



DT4EU
CONSORTIUM

TRASYS International

Architecture of TO BE

Scoping Study of Reportnet 3.0

Date: 29/11/2018

Doc. Version: v2.2

Document Control Information

Settings	Value
Document Title:	Architecture of TO BE
Project Title:	Scoping Study for Reportnet 3.0
Document Author:	Aris Tzoumas
Project Owner (PO):	Stefan Jensen
Business Manager (BM):	Jonathan Maidens
Solution Provider (SP):	Søren Roug
Project Manager (PM):	Christian-Xavier Prosperini
Doc. Version:	V2.2
Sensitivity:	Public
Date:	29/11/2018

Document Approver(s) and Reviewer(s):

Name	Role	Action	Date
Persefoni Kampa	Business Analyst	Review	13/12/2018
Aris Vahlas	Senior Architect	Review	13/12/2018
Jonathan Maidens	Business Manager	Review	14/12/2018
Søren Roug	Solution Provider	Review/Approve	14/12/2018
Christian-Xavier Prosperini	Project Manager	Review/Approve	14/12/2018
Stefan Jensen	Project Owner	Review/Approve	14/12/2018

Document history:

The Document Author is authorized to make the following types of changes to the document without requiring that the document be re-approved:

- Editorial, formatting, and spelling
- Clarification

To request a change to this document, contact the Document Author or Owner.

Changes to this document are summarized in the following table in reverse chronological order (latest version first).

Revision	Date	Created by	Short Description of Changes
1.0	15.11.2018	Aris Tzoumas	Initial version
2.0	29.11.2018	Aris Tzoumas	1. Minor document changes based on received comments 2. Updated descriptions contained in the requirements list 3. Added an appendix containing a preliminary Gap Analysis.
2.1	07.12.2018	Aris Tzoumas	Minor text corrections
2.2	13.12.2018	Aris Tzoumas	Updates on “INSPIRE harvester”, “INSPIRE mappings” and “Candidate technologies” sections

Configuration Management: Document Location

The latest version of this controlled document is stored in <https://projects.eionet.europa.eu/reportnet-3.0/library/03-executing/02-projects/01-scope-study/scoping-study-deliverables/architecture-be>.

TABLE OF CONTENTS

1. INTRODUCTION	4
2. RESULTING ARCHITECTURE.....	4
2.1. Principles.....	5
2.1.1. Follow EU design guidelines	5
2.1.2. Secure by design	5
2.1.3. High quality of delivered software	6
2.2. Data Architecture	7
2.2.1. Submission Agreement.....	8
2.2.2. Dataset.....	9
2.2.3. Data schema	10
2.2.4. Data visualisation.....	13
2.2.5. Data flow	14
2.3. Application Architecture	19
2.3.1. Microservices.....	20
2.3.2. Asynchronous Messaging	20
2.3.3. Record Store	21
2.3.4. Containers.....	22
2.3.5. Integration and Interoperability	37
2.4. Technology Architecture	38
2.4.1. Database infrastructure	38
2.4.2. Application infrastructure	39
2.4.3. Candidate technologies	42
3. REQUIREMENTS MAPPED TO ARCHITECTURE	43
4. APPENDIX 1: FULL DOMAIN MODEL	57
5. APPENDIX 2: C4 DIAGRAM KEY/LEGEND.....	58
5.1. Persons.....	58
5.2. Boxes (systems & containers).....	58
5.3. Lines (relationships & protocols)	59
6. APPENDIX 3: MAIN CONTAINER DIAGRAM	60
7. APPENDIX 4: PRELIMINARY GAP ANALYSIS.....	61

1. INTRODUCTION

The Software Architecture Document for the scoping study of the Reportnet 3.0 project aims to outline the high-level vision of the target software architecture of the project and describes the principles as well as some important aspects of the future system from different viewpoints.

The purpose of this document is to set the baseline for the design of the future system, which shall be completed in the next phase of the Reportnet 3.0 project. The resulting architecture has been primarily driven by the collected requirements and is in line with the “Reporting Directly to a Database” Feasibility Study conducted for Reportnet 3.0 by EEA.

2. RESULTING ARCHITECTURE

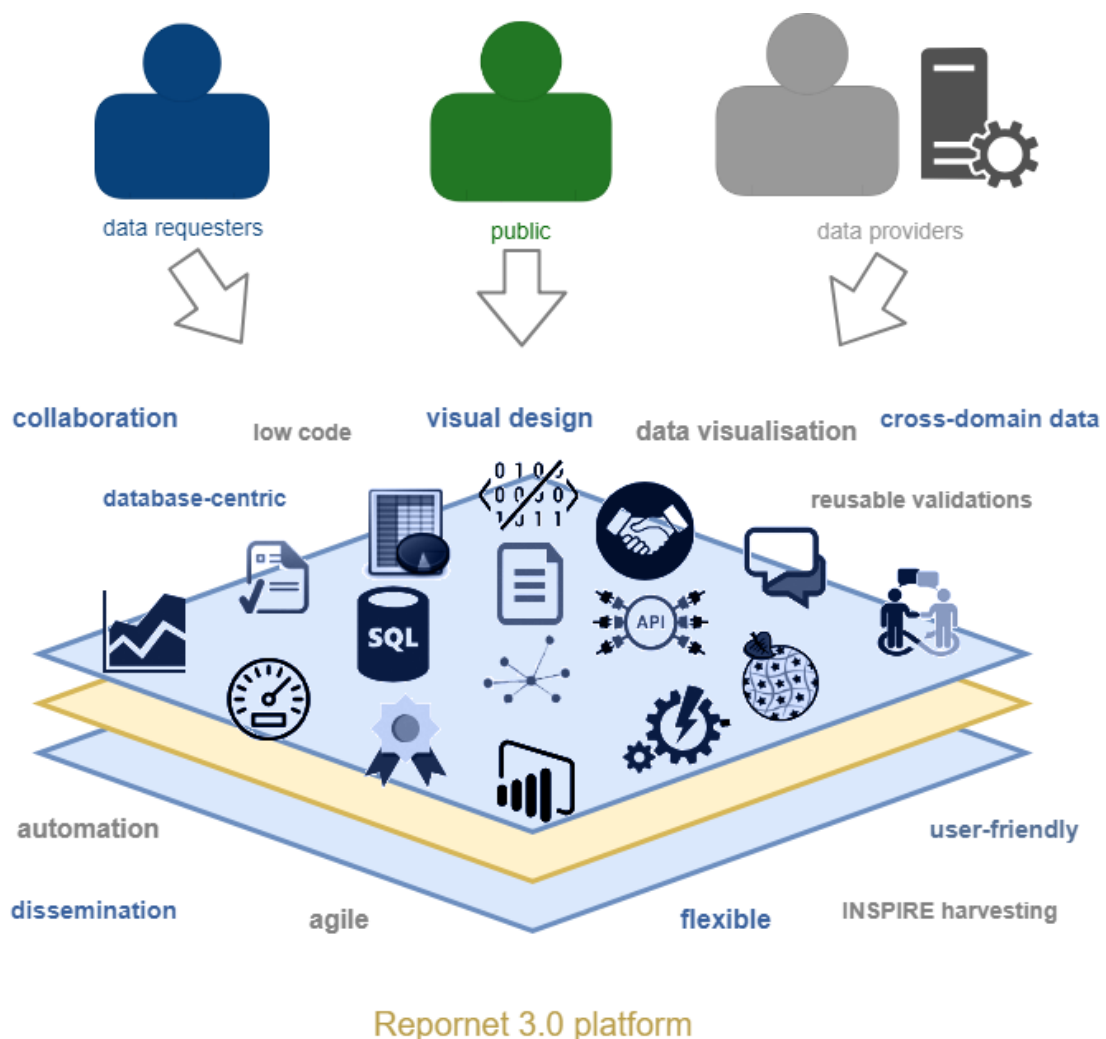


Figure 1 - Reportnet 3 platform overview

Reportnet 3.0 is envisioned as a centralised e-Reporting platform with its main purpose being to simplify and streamline the data flow steps across all environmental domains. Reportnet 3.0 will act as a central hub through which all e-reporting activities will be performed. The main highlights of the new platform include:

- Low-to-zero code platform purpose-built to maximise efficiency based on the existing skill set of the involved stakeholders.
- Collaboration during design and reporting, the platform will offer data sharing and communication tools to enable collaboration between stakeholders in various steps of the data flow.
- Database-centric from the ground up, natively supporting tabular structures aiming to simplify all steps of the process until the generation of the various data products.
- Flexibility on data delivery methods e.g. via user interface, file import, web-service or INSPIRE harvesting.
- Interoperability with other EU Reporting frameworks (e.g. INSPIRE) or international frameworks with strong EU involvement (e.g. SEIS).
- User-friendly and auto-generated web-forms helping reporters to provide data using a familiar interface resembling that of a spreadsheet.
- Cross-domain data integration & easy versioning of other reference data.
- Visual schema design with reusable validations and data visualisation possibilities offered at design time.
- Agile data flow implementation process allowing for quicker design, test and assessment cycles of new data flows or updates on existing ones.
- Automated data collections and streamlined generation of European datasets.

2.1. Principles

The following principles need to be respected during the implementation phase of the project.

2.1.1. Follow EU design guidelines

Statement

The new system should follow the existing EU guidelines for websites and proposed design principles. Indicative style guides can be found under the following URLs and have to be explored upon implementation:

- https://ec.europa.eu/info/resources-partners/guidelines-websites-under-eceuropaeu_en
- http://blogs.ec.europa.eu/eu-digital/design-principles_en
- <https://eui.ecdevops.eu/>

Rationale/Implications

Reportnet 3.0 should provide a modern look and feel, consistent with the Commission's coherent web presence. A user-friendly and output-oriented reporting platform will be beneficial for all stakeholders of the new system. The Commission's eUI platform shall also be assessed whether it can be used and to what extent for the new system's user interface.

2.1.2. Secure by design

Statement

Reportnet 3.0 should be designed from its foundation to be secure. OWASP secure coding practices shall be respected. Compliance

with all the relevant requirements for the Commission Decision 2017/46 on the security of communication and information systems in the European Commission is also mandatory.

Rationale/Implications

Security is a very important concern for any information system and should not be considered an afterthought, but should be incorporated in the software's design and implementation phases.

The system's developed security mechanisms shall be verified regularly through a well-established security testing process. Ideally, security testing must be automated and tightly integrated with the software delivery process as well.

Finally, the security plan shall be documented and accessible to the users.

2.1.3. High quality of delivered software

Statement

The developed system's quality shall be assessed and verified consistently before the release of each software iteration or maintenance release.

Rationale/Implications

The new Reportnet 3.0 platform will simplify the e-Reporting experience for all stakeholders. However, from a technical point of view, it is expected to be a highly challenging and complex software system and it is important to have its quality controlled throughout the system's lifetime in a reliable and cost-efficient manner.

Therefore, a modern and holistic development approach is proposed for the new system which enforces tight collaboration and a shared process for analysts, developers and testers which ensures high levels of software quality, namely the behaviour-driven development (BDD) process. In this process, the expected software behaviours (scenarios) are to be specified in a logical language that everyone can understand and verified during the software delivery process through automated acceptance tests. Although the initial cost of test automation is high compared to the traditional manual user acceptance testing, it is an investment which shall pay off in the long run.

2.2. Data Architecture

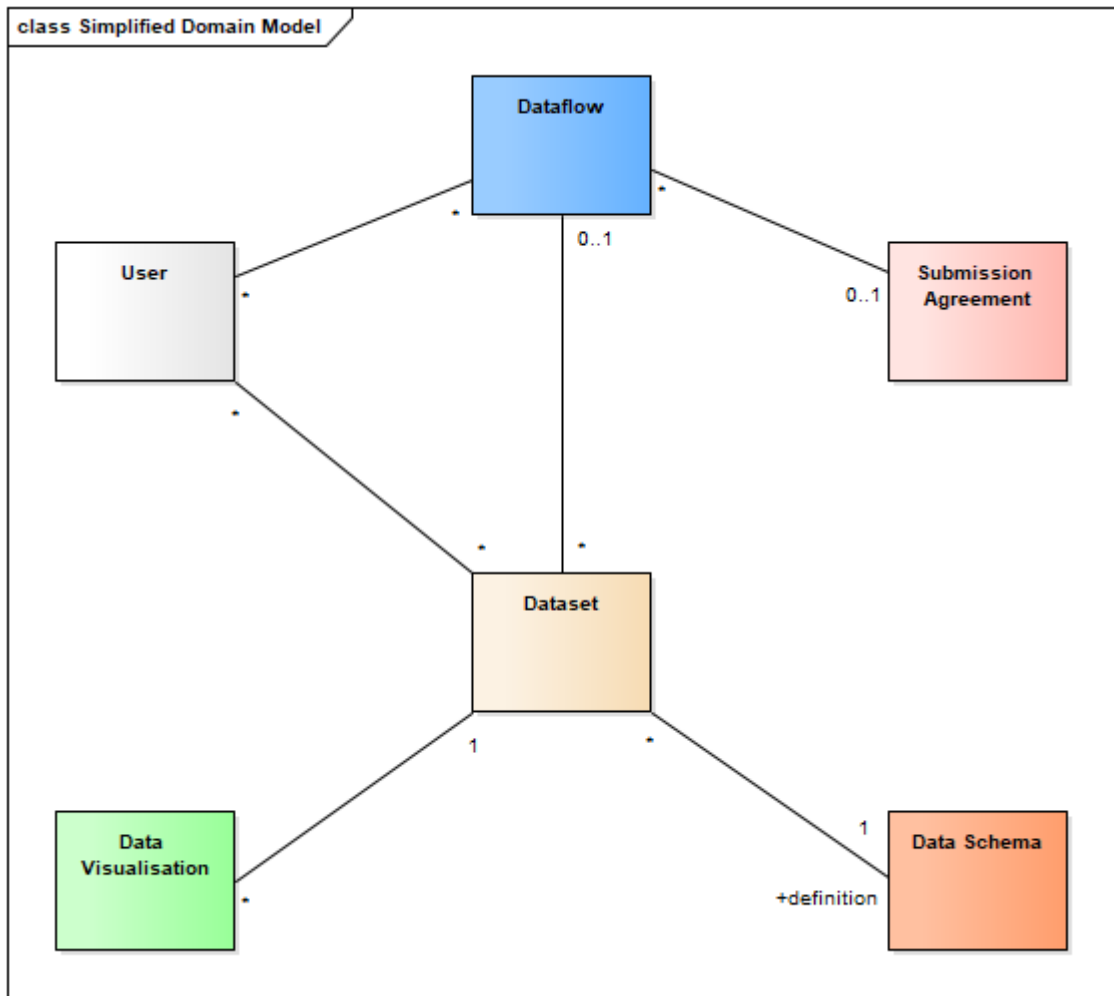


Figure 2 - Conceptual data model of Reportnet 3.0

The conceptual data model of Reportnet 3.0 is presented, a simplified version of which is included above. The purpose of such a model is to highlight the core concepts of the new system and their relationships. In the following sections, partial data model diagrams will be presented for each one of the core concepts with the purpose to assist the reader while reading the accompanying descriptions. The full conceptual model is available for reference in Appendix 1 of this document.

