists of three main sub-components, which are discussed in this section. The Data Manipulation UI is implemented by the Grafterizer data transformation tool. Grafterizer is responsible for producing data cleaning pipelines and graph generation mappings that together make up the Transformation Model used by the Big Data Runtime component. The Transformation and Enrichment component is implemented by the ASIA tool, which has been integrated with Grafterizer and is responsible for modelling the data and

---

5 https://nodejs
6 https://jupyter.org/
7 https://github.com/notablemind/jupyter-nodejs
aiding the process of integrating the Core Data from Event Registry, ECMWF and ABSTAT. Finally, the DataGraft portal is used as a sub-component responsible for user-facing miscellaneous tasks such as user management, managing user assets on the EW-Shopp platform (i.e., Enrichment Database endpoints, queries, transformation models) and enabling easy access to data.

2.2.1.1 Data manipulation UI

Grafterizer is a data transformation tool which is part of the DataGraft platform. It provides an interactive user interface for cleaning up tabular data and transforming it into a graph format. It supports data transformations from raw tabular data to knowledge graphs in RDF\(^8\) using a schema mapping (graph template), which in EW-Shopp has also been adapted to generate JSON document collections for the Enrichment Database (see Figure 3).

![Grafterizer data preparation process](image)

In the Grafterizer tool, transformation models consist of a pipeline of steps for data cleaning that can be found in the "Pipeline" tab (see Figure 4a) and a mapping template, defined in the "RDF mapping" tab of the transformation UI (see Figure 4b).

\(^8\) [https://www.w3.org/RDF/](https://www.w3.org/RDF/)
A transformation model is applied by taking each row in a tabular dataset, running it through each of the pipeline steps and then creating a corresponding graph through the graph template. As mentioned, Grafterizer currently supports two methods for graph generation. The default approach generates RDF triples, which can be stored in a semantic graph database. For example, suppose the following row of input data in Table 1 shall be mapped to RDF.

<table>
<thead>
<tr>
<th>match-id</th>
<th>match-label</th>
<th>Match Type</th>
<th>Clicks</th>
<th>Impressions</th>
<th>Google Ad Position</th>
<th>xsd-datetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>14071400102-...</td>
<td>Match of keyword [...]</td>
<td>F</td>
<td>0</td>
<td>1</td>
<td>3.7</td>
<td>2016-08-08 T00:00:00</td>
</tr>
</tbody>
</table>

In order to create an RDF graph out of the tabular data, a user needs to use an RDF mapping. Figure 5 shows an example mapping of that data.
Figure 5. RDF mapping for the data in Table 1

Executing the mapping in Figure 5 will result in a simple RDF graph, which is shown on Figure 6. Note that in the resulting RDF graph all of the prefixes have been expanded to the URIs that they point to.

Figure 6. RDF graph corresponding to the data in Table 1

In EW-Shopp, to support our architectural requirements, the RDF mapping approach has been extended with a tool for generating ArangoDB collections\(^9\) that takes advantage of the RDF mapping

\(^9\) [https://github.com/datagraft/Datagraft-RDF-to-Arango-DB](https://github.com/datagraft/Datagraft-RDF-to-Arango-DB)
templates. The tool is open-source and is currently available both as a script (used by our current implementation of the Processing component) and as a RESTful web service (will be used in the next version of the EW-Shopp platform to provide a wizard interface for publishing data in ArangoDB). It consumes input tabular data and a transformation model and generates document collection and, if graph connections are defined, edge collections that can then be imported in ArangoDB.

Mapping to ArangoDB collections can be done in various ways because of the flexibility of ArangoDB’s data model. Whereas in RDF each entity and property is by design in itself a node, in the data model adopted by ArangoDB, this is not necessarily true. Entities are represented by JSON objects with a (numeric) key attribute, and optionally other attribute-value pairs. This allows for the properties of an entity to be stored as attributes of the JSON object that is representing it. A graph is formed by simply connecting the individual JSON objects through their numeric keys. Each of these connections are declared in separate JSON objects in a special collection called an edge collection. Given that flexibility, two mapping strategies have been developed: 1) 1-to-1 with respect to the RDF data model – each node in the RDF mapping corresponds to a node in the node collection – this strategy is the most expressive, but performs badly when the number of stored entities is large and/or when there are many connections between entities; and 2) flattened RDF mapping – this is the chosen mapping strategy for implementing the business cases of EW-Shopp. It takes advantage of the ArangoDB data model to store properties as object attributes following a set of rules:

- **URI nodes are mapped JSON objects, which serve as nodes in ArangoDB**
  - URIs, which in RDF uniquely identify nodes, are used to generate unique numeric keys for the JSON object (numeric keys enable more efficient storage and lookup). This is done using a standard hash function and the keys themselves are stored in a special attribute called _key.
  - Edges between nodes are generated based on the links between URI nodes in the RDF mapping template.
  - Exception: rdf:type mappings – in RDF, these are used to specify type mappings for RDF entities. Types in RDF are URI nodes, which point to the semantic classes in an ontology or vocabulary, similarly to classes in object-oriented programming special case. The classes specified are instead stored in a 'type' attribute, which is an array of types.

- **RDF literals are mapped to JSON attributes for the URI node objects**
  - Exception: rdfs:label – in RDF, these mappings are used to denote textual labels to denote entities. In the ArangoDB mapping, the values of these mappings are stored in a 'label' attribute and used to display labels in the Graph interface of ArangoDB.

- **Prefixes and fully qualified RDF URIs are also stored in the resulting JSON object in ArangoDB.**
  - The specified prefixes in the mapping are additionally kept in separate JSON objects in the node collection to avoid overlaps with other prefixes and for enabling namespace-based lookups (based on the RDF namespaces defined in the mapping).

Given the rules above, the resulting ArangoDB object from the data in Table 1, according to the RDF mapping in Figure 5 will be as shown in Listing 1.
Note that in this case there are no edge collection entries, as there were no specified connections between URI nodes in our example mapping. See Section 2.2.4 for an example of an edge collection entry.

Grafterizer allows for the transformation models to be exported as self-contained Java executables (JAR files), which can be applied independently of the Grafterizer tool. Furthermore, the transformation models can be exported in a JSON format, which is consumed by the aforementioned script/web service for generating ArangoDB collections. In the EW-Shopp platform, these serializations of the Grafterizer transformation models are used to achieve horizontal scaling. This is necessary when integrating the data cleaning as a step of the data workflow, which is executed in the Processing component of the Big Data Runtime.

Thereby, the JAR files of transformation models that clean up tabular data are encapsulated in Docker\(^\text{10}\) images, which are used to create containers that are then deployed on the Processing cluster (see section 2.2.5). They consume input tabular data from the shared file system and store the outputs (also tabular data in this case) where they can be in turn consumed by later steps in the data workflow.

The JSON export of the transformation model, on the other hand, is one of the input parameters to the script that generates the ArangoDB collections to be stored in the Enrichment Database. The other input parameter is the path to the file that was cleaned up with the JAR transformation. Thus, the ArangoDB collection generation tool can also be encapsulated using Docker and directly fed data through a shared file system.

2.2.1.2 Transformation and enrichment

This chapter describes the role of ASIA and ABSTAT in the EW-Shopp platform, then it specifies what is integrated in EW-Shopp platform v.1.

While Grafterizer already supports transformations of tabular data into graph format, the user needs to manually define these transformations. ASIA and ABSTAT are aimed at supporting the creation of these transformations using dedicated algorithms and eases the process of fetching data from third-

---

\(^{10}\) [https://www.docker.com/](https://www.docker.com/)
party sources to enrich business data, in particular with events and weather data, but also with product and geospatial data.

Semantic reconciliation algorithms are integrated into a user interface based on Grafterizer’s front-end framework. Semantic reconciliation is supported by two main functionalities:

- **Schema-level linking**: mapping the data schema, which in tabular data consists of a list of columns possibly associated with a header, to shared vocabularies and ontologies (schema-level links, also referred to as mappings).
- **Instance-level linking**: linking data values that occur in the table to shared systems of identifiers (instance-level links).

These operations, embeddable to some extent in Grafterizer’s data transformations, will be provided by ASIA as semantic reconciliation services (also referred to as schema-level and instance-level linking services). While these linking services are helpful for establishing schema-level links, they are necessary for establishing instance-level links, which require disambiguation of a large number of textual values in the table (e.g., location names in AdWords data as used in BC4). In particular, instance-level data linking services are needed in EW-Shopp to support reconciliation of product-related data and location-related data against reference product knowledge graphs (the GfK catalogue) and a geospatial dataset (GeoNames) as further discussed in D1.1, D1.2 and D1.3.

Schema-level and instance level links are created as **annotations** for the table, also referred to as **schema-level and instance-level annotations**, respectively. For this reason, ASIA can be viewed as a **semantic table annotation tool**, powered by **semantic table interpretation** functionalities that are modularly implemented by the linking services developed in the ASIA back-end. The linking services suggest annotations that can be validated by the users through ASIA’s user interface. The interface lets also users configure the linking services. Annotations generated with ASIA support two main functionalities: generation of knowledge graphs from tabular data and enrichment of tabular data with third-party data, with the latter being more prominent in the EW-Shopp project.

**Generation of knowledge graphs from a tabular data set.** Schema-level and instance-level annotations are translated into executable data transformations to generate a knowledge graph from a tabular data set. Executable data transformations are implemented using the same language used in DataGraft to map tabular data to RDF (which encodes schema-level links already), and extending it to handle instance-level links.