2.2.1. Submission Agreement

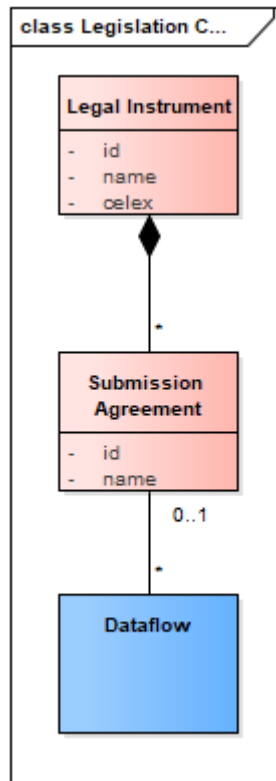


Figure 3 - Legislation context diagram

A submission agreement refers to the legal obligation of an entity (country or company) to report on environmental information at either European or National level. In its legal form a submission agreement defines what needs to be reported, by whom and when, and is typically derived from a European or another National or International legal instrument. Submission agreements and legal instruments are modelled and shall be managed within the system by specific user roles that remain to be defined, however the concept is fully materialised only with the help of additional concepts such as data flows covering the aspects of "who" and "when" and datasets ultimately describing "what" needs to be reported. More details on these additional entities shall follow later in the document; in the meantime, it should be enough to know that for the system a submission agreement acts essentially as a grouping and retrieval mechanism for another core concept of the system, data flows.

2.2.2. Dataset

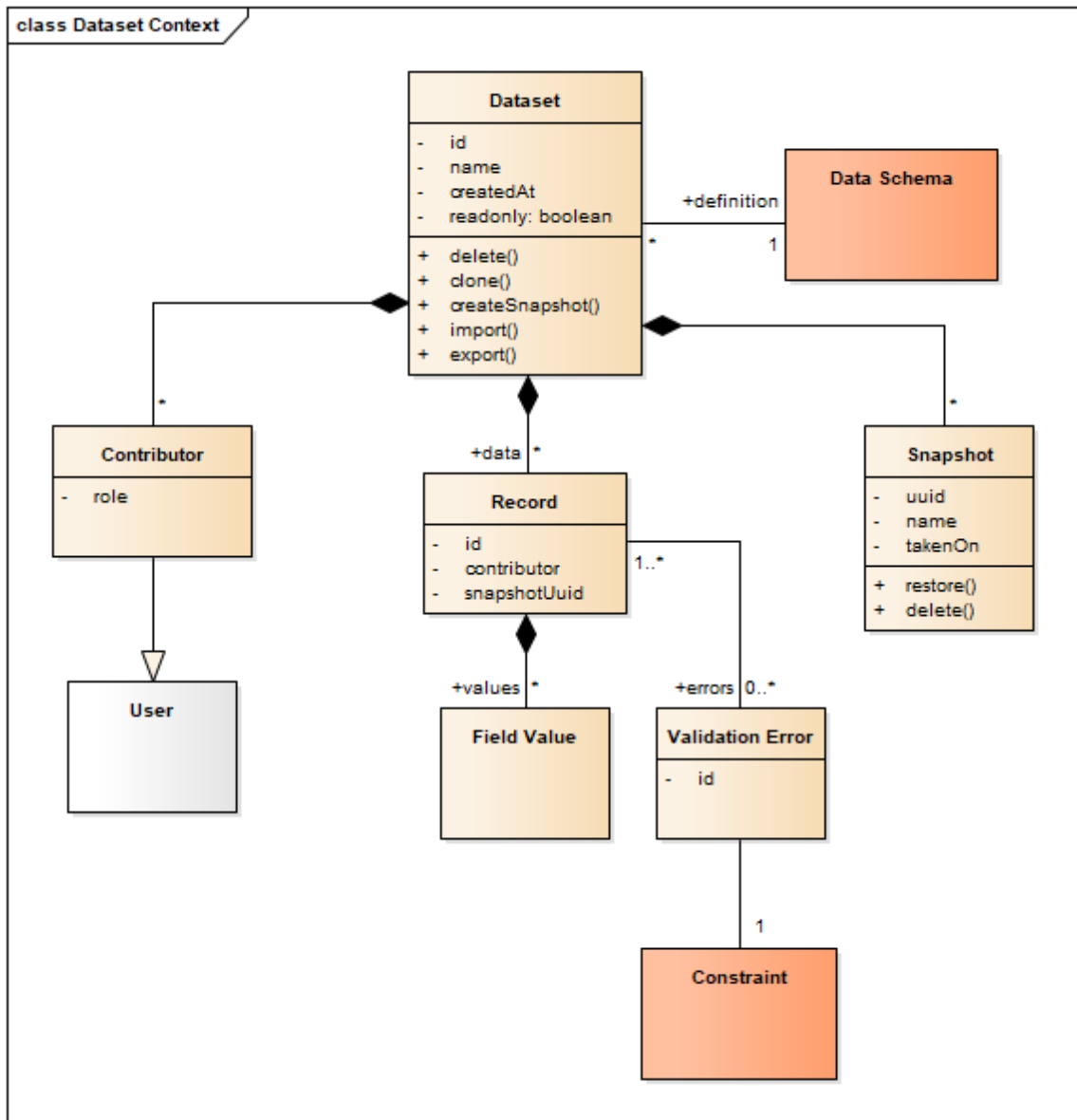


Figure 4 – Data set context diagram

The dataset is a very important concept in the new system as in its generic form it captures all important aspects of one of the system's most valuable assets, reported data in the various reporting steps. A reporter might prefer thinking of a dataset as an augmented spreadsheet, whereas for a database expert a dataset resembles a relational database with the ultimate truth lying somewhere in between. First and foremost, the purpose of a dataset is to store reported data as records in a tabular form. Not everybody is allowed to act on datasets but specific user permissions are defined in the dataset's list of contributors and depending on a user's role, one might be allowed to perform specific actions on a given dataset (e.g. view, edit, create snapshot, comment on data or change data structures). Datasets allow for incremental accumulation of reporting data and as new data arrive automated quality checks (validations) get executed which may lead to the addition of validation errors on data records as an indication of possible data quality issues. Data can be added to a dataset through various methods (spreadsheet-like user

interface, Restful APIs¹ or INSPIRE harvesting) and different input formats (json, csv, xml) depending on the capabilities and preference of its contributors. The owner of a dataset is allowed to create snapshots of a dataset, which represent an immutable version of the dataset's contents at a specific point in time. Such snapshots may have multiple uses, apart from being the primary reporting delivery vehicle as we will discuss in the next sections, they are also a means of saving an intermediate version of a reporter's work which may be restored at any time giving him plenty of room for experimentation. A snapshot doesn't exist in a browsable form, but pretty much like a database dump a user has to restore a dataset's snapshot first in order to be able to actually view its contents and resume working on the backed up data. Snapshots apart from data also capture the structure of the dataset at the time the snapshot was taken.

2.2.3. Data schema

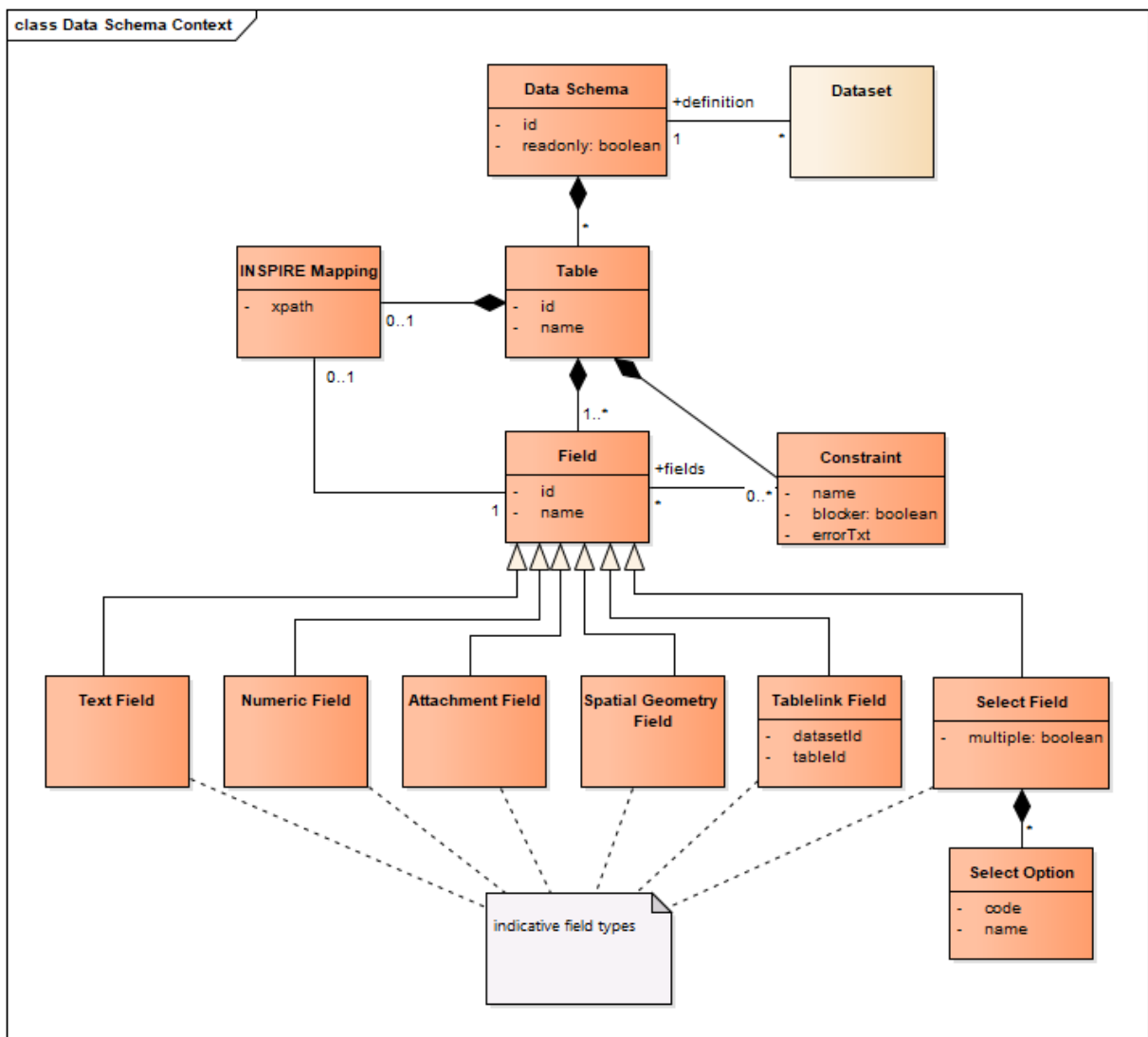


Figure 5 – Data schema context diagram

¹ RESTful API: web service based on representational state transfer (REST) technology, an architectural style and approach to communications often used in web services development

Datasets are primarily meant to store data, but before being able to store any kind of data, the designer of a dataset needs to define the expected format (schema) that the data should follow. This task entails defining tables (sheets) and then for each table deciding its fields (columns), their names and types. The system is expected to support a number of different field types and apart from simple types, such as text and numeric fields, more advanced field types shall be readily available when designing a dataset such as:

- **Attachment fields** holding both file content and metadata;
- **Spatial fields** containing points or geometries;
- **Table-link fields** referencing data from other tables belonging to the same or even another dataset (cross-domain);
- **Selection fields**, only allowing values from a restricted set of predefined values (reference data).

As much as a dataset's field type might bear a strong resemblance to a relational database's data type, the new system focuses on "business" field types aiming to make a dataset's design and by extension the data entry experience as simple and straightforward as possible. This means that under-the-hood a business field might translate to multiple columns or even tables when it will be mapped to its underlying physical relational database model, however the architecture's goal is to mask this complexity away from the end-user and only expose such technical details to the appropriate audience, i.e. operators of downstream systems at EEA such as FME, Tableau or the Dissemination platform who will consume the reported data in order to create bespoke reports and final data products on-demand.

2.2.3.1. Constraints

Apart from some implicit constraints that will be automatically enabled as dictated by the type of some special fields such as table-links (implying referential constraints), selections (implying a restricted set of allowed values) and numeric fields (implying numeric characters only), a designer may also wish to explicitly define additional constraints on a table during schema design such as

- required fields;
- non-empty fields;
- unique fields;
- expected regular expression patterns on textual fields;
- etc.

The system shall make it easy for designers to declare such constraints along with meaningful error messages and the intention of such constraints will be to provide valuable feedback on the quality of the reported data during the data collection and acquisition steps. Violation of the aforementioned constraints shouldn't necessarily prohibit data records from being added to a dataset - which would be the case if the same constraints were actually defined in a relational database, instead the system should provide all necessary tools (error dashboards) and visual cues to help users locate, understand and fix any blocking validation errors.

Complex business validations

To a great extent, it is expected that with this set of explicit and implicit constraints, Reportnet 3.0 will elegantly cover a large part of the quality checks currently existing in Reportnet 2.0. However, it is also expected to have requirements for more complex validations that a reporter won't be able to express with the platform's restricted constraint vocabulary. Such complex validations are expected to be custom-developed outside of the system as part of the data flow's implementation in external systems (eg. FME) with the sole responsibility of Reportnet 3.0 being

to allow the seamless and uniform integration of all external validation plugins with the rest of the platform. Technically, such a responsibility could be fulfilled by providing the necessary hooks in terms of events that might trigger an external validation process and finally the necessary APIs to update the system with the resulting validation outcome.