**Enrichment of tabular data with third-party data.** Schema-level and instance-level annotations with reference knowledge graphs are used to facilitate enrichment of business data with data from these reference knowledge graphs (e.g., brands of a product from product data, or coordinates of locations from GeoNames) or third-party data. This is achieved by using the links encoded by annotations as bridges to third-party data sources (e.g., by using GeoNames identifiers to invoke event data services, or by using coordinates extracted from GeoNames from location identifiers to invoke weather data services). To support this key enrichment process in EW-Shopp, ASIA will provide **data enrichment widgets**, which exploit annotations to ease the
extraction of additional data from third-party sources and their fusion into the original tabular data.

**ABSTAT for data transformation and enrichment.** *ABSTAT* profiles have three main roles in data enrichment:

- Support data understanding: profiles help users understand the structure, content and vocabulary/ontology of a reference knowledge graph, to which a user wants to map his/her data. Only by understanding these properties of a reference knowledge graph, a user can take informed decisions about schema-level annotations.
- Provide schema-level linking services to suggest annotation in *ASIA* interface. *ABSTAT* offers two main APIs that support this functionality: i) *ABSTAT* Autocomplete, which provides autocomplete functionalities using vocabulary/ontology terms used knowledge graph profiles indexed in *ABSTAT*; ii) *ABSTAT* Search, which provides full-text search over profiles.
- Provide information useful for linking algorithms.

*ASIA* and *ABSTAT* are developed using an agile methodology: new functionalities are frequently added based on development activities driven by needs emerging in different application scenarios. Functionalities that will be added as part of EW-Shopp activities are:

- **ASIA:**
  - Development of instance-level linking services.
  - Development of data enrichment widgets for *ASIA*.
- **ABSTAT:**
  - Profiling for graph data stored in ArangoDB.
  - Development of a scalable distributed infrastructure.

Development of instance-level linking services is an on-going activity that will contribute to D3.2, according to the project work plan. Development of data enrichment widgets for *ASIA* will follow up as optimization of the platform after linking services are created. Profiling for graph data stored in *ArangoDB* and development of a scalable distributed infrastructure for *ABSTAT* are also on-going activities.

The EW-Shopp platform - v1 includes the following versions of *ASIA* and *ABSTAT*:

- **ASIA v0.1**\(^{11}\), licensed under the GNU Lesser General Public License v3.0\(^{12}\) licence
  - The tool provides schema-level linking functionalities and a fully operational user interface integrated with Grafterizer front-end. Integration with *ABSTAT* APIs to support schema-level annotations is completed.
- **ABSTAT v0.1**\(^{13}\), licensed under the GNU Affero General Public License v3.0 licence

\(^{11}\) [https://bitbucket.org/disco_unimib/asia](https://bitbucket.org/disco_unimib/asia)  
\(^{12}\) [https://www.gnu.org/licenses/](https://www.gnu.org/licenses/)  
\(^{13}\) [https://bitbucket.org/disco_unimib/abstat-core](https://bitbucket.org/disco_unimib/abstat-core)
The tool supports profiling of Knowledge graphs represented in RDF and formatted according to N3 format. It consists of five components:

- **Builder**, which executes the summarization algorithm that generates the profiles.
- **Storer**, which reads/writes raw data generated by the Builder and store the result in a data lake used as temporary storage to manipulate summaries.
- **Data Loader**, which reads data from the data lake, stores data in a MongoDB database, and indexes data in an ElasticSearch engine. MongoDB and ElasticSearch serve different data access operations.
- **Explorer**, which implements operations required to support browsing, search, and autocomplete over summaries.
- **Viewer**, which provides a user interface to run summarization, and let users browse and search profiles.

### 2.2.1.3 Asset management

![DataGraft platform overview](image)

*DataGraft*[^1] is a collection of tools for integrated management of transformations, hosting and access of graph data as shown in Figure 7. It is organized as a set of cloud services presented to the user through a Web portal. Its original target has been towards RDF data stored in triple stores. For the EW-Shopp project the tools are extended to provide transformation and hosting of graph data in *ArangoDB*[^2]. In this concept node data, in tabular form, and the edge data, graph relationships, can

---

[^1]: https://datagrafit.io/
[^2]: https://www.arangodb.com/
be stored and queried in the same database. This chapter describes the extensions done to the DataGraft tools to support ArangoDB.

The transformation of data into ArangoDB graph format is different from the standardised triple store. The v1 implementation has been focused on testing out the transformation concepts to be used for the Big Data Runtime in order to scale up for big datasets.

DataGraft handles development of cleaning steps and transformation mapping of sample data sets. The transformations are stored as assets in the portal. Upload-ready transformed collection data in JSON format to ArangoDB can be done using local scripts or by using the portal.

Access towards ArangoDB databases is implemented using the REST API interface for accessing the ArangoDB database and storing the login credentials. The login credentials and the databases are registered by the user using the ArangoDB Web interface and copied into the DBMS admin page shown in Figure 8. Two sets of user credentials are handled full access and read access. Full access is only available for the asset owner, while the read-only access can enable access to other users if the asset is registered as public. Multiple systems can be instantiated for full flexibility of user access and database location. This makes the DataGraft portal able to connect to databases installed at different locations in the cloud, or at customer premises.

The DataGraft portal will use the credentials when creating, querying and deleting collections inside the database. Each database can be connected to DataGraft as an ArangoDB asset. The asset is the main access point for displaying metadata, uploading new data and querying. The metadata managed by DataGraft consists of a textual description, keywords and data access license. In addition, the ArangoDB asset page displays synchronized information about available collections as shown in Figure 9.
Querying of collections can be done using the AQL\textsuperscript{16} query language. In DataGraft, AQL queries can be issued from the query panel as shown in Figure 10. AQL is a dedicated query language specific for ArangoDB. AWL queries can also be stored as assets on DataGraft and reused. This makes it easy to access transformed data to document the result.

\textsuperscript{16}https://docs.arangodb.com/3.2/AQL/index.html
2.2.2 Data analyser component

The analyser component implementation is strongly linked to the functionality it provides. It has two main modes of operation, which are named learning and prediction. The detailed description of these are presented in deliverable D3.1. Only their short outlines are provided hereafter.

The learning mode is used when a model is built from historical data. Typically, this is a batch job where a large dataset is provided to the analyser which processes it and outputs a model performing the analytic function. The latter can be a prediction of some value such as the number of page views within a certain category of products on a web store or a most likely category for a given new product.

On the other hand, prediction is the production mode of the analyser component, used for operational deployment of the learning mode output. The model built in the learning stage is given
new examples to process. For the two examples from the previous paragraph these would be data about the weather and event context for an upcoming date with the target web store category and descriptions of new products submitted to a web store. The model processes the new examples and returns the predicted value.

2.2.2.1 Learning
Learning is a batch operation performed by processing a dataset of historical data. The dataset is processed by QMiner, which builds a model of the relation between some observations and a target value. Since QMiner parses the data into its own internal representation due to optimization for the learning algorithm, the source of the data (e.g. a table in a database or a file on disk) is not important. The only thing needed is the specification of the learning problem, where the target variable and the input features are specified. QMiner uses JSON as the format for specifying the input data schema.

Data is extracted from the ArangoDB database via the official JavaScript driver library and loaded into QMiner. A data management API linking ArangoDB to QMiner supports this transfer following a specification (in JSON) which provides both database information, such as hostname, port and database name, as well as the data fields to obtain.

It is worthy to note that data could be just as easily consumed by QMiner in some other format. The most relevant example of this is that data is provided “raw” as a CSV file on disk, possibly even unenriched. This option can be used during pilot deployment when the platform may not yet be fully developed and allows for parallel development of the analytics and integration services or in cases were businesses may forgo the enrichment stage of the pipeline for any reason (e.g. data restrictions, software/hardware limitations, pre-enriched data etc.). In this case SFTP can be used to transfer into a designated folder any relevant data files in CSV format as well as a JSON configuration file. The latter must now also specify the enrichment settings so that the data loading scripts can use the enrichment API specified in deliverable D1.3 to enrich the data on-the-fly as it is imported into QMiner.

QMiner builds a model using the provided integrated and enriched dataset. The analytics method used and its parameters are specified in the configuration file along with the feature space described following the standard QMiner JSON schema description format. The model is dumped into a binary file in the same folder as the input data. The model file can be loaded into new QMiner instances to perform prediction in the production environment.

2.2.2.2 Prediction
The model is deployed using a simple Node.js HTTPS server architecture and is made available through a RESTful API. Query examples are received via POST requests encoded into JSON format following the same schema definition as the one provided a learning time. Predictions are returned by extending the received JSON structure with an appropriate output field. It is up to the querying services to supply input data in the expected format. If the data is supplied via ArangoDB this only means transformation of the database records into query format. In the case of direct use, the

18 https://github.com/arangodb/arangojs
examples may need to be integrated and enriched on the fly using the same procedure as described for the learning stage.

For setting up such a prediction instance of QMiner the only things needed are a model binary file built by the learning stage and a JSON configuration file specifying the model filename and server parameters (hostname, port). A setup script initializes QMiner, loads the model and starts the prediction server.

2.2.3 Data reporting component

Knowage is an open source suite developed by ENGINEERING using a continuous process of evolution and maintenance, thanks to efforts coming from internal resources and from the community of developers who joined the project. Knowage is released in two versions: Community and Enterprise Edition. A brief description of this suite and its functionalities has been provided in deliverable D3.1 (sub-section 5.4.3).