2.2.3.2. *INSPIRE Mappings*

When the service will provide the dataset encoded according to an agreed harmonised data model (data structure), e.g. INSPIRE data models and XSDs, the semantic mapping between the dataset and the reporting database could be developed in advance on the Reportnet 3.0 side, and used on all such datasets following this model. When the service provides a dataset encoded following other data structures, the semantic mapping will have to be done by the experts on the reporters side, who would understand the data and the structure. In case of INSPIRE datasets and services, an important point is the verification if the dataset is encoded according to the expected INSPIRE data model and encoding rules. This could be done in several ways, e.g. by applying quality control checks (internal) or by using INSPIRE Reference validator (external), considering also other requirements such as performance and automatisisation. Regarding identification of INSPIRE data models and schemas for the semantic mapping, indeed, this could be agreed during the design of the reported database. This way, data reporters will know what INSPIRE data models are expected and what kind of data is foreseen to be mapped and further extracted / transformed from the INSPIRE datasets (e.g. INSPIRE dataset might include data that is not directly requested and used in the reporting data flow).

2.2.3.3. *Schema changes and side effects*

A dataset may start receiving data as soon as one table with one field is defined in it. Changing the schema of a dataset with data already added to it will not be prohibited by the system in principle², however some side-effects should be expected, such as:

- When changing a field type to another incompatible type (text to numeric) or adding a new constraint on an existing field, a flood of validation errors are expected to be triggered for most of the table's records indicating the sudden incompatibility of the existing values with the performed change. Data loss could also happen in case of fields not permissive of invalid values.
- Restoring a previous dataset snapshot will effectively revert any schema changes performed after the snapshot was taken.

² As we will explain later on, different dataset types will exist in Reportnet 3.0 and schema changes will only be possible for a small subset of these dataset types.

2.2.4. Data visualisation

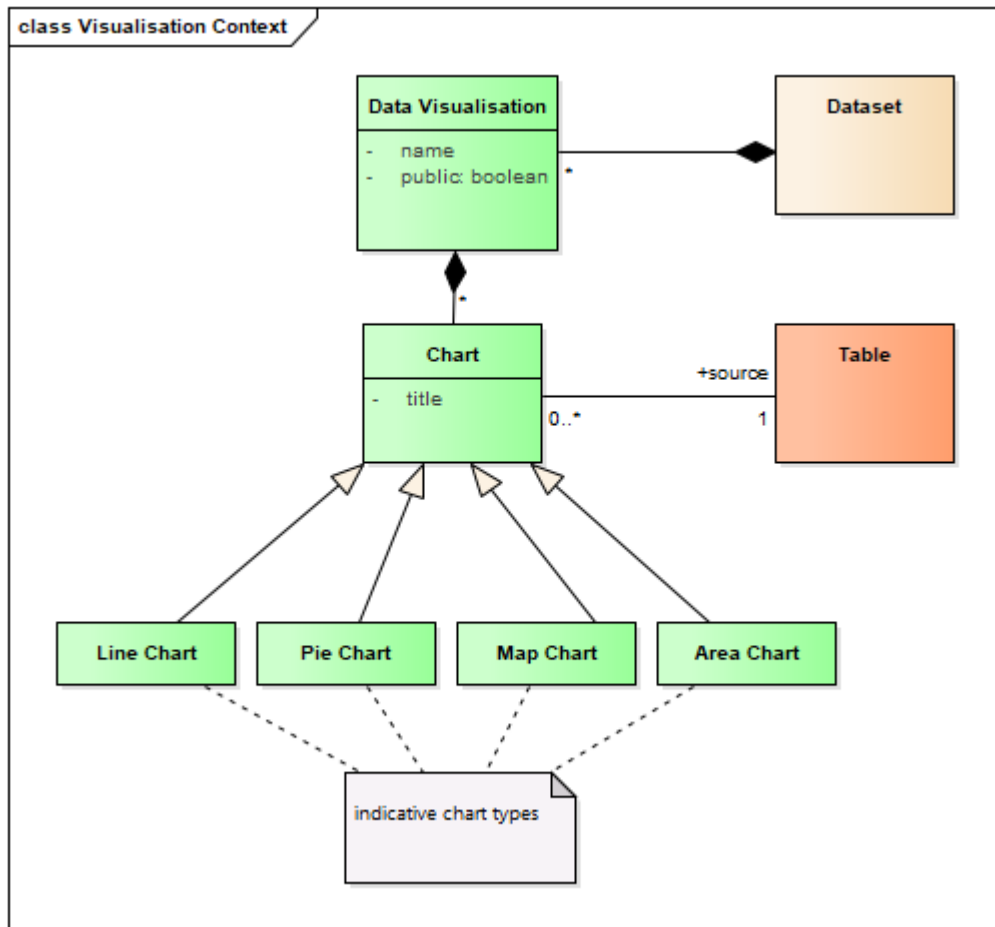


Figure 6 – Visualization context diagram

Although datasets hold valuable information, this information is kept in the form of records inside tables which is an inefficient communication format for quickly analysing and reasoning about data. Thus, Reportnet 3.0 shall enable its users to create alternative data representations at will by choosing from an assortment of various types of charts and creating simple but powerful data visualisation dashboards.

These new visualisation options are expected to help all stakeholders in multiple ways

- Data providers will gain a powerful new means of quickly visualising what they have reported so far, making it easier to detect gaps or anomalies in their reports;
- Data requesters will design better data schemas with visualisations in mind since they will be able to test instantly and they will also gain faster insights on reported data even at the initial steps of the reporting process;
- The public will enjoy informative visualisation products on top of data collections and as soon as a European dataset will be released.

Data visualisations are expected to be defined at the dataset level, however visualisation dashboards and their contained charts will be directly dependent on the dataset's schema, a fact which has a number of implications

- A schema change on a dataset could have as a side effect the invalidation of an affected chart;

- Dataset snapshots must also include data visualisation configuration information and restoring a previous snapshot would typically require restoring the snapshotted visualisation configuration as well, at least whenever the schema of the restored snapshot is different from the current one.

2.2.5. Data flow

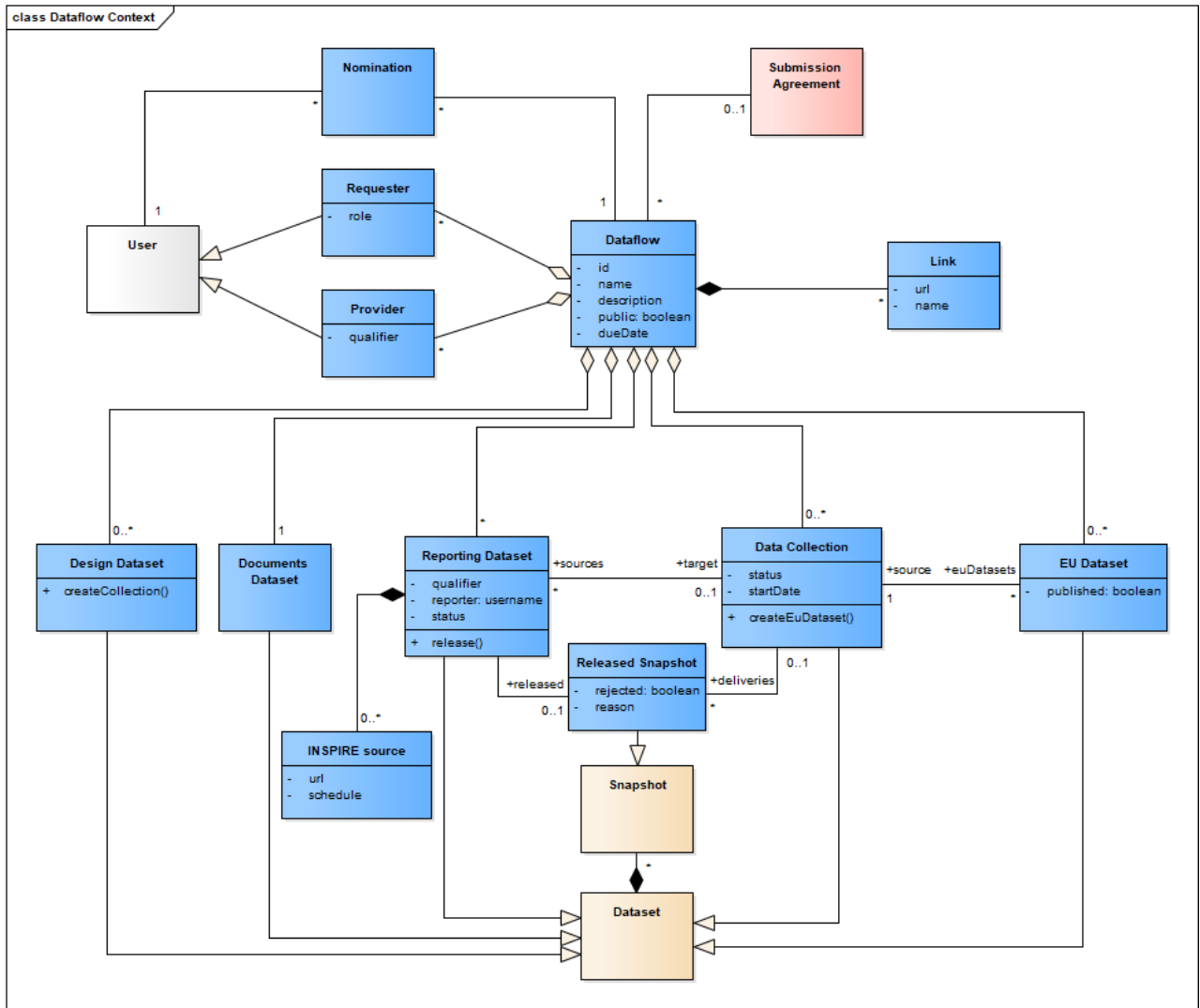


Figure 7 – Data flow context diagram

The data flow as a concept in Reportnet 3.0 encapsulates and orchestrates all steps of the reporting process for the duration of a single reporting cycle, from the first step of defining the data format up to the last step of the creation and publication of a European dataset. Data flows in the system are expected to be created by users holding the data custodian role and a data flow can also optionally be linked to a submission agreement.

Many stakeholders interact with a data flow in the system during its lifetime and we can divide them in two large groups, data requesters and data providers. Data requesters are users with various different roles in the system who assist in defining, managing and supporting a data flow. Data providers on the other hand, are the persons who will be tasked to provide the actual data in the data flow and they may either represent a country or a company, depending on the data flow. A data custodian is the responsible role for defining and managing these two user groups

for the data flows that fall under his jurisdiction. The system is also expected to support a self-nomination flow for data providers during which a user can request to become a data provider in a data flow and this request can either be accepted or rejected by the data custodian.

Many datasets are expected to be created within a single data flow and, depending on their purpose each dataset may exhibit some special attributes and behaviours, hence, different types of datasets can be identified in this context which are analysed below.

2.2.5.1. Documents dataset

A special kind of dataset with its sole purpose being to act as a repository for the data flow's supporting documents, such as guidance documents, definitions and other supporting content. The documents dataset will automatically get created at the same time with a new data flow and it is expected to always have a predefined and unmodifiable schema specifically designed to hold attachments and metadata. Access to the documents dataset will be limited to the data flow's participants (requesters and providers) and only data requesters are expected to have write permissions on it. Furthermore, no data visualisation or snapshotting requirements are envisioned for this special type of dataset, therefore relevant functionalities could be unavailable in the user interface.

2.2.5.2. Design dataset

A design dataset can be created by the data custodian and is essentially a sandbox dataset used prior to the actual data collection with its main purpose to help in designing, testing, validating and finalising the data schema of the data flow. Naturally, the schema of the design dataset is always modifiable but only a subset of the data requester roles shall be allowed to perform modifications on it, the same set who will also be allowed to setup the data visualisation dashboards and experiment with the different visualisation options.

Data on this dataset can be added by any participant as this will be a means of getting familiar with the reporting schema and providing early feedback helping to shape its best format. The dataset will be visible and commenting will be open for all participants of the data flow. To be more precise, a design dataset's contributors' list and roles within will be indirectly controlled by the data flow's own list of requesters and providers. As soon as the participants have agreed on the final report design (schema, validations & visualisations) the data custodian will be able to move on to the next step and create a data collection out of it.

The system isn't expected to impose any restriction on the number of design datasets that can be created within a data flow, it is rather left as a choice to the data flow's custodian who may want to create multiple designs for different purposes. This means that multiple design datasets may exist within the same data flow with different schemas, validations and visualisations defined in them and from every such dataset a data collection may start at any time. Complex reporting obligations, such as the Water Framework Directive could benefit from this feature, as it will be possible in the new system to break down large collections into smaller chunks, each one having a different data structure and lifecycle.

2.2.5.3. Data Collection and Reporting datasets

A data collection is another special dataset designated to gather all data as they get submitted by the data flow's providers. A data collection dataset shares the same schema, validation rules and visualisations as the reporting dataset from which it originated, however its schema is not allowed to change throughout the collection process. No person will be allowed to directly add or edit data on a data collection dataset, instead the only way of moving data into the collection dataset will be through a reporting dataset.

As soon as the data custodian commences a data collection from a design dataset, the system will create one data collection dataset and multiple reporting datasets, one for each data provider of the data flow. The reporting dataset is the reporter's "private" workspace, it is managed exclusively by him and he may even share it with other trusted users for delegating the data entry task. Reporting datasets also share the same schema, validation rules and visualisations with their originating design dataset and their schema shall remain read-only for the full collection period.

Delegated access on reporting datasets

It is a common practice for country representatives to delegate the responsibility of reporting to a number of subordinates, e.g. regional authorities or industry, who own and can submit their data. In this context, each delegated person is considered to be a contributor in the reporting dataset but with special permissions, in the sense that they will be restricted to submit and view exclusively their own data in the reporting dataset. Data providers are responsible for managing delegations on their reporting datasets, they will be able to modify or delete data provided by any delegate and will be able to monitor each delegate's progress through the available status dashboards, as discussed later in this document.