In the first release of the EW-Shopp platform, the main improvements to Knowage have been implemented to enable integration with the Enrichment Database – the storage component containing the data to be visualized in Knowage through dashboards or reports. This integration will take advantage of the features provided by Knowage to define a dataset from a REST service that provides the data to be visualized. Thereby, the API provided by ArangoDB will be used to get the data to be visualized in Knowage in order to enable a more immediate and intuitive understanding. To achieve this goal, after the connection to ArangoDB and the retrieval process the needed analytical documents are defined, and use the Knowage suite’s module library to define reports.

2.2.4 Enrichment database

The ArangoDB database will serve as the Enrichment Database in the overall architecture of the EW-Shopp platform. ArangoDB is a multi-model NoSQL database providing support for document, key-value and graph data models in the one package. One of the main advantages of ArangoDB over other NoSQL solutions is the support for a graph model, which allows for easier integration of multiple datasets through connections between documents. Furthermore, ArangoDB is easy to deploy on ad-hoc infrastructure, since it is a fully certified package for the DC/OS\(^{19}\) cluster management system. Easy deployment and management is one of the main requirements for the EW-Shopp Corporate environment, which will be deployed on the premise of users of the platform. In terms of support for other infrastructure configurations, ArangoDB also provides deployable recipes for public cloud-based infrastructure-as-a-service platforms like Amazon AWS\(^{20}\) and Microsoft Azure\(^{21}\), as well as Docker images for all of the necessary database components for Docker-based deployment.

---

\(^{19}\) [https://dcos.io/](https://dcos.io/)
\(^{20}\) [https://aws.amazon.com/](https://aws.amazon.com/)
\(^{21}\) [https://azure.microsoft.com/en-us/](https://azure.microsoft.com/en-us/)
In terms of the storage engine, ArangoDB provides two options – in-memory (memory-mapped files, a.k.a. mmfiles), or disk-based storage – using the RocksDB storage engine. In-memory storage is appropriate in cases where the datasets do not exceed the amount of memory available on the host. The RocksDB engine keeps an in-memory cache of a subset of the data and loads additional data from disk on-demand. For the purposes of the EW-Shopp platform, the storage requirements (as described in D2.1) prohibit the usage of in-memory storage. Thus, the RocksDB storage engine is the more appropriate choice.

In ArangoDB data are structured in collections with keys, attributes and values. Every entity in this data model is stored in a document collection and represented as a JSON object (see Listing 2), thus implementing both the key-value and document store models (key-value collections are implemented through flat key-value JSON objects).

```
{
  _key: "5",
  numberOfClicks: "2",
  numberOfImpressions: "1",
  adPosition: "2.0",
  date: "2016-07-08T00:00:00"
}
```

Listing 2. Example document collection entry

The graph data model is implemented through expressing connections between documents (entities). These connections are realised through a special type of collection called an edge collection. Edge collections have two extra parameters in addition to a normal document collection – from and to, which point to the keys of the entities in one or more document collections. For example, the edge collection entry in Listing 3, connects two nodes with key attribute values of '5' and '6'.

```
{
  _from: "5",
  _to: "6"
}
```

Listing 3. Example edge collection entry

For the purpose of the first prototype of the EW-Shopp platform, two experimental instances of the ArangoDB database have been deployed. These instances are available internally to the stakeholders in the EW-Shopp project. The database instances are deployed on the premise of SINTEF. They have two different configurations. The first configuration uses a three-node mmfiles cluster. It has been used for initial experimentation for lower-volume data, which has been sharded over the three different instances. The second deployment uses a single-node deployment with the RocksDB engine. It has lower memory requirements, which is important for the volumes of data that the EW-Shopp platform intends to support and has been therefore determined more appropriate for the first prototype. The instances have been deployed in an isolated network, accessible only through a VPN so that the ArangoDB server is not accessible directly over the Internet (for security purposes). This simulates the Corporate environment of the EW-Shopp platform.
The single-node configuration, which will be used as the main implementation of the *Enrichment Database* has been deployed using the Docker-based deployment option of *ArangoDB*\(^2\).

### 2.2.5 Processing component and data workflows

This section describes the hardware and software implementation of the data ingestion pipeline. It implements the *Processing* component of the Corporate environment in the overall architecture by providing data processing pipelines for ingesting raw data and importing data to the *ArangoDB Enrichment Database*. Figure 11 shows the big data runtime environment, highlighting the currently implemented set-up, which has been deployed on the premise of SINTEF.

![Figure 11. Big Data Runtime](image)

The following subsections describe the different component of the big data runtime.

#### 2.2.5.1 Processing Components

Upstream data delivered to the EW-Shopp platform runs through an ingestion pipeline before arriving in the *ArangoDB* multimodal storage. Depending on the type of upstream data, processing steps may differ, but generally follow the same four step pattern.

1. Decompress the data.

\(^2\)[https://docs.arangodb.com/3.2/Manual/Deployment/ArangoDBStarter.html](https://docs.arangodb.com/3.2/Manual/Deployment/ArangoDBStarter.html)
2. Conversion to comma separated value (CSV) format.
3. Cleaning of data.
4. Conversion into a graph format.

Each processing step is wrapped in a Docker image from which containers can be instantiated. The pipeline of steps (or images) is arranged into a stack (in Rancher/Cattle terminology) where each step can scale out to launch in as many parallel instances as required by the data volume (within the constraints of the hardware platform).

Within each processing step, a shell script watches for new files an input directory. If a new file arrives in the input directory, the file is moved to a working directory, operated upon (i.e., decompressed, converted to CSV, cleaned, converted into a graph representation), and then moved to an output directory. This move from input to working to output directory effectively hands off the different processing steps to each other and prevents race conditions between the individual services. Since all nodes mount a shared filesystem, later processing steps are not constrained by which node previous steps have run on.

2.2.5.2 Initial Experiments
Initial experiments were performed using the JOT dataset from BC3. This set is delivered as approximately 100GB of compressed CSV files. Here, the processing chain above instantiates to the following concrete steps (with the core tool in parenthesis).

1. Unzip delivered data (`unzip` command).
2. Convert files from tab separated to comma separated (`tr` command).
4. Convert to ArangoDB graph (using the ArangoDB collection generation tool described in section 2.2.1.1).

Wrapping simple commands such `unzip` and `tr` in a Docker image is done in order to enable the scalability of the pipeline.

2.2.5.3 Dataflow Management using Apache NiFi
As outlined, processing steps are currently wrapped in Docker images and are composed together by defining suitable combinations of directories. These are defined in stacks within Rancher/Cattle. Stacks can be managed through the web interface as well as an API, which means that processing flows could be put under version control. This, however, requires significant middleware development. Additionally, there is currently no overview of the current utilization of the pipeline. Again, such functionality requires engineering suitable middleware.

Apache NiFi is an open source data flow management system based on processors and their connections. In addition to supporting vertical and horizontal auto-scaling based on utilization, amount of queued up data, and a host of other parameters, NiFi also provides high-level overview of pipeline utilization as well as a data provenance scheme where the flow of a parcel of information through the system is retained in a history file.
Given these features, effort is underway to embed the above pipeline within Apache NiFi. Due to the complexity and integration requirements of NiFi, the effort to arrive at a working prototype is larger than for the simple pipeline. However, the functionality of even a minimal prototype implemented in NiFi exceeds that of the current Docker/Rancher/Cattle based solution.

2.3 EW-Shopp Platform deployment

This subsection summarises the infrastructure for the deployment of the individual EW-Shopp platform components.

2.3.1 Data wrangler component

The Data Wrangler component is implemented by the DataGraft platform along with the set of services and extensions described in sub-section 2.2.1. The component's architecture allows for flexible deployment options, which are used for the purpose of the EW-Shopp platform.

The main (production) environment of DataGraft is deployed on public cloud infrastructure provided by Amazon’s AWS platform and publicly accessible at https://datagraft.io/. DataGraft uses a microservice architecture, whereby the individual sub-components are encapsulated in Docker containers. The main environment is managed by the Elastic Container Service (ECS), which sets up the networking and server hosting for the components. The catalogue of DataGraft assets is maintained in a relational database (PostgreSQL), which is provided by the Amazon RDS service (RDS). DataGraft supports uploading and making available individual files, which in the production environment are hosted using the Amazon S3 block storage.

As of the writing of this deliverable, the new features and additions to the DataGraft platform and respective services have not yet been deployed on the production environment. Instead, a separate secured environment hosted on SINTEF’s premise is used and made available to the EW-Shopp partners through a VPN connection. The same Amazon services are used (S3 and RDS), with the exception of ECS, which is replaced by Docker Compose.

2.3.2 Data analyser component

The two primary challenges are as follows. Firstly, to make custom processors available in NiFi, the processing module needs to either be converted into a generic script which reads from ‘stdin’ and writes to ‘stdout’ or be compiled to link to a NiFi native interface. Secondly, to obtain optimal throughput, a variety of parameters governing size of data chunks, processing time limitations, as well as vertical and horizontal scale-out activations need to be tweaked. Optimal performance thus requires experimentation and benchmarking.

---

23 The two primary challenges are as follows. Firstly, to make custom processors available in NiFi, the processing module needs to either be converted into a generic script which reads from ‘stdin’ and writes to ‘stdout’ or be compiled to link to a NiFi native interface. Secondly, to obtain optimal throughput, a variety of parameters governing size of data chunks, processing time limitations, as well as vertical and horizontal scale-out activations need to be tweaked. Optimal performance thus requires experimentation and benchmarking.