Filling reporting datasets through INSPIRE services

Provided that INSPIRE mappings were defined during the dataset's design phase and the reporter's country has already setup the appropriate INSPIRE node(s) holding relevant data, the reporter will be able to add the appropriate INSPIRE URLs as input sources for his dataset and setup a recurring harvesting schedule (e.g. daily). In the background, the system will schedule the appropriate jobs which shall harvest these sources and attempt to import extracted data into the reporter's dataset automatically.

Releasing reporting datasets

Every data provider is responsible to perform or delegate the task of filling the reporting dataset with the required data using any of the available methods and once done, release it to the data collection. The action of releasing a reporting dataset to a data collection involves the system creating a snapshot of the reporting dataset and importing it to the target collection dataset. The reporting dataset will remain active and open for modifications even after its delivery to the data collection in order to allow for additions, corrections and redeliveries. In case of repetitive release actions (redelivery), previously released data shall be completely replaced by the new data in the data collection dataset, however the release history will be traceable through the reporting dataset's preserved snapshots.

Rejecting a released dataset

Even when no blocking validation errors exist on a released dataset, it may be rejected by the data requester during the manual assessment process. Reporters shall be notified by the system about rejected deliveries, along with an informative containing the rejection reason and they shall be allowed to prepare and release a new, corrected snapshot or reuse a previously (non-rejected) released snapshot, if available.

Data collections and data providers' management

Data providers may be dynamically added or removed in a data flow, even when a data collection is in progress. The action of adding a new provider in this case will trigger the creation of another reporting dataset assigned to the new reporter and this dataset will also be added in the data collection's allowed sources' list, whereas during the removal of a provider the system will retain his reporting dataset but it will be disconnected from the data collection so that he can no longer contribute to the data collection, probably removing any previously released data from the collection too. A disconnected reporting dataset can only be reconnected to the data

collection if the data custodian adds the owner of this dataset to the data flow's list of data providers.

Data collection and visualisations

As already mentioned, data visualisation dashboards will be initially copied from the originating design dataset to both the data collection and the individual reporting datasets, however these visualisations will be editable. Owners of the datasets (data custodian will own the data collection and individual reporters will own the individual reporting datasets) will be able to create new data visualisation dashboards or even edit the existing ones at will.

Status dashboards

Specifically for these two types of data sets the system shall provide some additional reporting status dashboards so that stakeholders can have a quick overview of the data collection process. Data providers will be able to see their own dataset's status dashboards containing aggregations of the submitted data and errors (grouped by contributor in case of collective work), whereas data requesters will be able to consult the data collection's status dashboard and thus supervise the overall collection process having visibility on the status of the collection per data provider.

Data collections and EU datasets

At any time, even before the data flow's due date, the data requesters may jointly decide that the quality and quantity of the reported data has reached an acceptable threshold to justify the creation of a European dataset. European datasets can be created from data collections and the relevant action can only be performed by the respective owner of the data collection. A European dataset is another special type of data set, which is discussed in the next section.

2.2.5.4. EU dataset

A European (EU) Dataset is essentially a branch out of a data collection's snapshot which can be further modified (excluding its schema), by data custodians or data experts for cleansing purposes, who will eventually create additional snapshots as the cleansing process advances. Data visualisations on EU datasets are copied from their originating data collection dataset but they can be edited in order to come up with the final "public" visualisations. Another important aspect of EU datasets is that they are publicly visible in Reportnet 3.0 and together with their public-facing data visualisation dashboards, they consist the primary auto-disseminated product of each data flow. Of course, additional products are expected to be created from an EU dataset and totally diverse products will be required for each data flow, but the processes which will create such bespoke products are considered to be external to Reportnet 3.0. Nevertheless, as soon as any additional data product becomes available, the data custodian will have the option of publishing it in Reportnet 3.0 by including the relevant URL link inside the data flow.

There is no limit in the number of EU datasets that a user can create within the scope of a single data flow, however proliferation of EU Datasets in a data flow is possible to cause unnecessary confusion to interested parties. Thus, selective publishing of EU Datasets is a functionality, which might be useful in the new system.

2.2.5.5. Repeating data flows

A new data flow is expected to be created in the system for the new reporting cycle; however, it is a common use case that a data custodian may wish to use the previous data flow as a template for the new one. But instead of unnecessarily copying all datasets of the previous data flow to the new one, a custodian should be allowed to select which (design) datasets contained in the source data flow will be copied to the new one.

2.2.5.6. Prefilling data

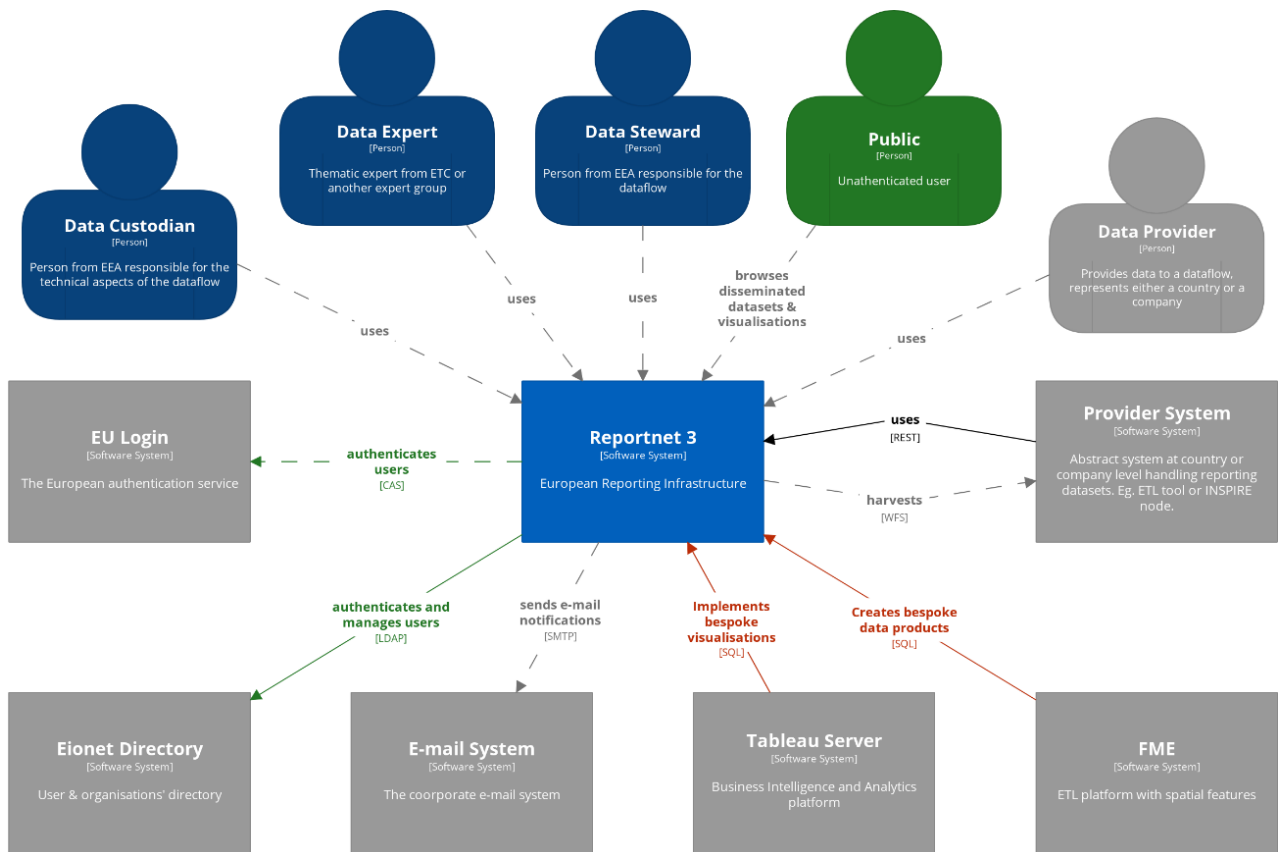
It is also common to have to start a new cycle for a data flow with data prefilled from the previous cycle. The system can support this use case as well, since a data custodian will be able to select an EU dataset from the previous data flow and convert it to a design dataset in the next cycle's data flow. Commencing a new data collection from this new design dataset would then trigger the system to seed each newly created reporting dataset with all records that exist in the design dataset and their qualifier matches the qualifier of the target dataset.

2.2.5.7. Up-to-date data

The same process as the one already described could be followed for up-to-date data as well, meaning that countries would need to "release" their datasets frequently and at regular intervals. However, such a process appears to be inefficient as even for minor changes a full snapshot should be taken and transferred to the data collection, therefore some alternatives could be considered:

1. **Append-only releases:** The countries' reporting datasets will be cleaned up after each (partial) release and repetitive releases will append data instead of completely replacing previously released data in the data collection.
2. **Up-to-date datasets:** Create another special dataset type where all data providers will have direct write access to (only for their own data) and completely skip the "release" concept. This alternative though will introduce extra complexity if the delegation concept needs to remain on the up-to-date datasets as well.

2.3. Application Architecture



System Context diagram for Reportnet 3

Context Diagram for Reportnet

Figure 8 – System context diagram

The core concepts of the application architecture are presented below and further in the document, separate sub-sections provide more details on the individual containers of the system. The key/legend for the C4 diagrams presented in this document is available in

Appendix 2: C4 Diagram Key/Legend, whereas the overall container diagram of the new system is also available in larger (A3) page size in **Appendix 3: Main Container Diagram**.

2.3.1. Microservices

The proposed application architecture for Reportnet 3.0 platform is following the microservices architecture pattern, which structures the system as a set of small, loosely coupled, autonomous and collaborating services. Each service implements a set of narrowly related functions, typically centered on a core domain object of the system, for example, data flows, datasets and notifications are expected to be handled by different services. No prescribed formula exists regarding service granularity, however it shall be noted that too much of fine-grained services usually results in unnecessarily high complexity. The proposed architecture tries to achieve a good initial level of service granularity which can be further adjusted at the latter stages of the project.

Services can be distinguished in two main categories:

1. **Client-facing** services, which handle requests from the application's clients and therefore provide RESTful APIs and
2. **Internal** services, which perform vital functions of the system but remain invisible to external clients.

Apart from the individual functions that each service implements as part of their own bounded context, there are some cross-cutting concerns which must be uniformly provided by all services of the system, such as:

- **Logging:** services must use common log patterns in their log statements to facilitate log aggregation.
- **Health checks:** each service must provide a uniform "monitoring" URL, which can be pinged by a monitoring agent to determine the health of the service. Multiple health checks will compose the overall health status of the service and it is equally important to be able to monitor both the overall health of the service and each individual indicator. For easily identifying and fixing errors, each service should also be in a position to communicate descriptive error messages for failed health checks.
- **Metrics:** services shall expose application metrics, preferably in a uniform format. Application metrics are measurements performed by the service itself, which provide insight into what the application is doing and how it is performing.

Existing open source microservices' frameworks can handle all the above concerns with minimal effort from a developer so that the task of developing a new microservice in the system remains easy and quick.

2.3.2. Asynchronous Messaging

Inevitably, at some point services will need to collaborate with each other in order to implement some higher level business function. To achieve loose coupling between services and a high level of service availability, the proposed application architecture promotes the usage of a message bus as a central architecture block acting as the nervous system for all inter-service communication needs. A message bus promotes an asynchronous communication style for collaborating services and consequently classifies Reportnet 3.0 as an eventually consistent distributed system, nevertheless highly scalable and available.

Different kinds of messages will flow through the various topics of the message bus, however two main messaging concepts, or categories of messages can be identified in the architecture:

1. **Events:** an event message represents something that took place in the system's domain. They are always named with a past-participle verb, e.g. *DatasetCreated* and since an

event represents something in the past, it can be considered a statement of fact and can be used to take decisions in other parts of the system.

2. **Commands:** they are the means by which a change to the domain objects of the system is requested. They are named with a verb in the imperative mood and typically, also include the data type, for example *CreateDataset*. Unlike an event, a command is not a statement of fact, it is a request, and thus may be ultimately refused by the system. It is not uncommon to expect the system to create new events in response to processed commands which convey the system's response to the requested command, either successful or not.

The system's services use both types of messages for collaborating with each other, events as the primary means of populating and maintaining in-memory views of foreign domain objects and commands as a means of requesting operations to be performed on remote services.

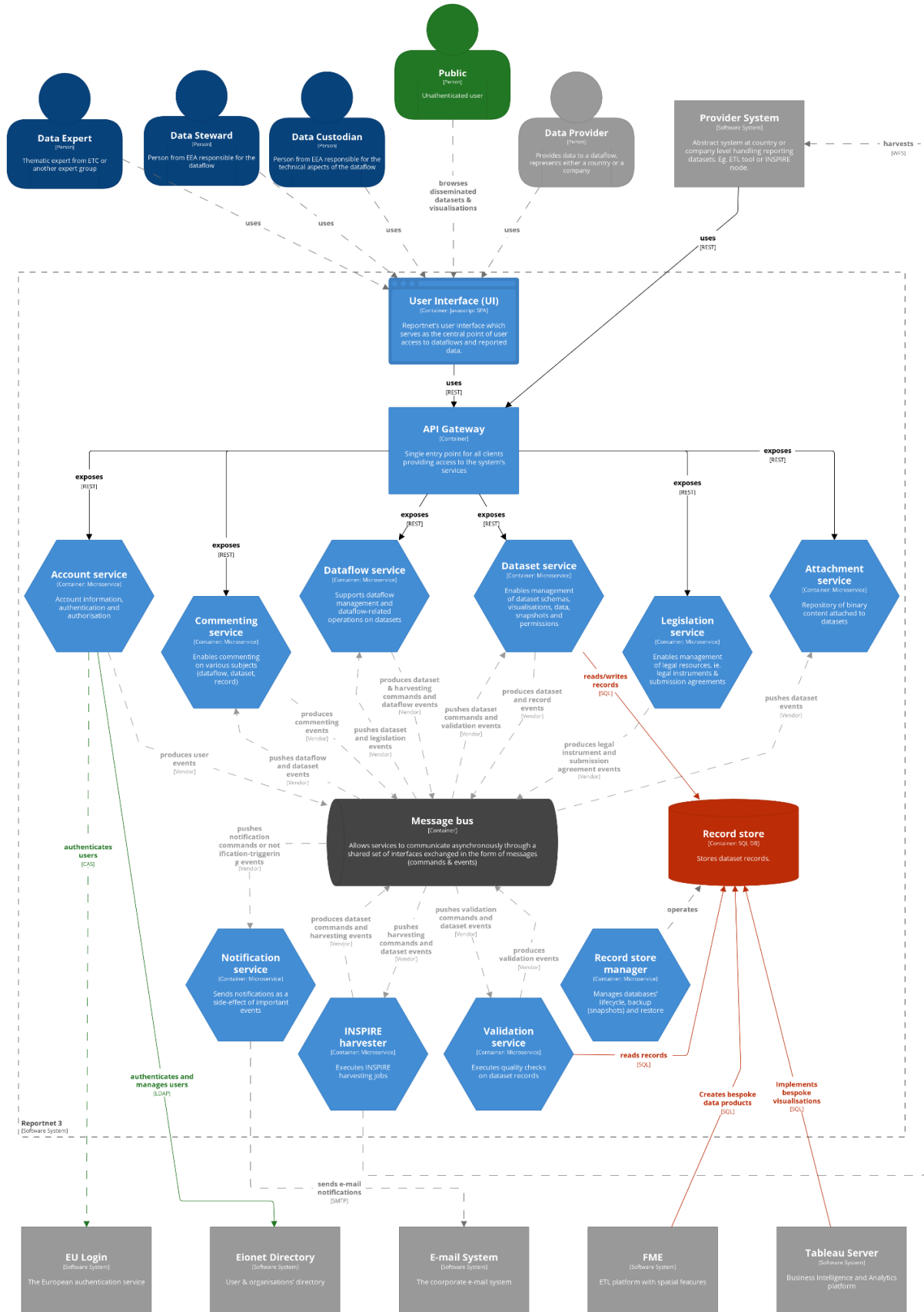
2.3.3. Record Store

Another central building block of the application's architecture is the record store, a relational database cloud service holding the reported data from all reporting steps of each data flow. The strategic decision to keep reported data in relational databases simplifies many aspects of the previous system (Reportnet 2.0), enables a user-friendly dataset design and management experience for the system's stakeholders, straightforward mappings of the resulting structures to a physical relational model and also far easier and effortless integration for the downstream processes at EEA which will eventually produce the final (bespoke) data products of the data flow.

Management of data in the record store shall be exclusively performed through the system's interfaces and direct data manipulation in the database must be strictly prohibited (i.e. via SQL ALTER or INSERT statements) since this would lead to corruption of the system's internal state. However, downstream systems are allowed to directly interface with the database for read-only use cases.

For more details on the unique technological challenges associated with this important architectural component please refer to section 2.4.1 of Technology Architecture.

2.3.4. Containers



Container diagram for Reportnet 3
Main containers

Figure 9 – Container diagram

2.3.4.1. User Interface



Component diagram for Reportnet 3 - User Interface (UI)

Reportnet's user interface which serves as the central point of user access to dataflows and reported data.

Figure 10 – UI Component diagram

Reportnet 3.0 is expected to provide a rich web-based user interface adaptable to the different needs of its stakeholders. On a functional level, the user interface's main functional components are identified and listed below:

- **Legislation management:** Data stewards and data custodians will be able to easily define and manage legislative instruments and submission agreements through the user interface. Additionally, appropriate legislative instrument & submission agreement list pages shall provide easy navigation to the data flows for all users.
- **Data flow management:** New data flows can be created by data custodians who can also manage the relevant participant lists and are guided by the system to perform all necessary dataset operations for the data flow's successful completion. Other users may also access data flows and contained datasets, which they are entitled to access, either through their personalised data flow list page or by navigating to them from the relevant legislative resources. Users shall be enabled by the user interface to gain quick visual overviews of configured data flows in the system, their configuration, progress and current status.
- **Status dashboards:** Special user-friendly status dashboards providing helpful aggregate indicators of the data collection progress such as number of records per table and error statistics will be available through the user interface both for reporting and data collection datasets.

- **Dataset editor:** All different types of datasets will be managed through the dataset editor. Depending on each user's permissions on a dataset and the actual type of the dataset, users can use the editor in order to design a dataset, i.e. define its schema, set validation rules and prepare data visualisation dashboards. The dataset editor also allows users to enter data in the dataset manually via a spreadsheet-like user interface and provides additional tools, which assist users to import data from various different formats (e.g. xml or csv). The same user interface will be used for viewing, sorting and filtering the data in their native tabular form. Finally, the user interface shall offer an API documentation page for each dataset, documenting how external systems may interact and integrate with each dataset.
- **Data visualisations:** It will be possible to visualise the records of a dataset in the user interface through user-defined charts, which will allow users to prepare alternative views of data for intuitive data exploration.
- **Public views:** Unauthenticated users are expected to be able to browse public content in Reportnet 3.0 such as legislative instruments, submission agreements and public data flows in the system as well as published data collections and European datasets along with their public data visualisation dashboards.
- **Search:** The user interface shall provide all necessary search capabilities to its users, allowing for easy retrieval of resources, such as legislative instruments, data flows and datasets.

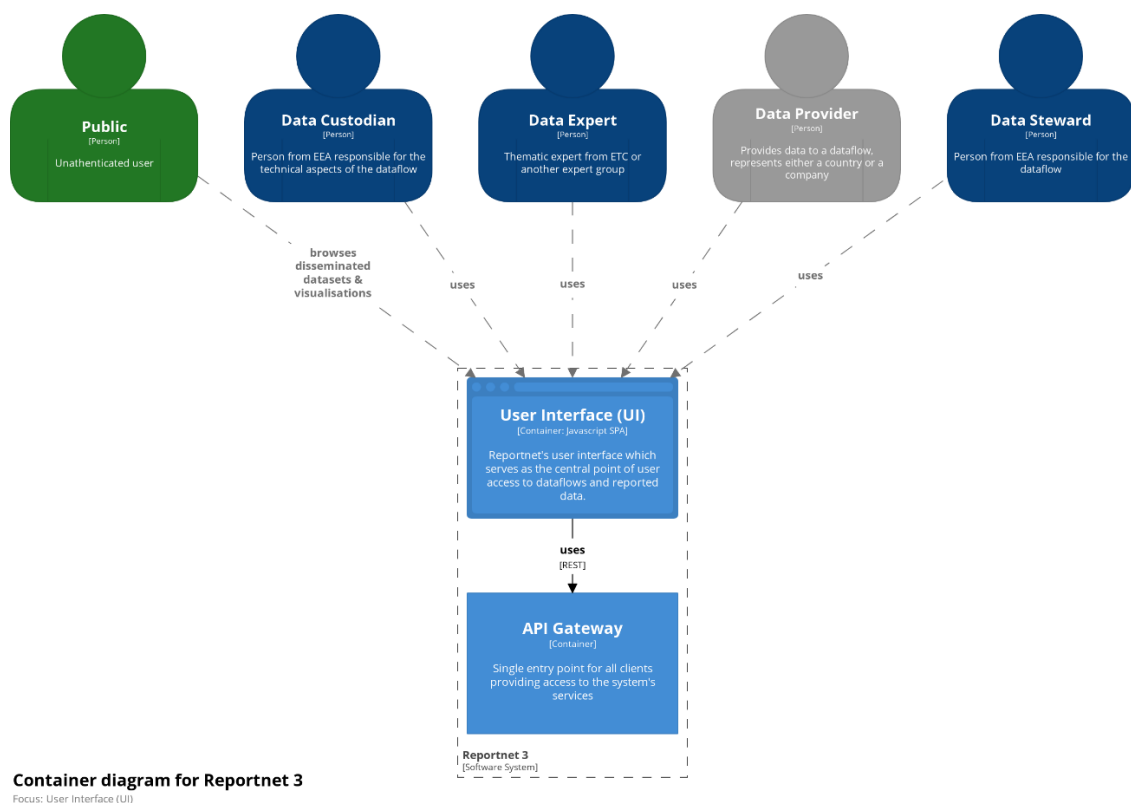


Figure 11 – UI container diagram

The resulting user interface (UI) is expected to be a complex front-end application tasked to support many complex and rich user interactions, therefore it is proposed to be built as a single-page application (SPA) which will run on the user's browser and leverage a modern javascript framework to provide an advanced and seamless user experience resembling that of a native

desktop application. The user interface will have access on the system's service layer through the API Gateway which will expose all required RESTful APIs of the system.

2.3.4.2. API Gateway

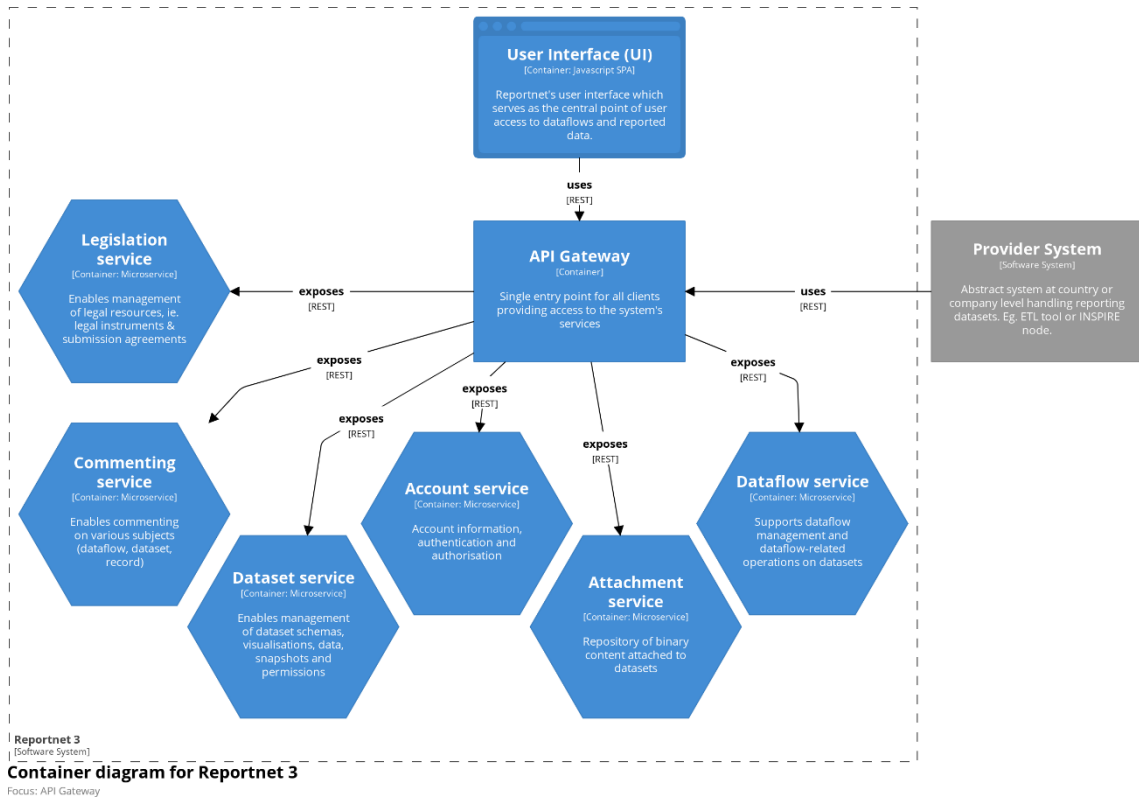


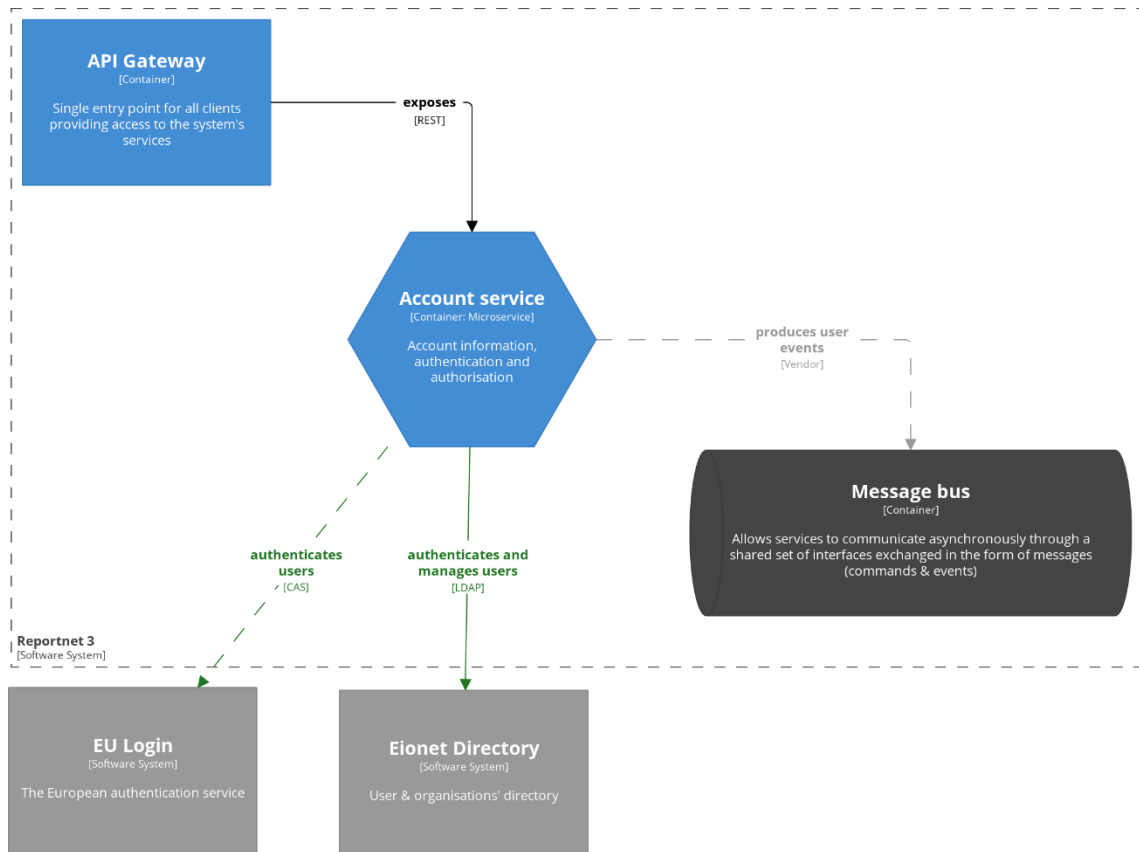
Figure 12 – API Gateway container diagram

The API gateway is the entry point for all clients (i.e. user interface & third-party data provider systems). Clients are not able to access the system's services directly; instead, they call the API gateway, which is responsible to call the appropriate services on the back end. An API gateway is a non-functional component of the architecture which in its simplest form acts as a reverse proxy for backend service APIs but may also perform more advanced transformations or aggregations of responses as required.