24 https://aws.amazon.com/ecs/
25 https://aws.amazon.com/rds/
26 https://aws.amazon.com/s3/
27 https://docs.docker.com/compose/
A machine has been provided at JSI to host an instance of QMiner for the platform needs. The machine has the following specification:

- Processor: 2x Intel Xeon E5-2630 (2.30GHz) 6-core/12-thread
- Memory: 64 GB DDR3 RAM (1333MHz)
- Disk: 3 TB (SATA)

In our experience from previous research and commercial projects this hardware is sufficient for efficient development of industry-level analytics models. Should the needs of some business partner exceed its capabilities, a new instance can be provided on hardware they supply or new computing capabilities can be rented in agreement with the other project partners and the European Commission. However, this option is considered to be highly unlikely.

### 2.3.3 Data reporting component

The Data Reporter component is implemented by Knowage and is deployed in the ENGINEERING server farm delivering a Cloud infrastructure providing both IaaS and PaaS capabilities. This solution combines ENGINEERING’s experience in applications, technology and supply of complex and highly critical services using Cloud and Big Data technologies. The used infrastructure has a large catalogue of technological and application resources, that can be managed using management portal functions accessed via a secure multi-tenant Web portal. The Knowage server instance connects to the common ArangoDB Enrichment Database through a dedicated VPN network (hosted on SINTEF infrastructure). It allows the data to feed the dashboards in the Web application interface of Knowage or to easily export them to third party application. These data will constitute the information used by Knowage for further visualization and reporting tasks.

### 2.3.4 Enrichment database

The Enrichment Database component for the first prototype, which is implemented by the ArangoDB database, is currently running in a single-node configuration. It has been deployed on a hardware node on the premise of SINTEF and runs on the following hardware:

- Processor: Core i7 3,3GHz 6-core/12-thread
- Memory: 64GB DDR4 RAM (2666MHZ)
- Storage: SSD using PCI Express 400GB

In production environments, depending on the requirements of the particular EW-Shopp platform user, our intention is to use a multi-node cluster with the RocksDB engine, which would be deployed either using the Docker option or DC/OS.

### 2.3.5 Processing component
The prototype of the Processing component, which is currently used in EW-Shopp for experimentation, runs on a 10-node cluster of Linux machines located on the premise of SINTEF. The machines are connected via an Ethernet fabric and mount a shared filesystem (GlusterFS). Each machine runs the Docker engine, so that Docker containers can be deployed. The particular virtual hardware allocated to each of the Docker containers that implement the processing pipeline depends on the load and is configurable. Therefore, no concrete hardware configurations of the nodes that host the deployment are discussed. Container orchestration is managed through a Rancher/Cattle installation. The machines are isolated from the public internet and can be only be accessed through a VPN connection.

Chapter 3  Platform API

This chapter provides details on the exposed APIs of each component in the first prototype of the EW-Shopp platform.

3.1 Data hosting platform services

This section provides Rest API overview for accessing the platform components without using the HTML WEB pages. Summary of available services is given together with links to detailed API specifications.

3.1.1 DataGraft catalogue

Setup and usage of DataGraft is typically done using the web interface. In addition, a REST API is provided with the following new entry points: DBMS Arango, ArangoDB asset and Query asset. Detailed API documentation can be found at

- DBMS Arango provides CRUD operations of Arango server host address and access information.
- ArangoDB asset provides access to a specific Arango database and its collections. CRUD operations of ArangoDB assets and its collections are provided. Upload of JSON files is supported for new and existing collections.
- Query asset provides AQL query execution on specific collections. CRUD operations of the Query asset and execution of queries are supported.

3.1.2 ArangoDB API

https://github.com/datagraft/datagraft-API
ArangoDB provides extensive APIs for accessing and manipulating data in ArangoDB databases. The full API of ArangoDB is available at [https://docs.arangodb.com/3.2/HTTP/index.html](https://docs.arangodb.com/3.2/HTTP/index.html).

In EW-Shopp, the ArangoDB API will be used by the different components to access and manipulate ArangoDB databases. Given that ArangoDB has been chosen for the implementation of the Enrichment Database component, the API will be consumed as follows:

- **Data wrangler component**: the data manipulation UI will use the ArangoDB APIs for the purposes of the data manipulation UI and the asset management sub-component. The data manipulation UI will be able to directly store collections that result from data transformations in the ArangoDB instances that will be managed through the asset management sub-component. The asset management sub-component, on the other hand, will use the database APIs for user management when registering databases in DataGraft. Furthermore, that component will make use of the collection APIs for retrieving and managing data in registered databases, as well as for importing new data in collections.

- **Data analyzer component**: the ArangoDB APIs will be used when accessing databases, collections and documents for extracting data from the common data warehouse and using them to produce analytics models.

- **Data reporting component**: the ArangoDB APIs will be used for accessing databases, collections and documents in ArangoDB when producing reports corresponding to the analytics models.

- **Processing component**: ArangoDB imports will be part of the data workflows of the processing component (the last step of the workflows). Thereby, the component will make use of the bulk import interface of ArangoDB, or the importer tool API\(^{29}\) depending on the size of the loads and the network configuration (the bulk import APIs are appropriate when the database is accessible only over the network; the importer tool is accessible when the processing component has local access to the Enrichment Database).