Typical cross-cutting functions performed by an API gateway include

- Load balancing of backend services
- SSL termination
- Request logging and analytics
- Rate-limiting
- Cross-Origin Resource Sharing (CORS) handling
- Authentication & session management
- Protocol translation

2.3.4.3. Account service



Container diagram for Reportnet 3
Focus: Account service

Figure 13 – Account service container diagram

Users may authenticate in Reportnet 3.0 either by using their eionet account or through their EU login account. With regards to authorisation, most of the access management is performed internally in Reportnet 3.0 since users' roles in data flows will be defined in the system without requiring any special roles to be defined in advance in an external user directory. Nevertheless, some basic role management still has to be performed in eionet directory, at least for managing data custodians, i.e. users who will be allowed to create and manage data flows in the system.

The account service is the system's container responsible for all account-related functions such as authenticating users against eionet directory or performing the single sign-on (SSO) integration with the EU Login service and synchronising EU Login accounts in the eionet directory. Concerns that follow a user's successful authentication such as management of authenticated http sessions and identity and roles' propagation to backend services are expected to be managed by the API gateway.

Apart from authentication concerns, the account service also provides all necessary read APIs for searching users and getting an account's details through the user interface. Moreover, to support token-based authentication schemes during system requests from external systems, the service provides a token management (create, update, delete) and authentication API. Initially, no additional private store is envisioned for the account service, therefore, API tokens are expected to be persisted in eionet's directory as a custom account attribute.

Message bus integration

Other services will be able to maintain their own customised views of user accounts by consuming user events emitted by the account service. For every new user account created in the system or any change in an existing account the service is responsible to produce the corresponding event (e.g. user created/updated/deleted) and get it published to the relevant message bus topic.

2.3.4.4. Legislation service

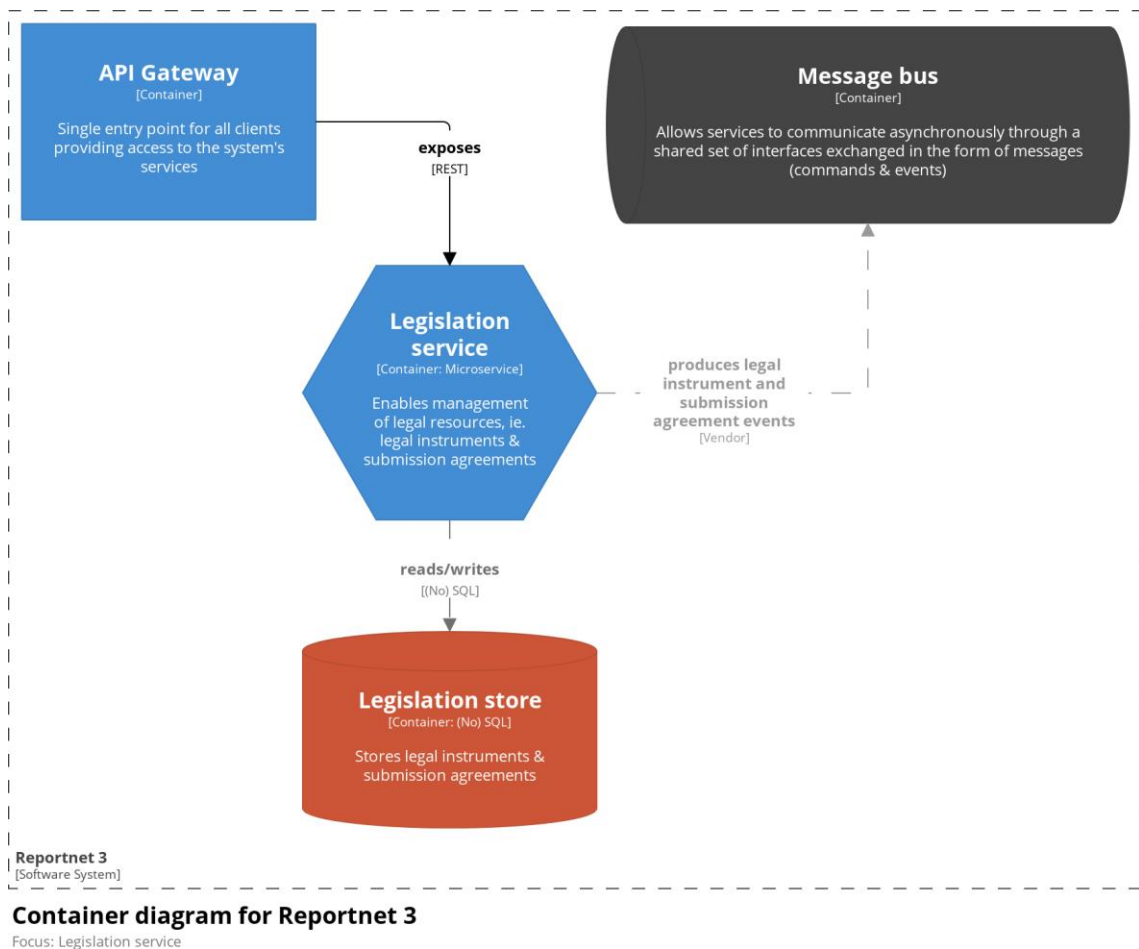


Figure 14 – Legislation service container diagram

The legislation service provides all necessary REST endpoints for performing search and CRUD (create, read, update and delete) operations from the user interface on the relevant legal resources, i.e. legal instruments and submission agreements. Read operations on these resources are permitted for unauthenticated requests, whereas only a restricted and predefined set of eionet roles should be allowed to create and manage legal resources. The service manages and maintains its managed objects' persistent state in a private legislation store for which the underlying technology may either be a SQL or another NoSQL database.

Message bus integration

The legislation service publishes events on two topics in the message bus, one for legal instruments and one for submission agreements to allow other services maintain their own custom views on legislative resources.

2.3.4.5. Data flow service

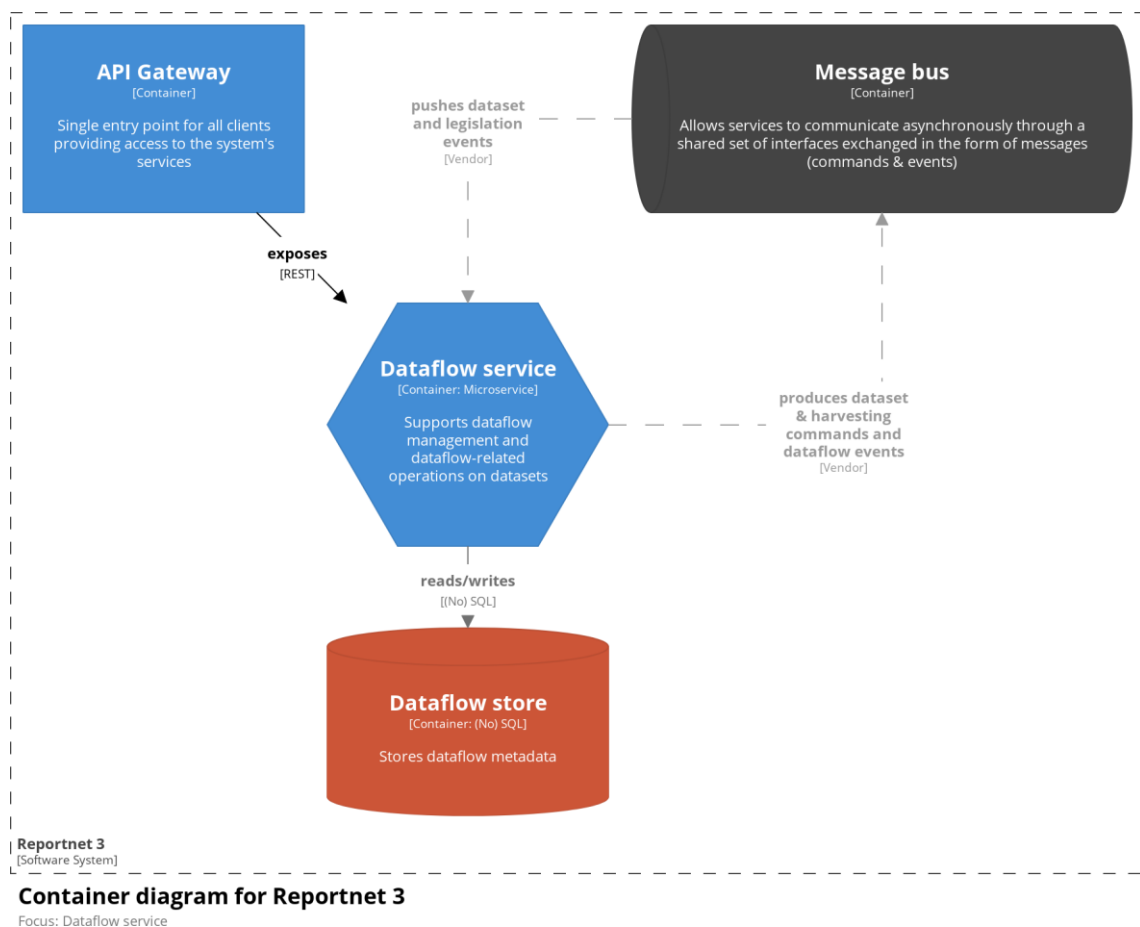


Figure 15 – Data flow service container diagram

The data flow service provides all necessary REST endpoints for managing data flows from the user interface such as:

- listing, filtering and CRUD operations on data flows
- data flow participants' management
- creation and management operations on datasets contained in a data flow, such as:
 - creation of design datasets;
 - creation of a new data collection from a design dataset;
 - creation of European datasets;
 - etc.

The service requires its own private store for persisting data flows and owned dataset metadata such as the dataset id, type, status, INSPIRE sources, etc..

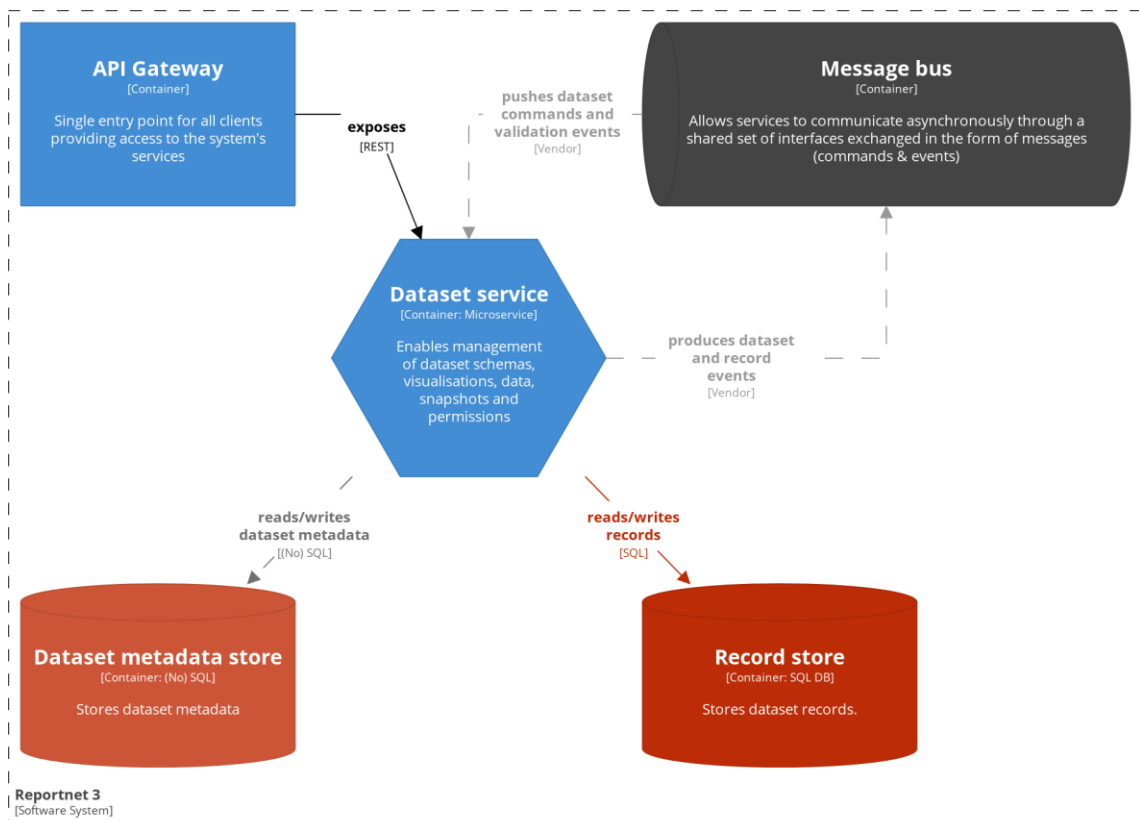
No restrictions are foreseen for the selection of this microservice's underlying database technology.

Message bus integration

The data flow service integrates with the message bus in many different ways:

- Listens for legislation events to maintain an in-memory list of allowed submission agreement identifiers.
- Produces dataset creation commands and listens for corresponding dataset events during the orchestration of dataset management transactions (sagas³).
- Produces dataset harvesting commands using an internal scheduling mechanism which implements the harvesting schedules defined by users in their reporting datasets.
- Produces data flow events (created, updated, deleted, etc.) on the relevant data flow topic to allow other microservices create their own projections on data flows.

2.3.4.6. Dataset service



Container diagram for Reportnet 3

Focus: Dataset service

Figure 16 – Data set service container diagram

The dataset service provides all necessary REST endpoints to allow management operations on individual datasets, such as:

- CRUD operations on the dataset's schema (tables/fields/validations).
- CRUD operations on data visualisation dashboards and contained charts.
- CRUD operations on dataset records.
- Read API for dataset record validation errors.

³ **Saga**: a sequence of local transactions and accompanying compensation actions which are only performed in case of a transaction failure.

The service's record CRUD API will also be accessible (through the API Gateway) by external provider systems and shall be documented using open standards, i.e. OpenAPI (a.k.a. Swagger).

The dataset service uses two different persistent stores:

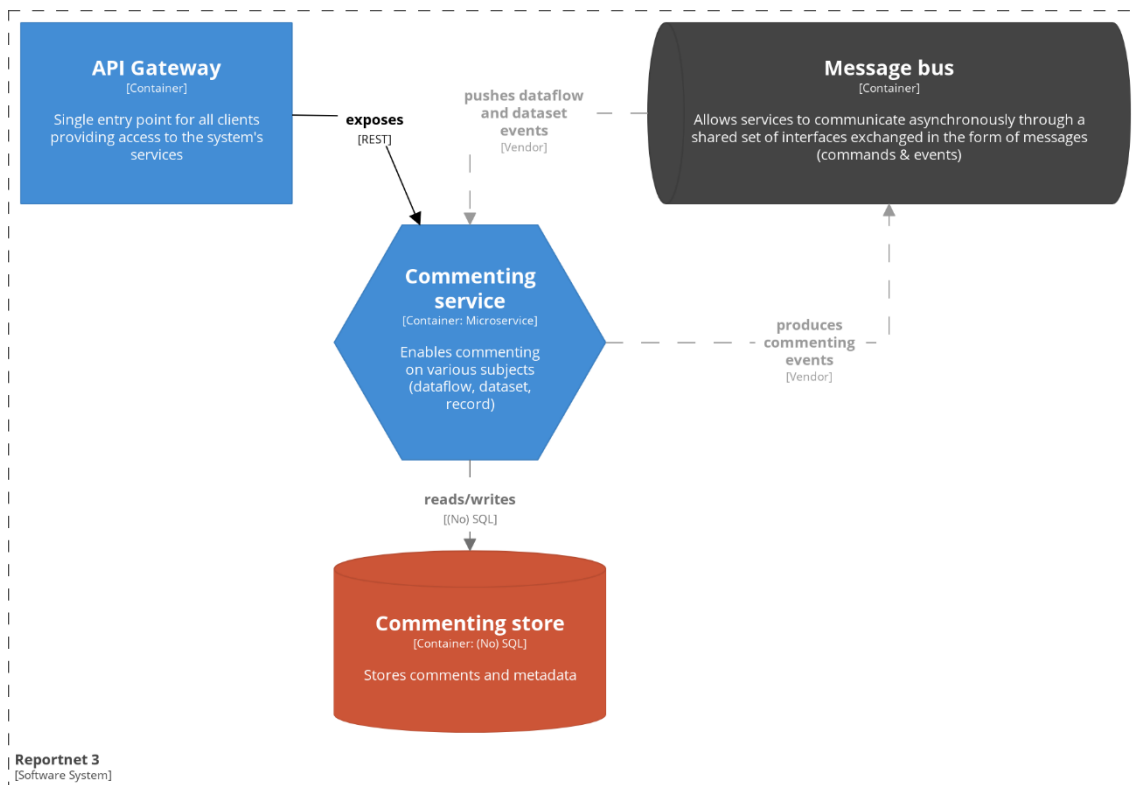
- An SQL database for persisting the actual dataset records in their physical relational form. A dataset's physical model is derived from its business schema and whenever a change occurs on the business schema, the dataset service is responsible to adapt its physical model accordingly.
- A dataset metadata store for persisting various metadata such as the dataset's business schema, the list of allowed contributors, configured data visualisations, the connection url of its corresponding physical database etc.

Message bus integration

The dataset service is a message-intensive service and integrates with the message bus in the following ways:

- Listens for dataset creation commands issued by the data flow service and creates new dataset objects.
- Listens for various other dataset business operation commands such as cloning and snapshotting, which are processed asynchronously.
- Listens for record validation events and persist them to the appropriate record store.
- Produces dataset-level events to allow other microservices maintain their own custom views on dataset metadata or react appropriately.
- Produces more granular record-level events to allow other microservices react to the arrival of new or updated records, e.g. by triggering validation commands.

2.3.4.7. Commenting service



Container diagram for Reportnet 3

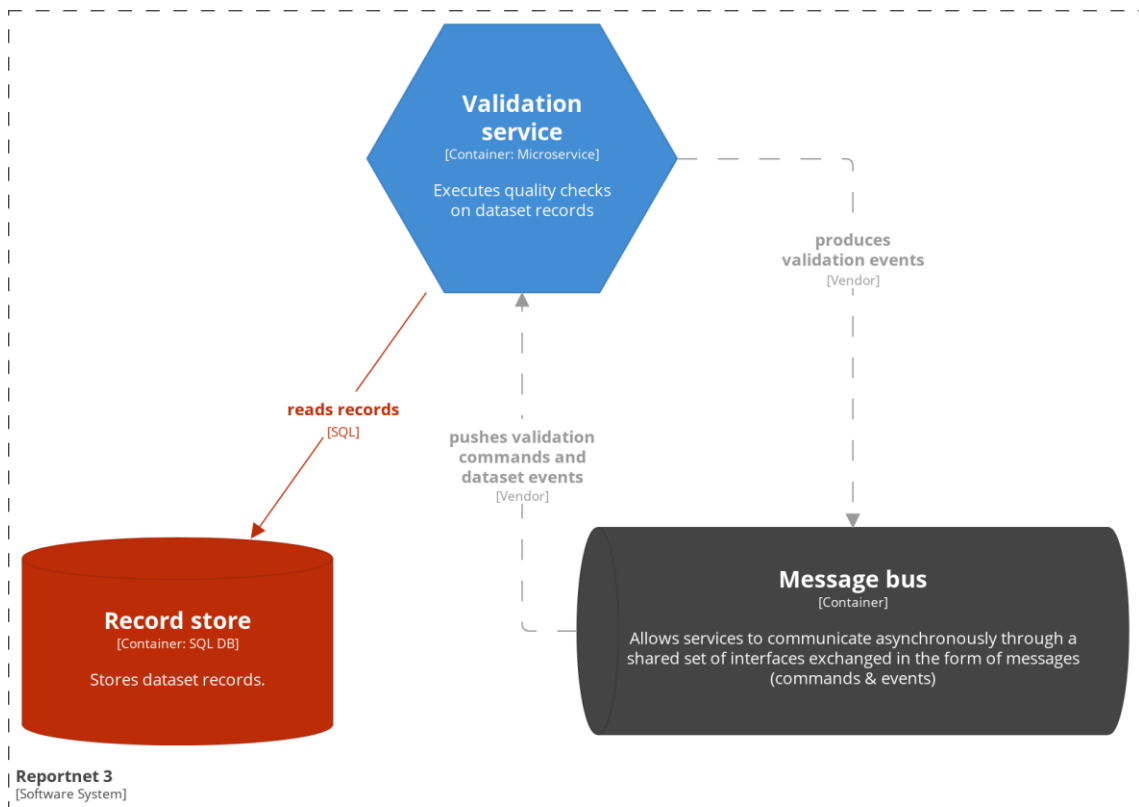
Focus: Commenting service

Figure 17 – Commenting service container diagram

Users may communicate with each other in Reportnet 3.0 in the form of comments on various subjects such as data flows and datasets and the commenting service provides the necessary REST endpoint used by the user interface to enable the commenting capability. The service is using its own internal persistent store for keeping comments and apart from the public facing REST API that it provides, it is also interfacing with the message bus both as a listener and a producer:

- It listens for data flow and dataset events in order to build in-memory access lists and enforce the appropriate access controls on comment creation requests.
- It publishes commenting events to allow other microservices (e.g. notification) react appropriately.

2.3.4.8. Validation service



Container diagram for Reportnet 3

Focus: Validation service

Figure 18 – Validation service container diagram

The validation service is an internal processing component of the architecture. It doesn't expose any kind of public facing API, but it is tasked to execute all natively supported validations on dataset records as defined in the datasets' designed schema. In order to be in a position to readily know what validations need to be executed whenever a new dataset record event is received from the message bus; the validation service has to maintain an in-memory map of datasets and the set of applicable validations for each dataset. Validations shall be executed by the service asynchronously through commands received on the relevant topic from the message bus and after a validation is finally executed against a record the service shall publish a validation event to the message bus containing the outcome. Such validation events are important to both the dataset service, which needs to persist them in the dataset store, and the user interface which needs to update any open dataset editors in case of a validation error.

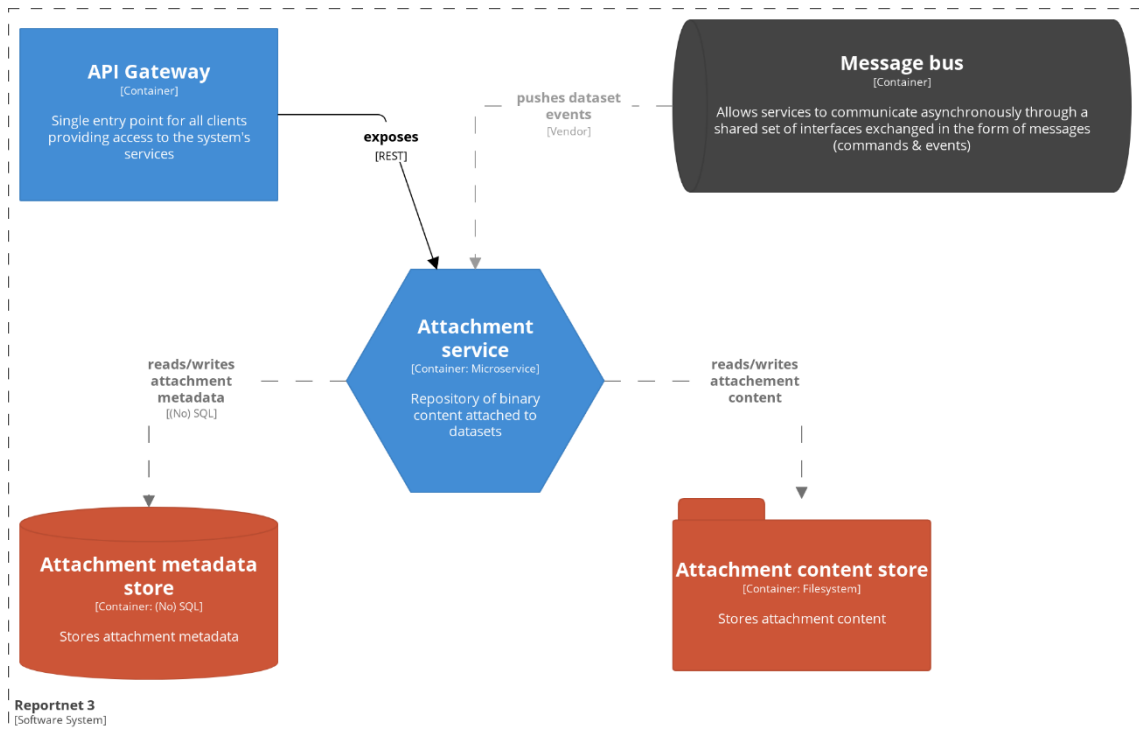
Validations are expected to be triggered in Reportnet 3.0 in any of the following cases

- Create/update/delete operations on a dataset's records
- Create/update/delete operations on a dataset's schema constraints

External business validations

Following the same asynchronous command/event paradigm, the system's default validation capabilities can be extended by any number of additional external validation processes (plugins) which can dynamically extend the system with additional and custom business validation logic pertaining to the special requirements of each data flow.

2.3.4.9. Attachment service



Container diagram for Reportnet 3
Focus: Attachment service

Figure 19 – Attachment service container diagram

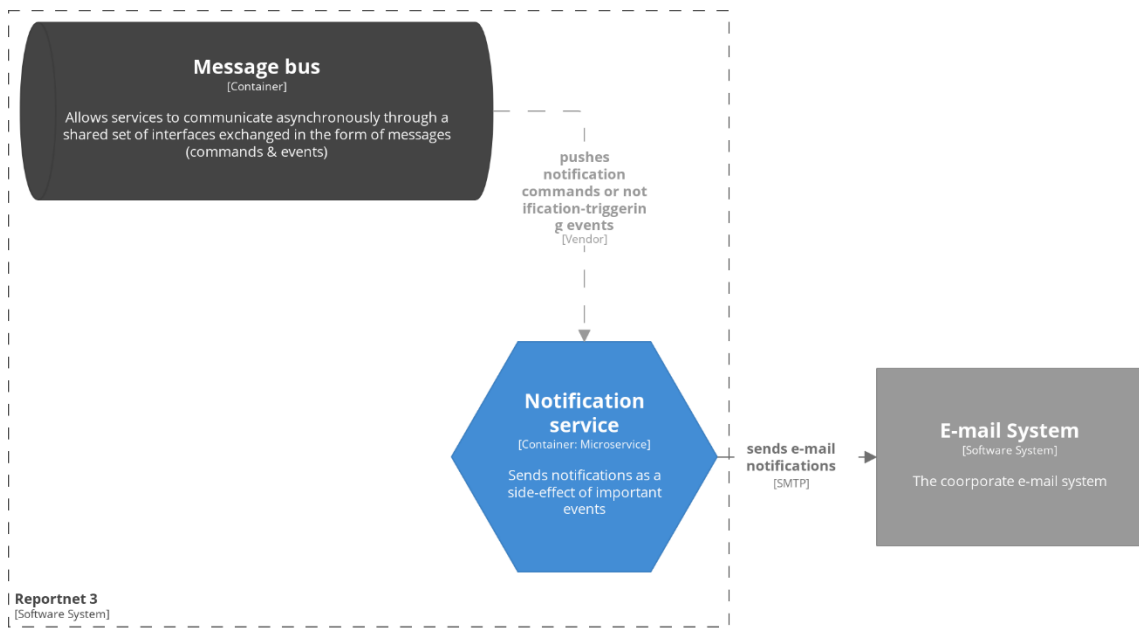
Any dataset, including the special documents dataset type, may allow file uploads through the selection of the special attachment field type. But binary data need not to be stored alongside other record data inside the dataset's relational database. The attachment service provides the appropriate filesystem store to efficiently store binary data of uploaded files.