### 3.2 Data Analysis APIs (QMiner)

The data analysis APIs are under active development. QMiner primarily being a library it has no set API, however the request form typically follows the structure of internal data and model schema. All operations described in that section are run using JSON input and can be initiated via REST. A clear example of this is the prediction operation described in section 2.2.2.2, where a data example is input to QMiner via a POST request and a prediction is returned. The request is encoded into a JSON following the QMiner standard for data schema definition. The exact structure of the rest of the API is expected to settle during pilot’s deployment.

### 3.3 Data Reporting APIs (Knowage)

\(^{29}\) [https://docs.arangodb.com/3.2/Manual/Administration/Arangoimp.html](https://docs.arangodb.com/3.2/Manual/Administration/Arangoimp.html)
**Knowage** is based on a service-oriented architecture (SOA) whose functionalities can be invoked as rest services. The front-end allows to invoke APIs in a simple and intuitive way. In particular, APIs concerning datasets and analytical documents are the most significant for the purpose of this deliverable. More details can be found in the official Knowage APIs website at https://knowage.docs.apiary.io/. The sub-sections below provide a brief summary of the APIs.

### 3.3.1 CRUD operations for datasets

The API supports the following operations related to datasets:

- Returning all datasets.
- Adding a new dataset.
- Removing a dataset.

### 3.3.2 CRUD operations for a specific dataset

The API supports the following operations related to specific datasets:

- Return the entire dataset.
- Update the dataset.
- Delete the dataset.

### 3.3.3 Operations on the dataset content

The API supports returning the dataset's content. It accepts a list of multiple query parameters, name1=value1&name2=value2&...&nameN=valueN.

### 3.3.4 Operations on document resources

The documents resource represents the list of documents that are visible to the authenticated user.

- List all documents.
- Adds a new document.

Before adding a new document, you have to pay attention to the fact that the document could depend on other objects, in particular a data source and/or dataset objects. In this case you have to create those objects first, using the standard web GUI. After you created those objects, you have to reference them by their label in 'dataSourceLabel' and ' dataSetLabel' request properties.

The document resource represents a specific document.

- Return document with specified Label.
3.4 Security and access control

For the purpose of the first version of the EW-Shopp platform, a set of security measures have been implemented to protect the confidentiality of the EW-Shopp user data. Hereafter, the set of security measures that have been implemented or are already available from the individual tools and services of the EW-Shopp platform are discussed.

DataGraft and Grafterizer: security in DataGraft and Grafterizer is implemented on several layers as follows:

- User login and SSL (valid for the DataGraft production environment only\(^\text{30}\); currently developed features for the first version of the prototype will be added there at a later stage)
  - Account information is protected by a password, which is encrypted and DataGraft does not store the non-encrypted version. Furthermore, current deployments of DataGraft use SSL certificates enabled through the CloudFront CDN on AWS. Other configurations of SSL are also possible if necessary;

- OAuth2 – DataGraft uses a standard implementation of RFC 6749 – token-based authorisation layer for control of client access to resources;

- Encrypted cookies – For both DataGraft and Grafterizer, front-end cookies containing session information are exchanged between the web UI and the back-end. This cookie stores a session identifier and encrypted session data when users are logged in to the DataGraft portal.

Enrichment Database and data workflows and publishing: as was discussed in section 2.2.2 and 2.2.5, a production Corporate environment to deploy the prototype Big Data Runtime infrastructure is simulated. Thereby, the Enrichment Database (ArangoDB) and Processing sub-component are deployed in an isolated network, which is behind a SINTEF-operated VPN server. Credentials to access the secured environment are only issued to the stakeholders of the EW-Shopp platform and the network and services are not exposed directly through the Internet.

Data Analyser: There are two key sensitive aspects of analyser operation: dataset transfer to QMiner at learning and transfer of data examples to/from QMiner during prediction. The model built by the system is not sensitive as it does not retain any of the data. For learning access to ArangoDB data is secured through VPN provided by SINTEF. If direct transfer of data files is needed, JSI runs a FTPS server on the machine provided for the platform. Finally, HTTPS is used to secure requests and replies during prediction. Should needs for any more sophisticated security measures arise during pilot deployment, they are simple to add due to flexible nature of QMiner as a library part of the Node.js ecosystem.

\(^{30}\) https://datagraft.io/
**Data Reporter:** The *Knowage* suite has an extensive permissions and authorizations management, which allows to precisely configure what each user can perform and see. This is represented by a login based system, using a classic username and password approach, but it is designed to be easily integrated with the most common Single Sign-On (SSO) systems such as CAS, OAuth2 and FIWARE IdM Keyrock. Regarding the connection to *ArangoDB*, the security of the connection is ensured by the use of a VPN connection (based on OpenVPN release 2.4.4) provided by SINTEF.