So whenever the user uploads an attachment in the user interface, she/he indirectly interacts with the attachment service via its exposed REST API and the service assigns to every uploaded document a unique identifier which is what will get stored in the actual dataset record. Retrieval of the actual file's content and relevant metadata will be possible through the attachment service's public facing API.

Message bus integration

In order to properly restrict access on uploaded attachments, the attachment service needs to be aware of the access list for each dataset in the system. To achieve this, the service listens for dataset contributor events from the relevant message bus topic and builds and maintains its own in-memory access list.

2.3.4.10. Notification service



Container diagram for Reportnet 3

Focus: Notification service

Figure 20 – Notification service container diagram

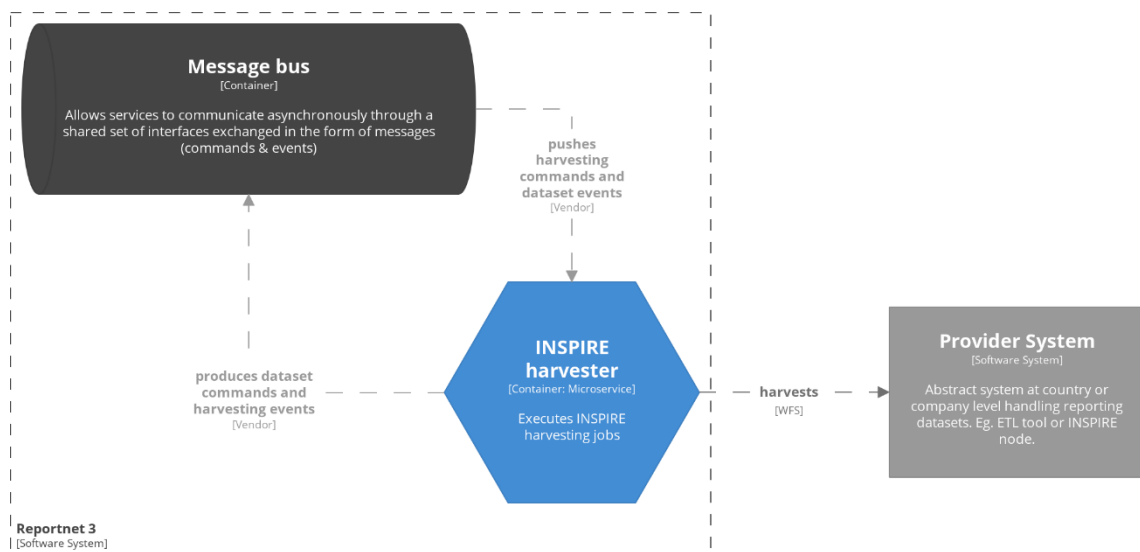
The purpose of the notification service is to capture important events from the message bus, map them to notifications and ultimately send them via an email to the appropriate recipients. Examples of possible notification triggering events include:

- Creation of a new data flow or data collection.
- Sharing a data flow or dataset to a new user.
- Addition of a comment containing a user mention.
- Manual rejection of a delivered dataset.

Apart from such notification-triggering events, any other microservice in the system might issue an ad hoc notification command, which the notification service will asynchronously process by sending an email to the appropriate recipient(s). For instance, such notifications may be published by the data flow service when a data flow deadline is approaching or by the commenting service when a new comment is issued on a topic watched by some users.

The notification service integrates with EEA's e-mail system (SMTP relay) for sending email notifications.

2.3.4.11. INSPIRE harvester



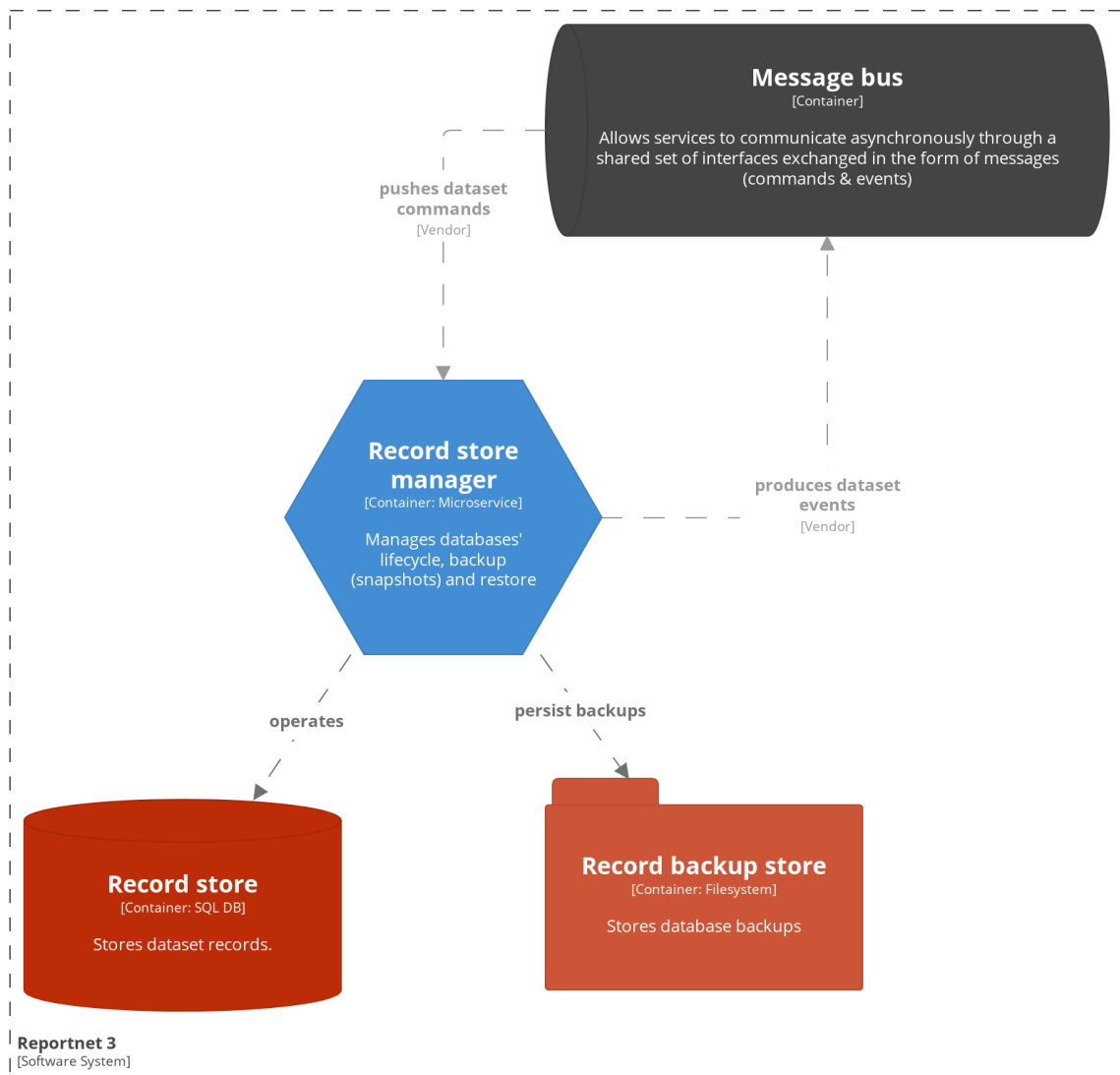
Container diagram for Reportnet 3
Focus: INSPIRE harvester

Figure 21 – INSPIRE harvester container diagram

The INSPIRE harvester is the microservice responsible for executing all INSPIRE data harvesting jobs which are defined by users in their respective reporting datasets. Harvesting jobs are executed asynchronously by the harvester and, theoretically unlimited harvesting parallelisation can be achieved by scaling up the service's containers. Harvested INSPIRE datasets are first validated by the service through the INSPIRE validator⁴, then transformed to dataset record upsert (update or insert) commands based on the user-defined INSPIRE mappings and the resulting stream of commands is finally pushed to the message bus to be further processed by the dataset service which will eventually import the harvested data to the database.

⁴ <http://inspire-sandbox.jrc.ec.europa.eu/validator/>

2.3.4.12. Record store manager



Container diagram for Reportnet 3

Focus: Record store manager

Figure 22 – Record store manager container diagram

Whenever a new dataset gets created in the system the record store manager is responsible for provisioning a new physical database in the system's record store and notify the dataset service in order to subsequently initialise it with the appropriate schema. Apart from creating new physical databases, the record store manager is also responsible for all low-level database operations such as:

- dropping the database whenever a dataset gets deleted
- performing database backups (snapshots)
- restoring backups

For persisting record backups, the record store manager requires the existence of a (shared) filesystem with adequate space.

2.3.5. Integration and Interoperability

As part of the application architecture of Reportnet 3.0, we have already described a system which offers an extensive set of RESTful endpoints for all its managed resources, i.e. submission agreements, data flows and datasets. JSON (Javascript Object Notation) shall be the primary data format used by the system's RESTful API, as this is currently the most widely used data format for data interchange on the web. Moreover, full documentation of the API shall be provided by the system which must be readily available for consumption by integrating parties through the usage of open standards, such as OpenAPI⁵.

Any external system may use the provided APIs to integrate with the system and it is expected to have multiple external systems integrating with Reportnet 3.0 for various purposes, including:

- National systems for automating data provisioning in the reporting flows, including up-to-date data.
- Data dissemination platforms, such as the EU Open Data Portal, for discovering and publishing public datasets.

Assuming that a large number of systems will depend on Reportnet's APIs, proper governance should be put in place with regards to the evolution of this API. More specifically, a proper versioning scheme must be followed and it is expected to have periods where two versions (old and new) of the same API will need to co-exist in order to offer a viable upgrade path for third-party systems.

As it has already been mentioned, apart from the aforementioned RESTful APIs, other systems, internal to EEA may integrate with Reportnet 3.0 at a different level, reading directly from the database layer. Existing analytical tools used now at EEA, such as FME and Tableau shall benefit greatly from this kind of integration in terms of efficiency, since it will be trivial for them to connect effortlessly and allow their operators start working on dataset consumption and processing as soon as possible.

⁵ OpenAPI: Standard for documenting REST APIs - <https://www.openapis.org/>

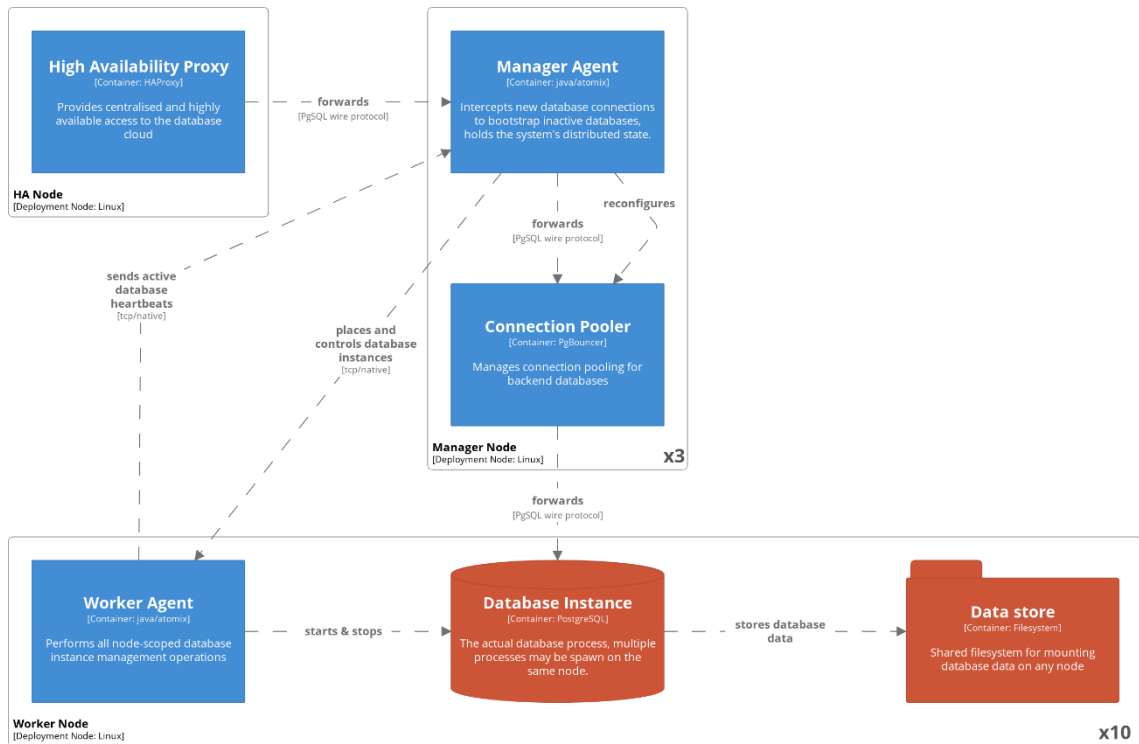
2.4. Technology Architecture

2.4.1. Database infrastructure

For every single dataset created in Reportnet 3.0 a new physical relational database is required to hold the dataset's tables and records. During the lifetime of a data flow many datasets are expected to be created, e.g. for a data flow with 30 countries providing data in it, a minimum of 33 datasets is required (1 design dataset, 30 reporting datasets, 1 data collection dataset and 1 European dataset). In case of multiple data collections performed within the same data flow the number of required datasets grows proportionally and hundreds of datasets may be required for a single data flow. Assuming that Reportnet 3.0 will host hundreds of data flows and that typically data flow cycles are repeated on a yearly basis, the number of physical databases is expected to reach and surpass the 100.000 mark in a few years.

The above numbers alone pose a challenging requirement for the database infrastructure that will support Reportnet 3.0 which additionally needs to provide abstractions similar to that of a database cloud provider such as automated provisioning of databases with centralised access, horizontal scaling, automated node management, fault tolerance etc.. Such abstractions go far beyond the existing clustering capabilities of a relational database such as PostgreSQL. The implementation of such an infrastructure is essentially a separate side-project and a problem, which requires careful design and consideration since it is key to the success of Reportnet 3.0.

If we take a step back and look at the bigger picture though, we can notice that regardless of the total number of databases, in practice most of the databases in Reportnet 3.0 are expected to be inactive (holding rarely accessed, past data) which effectively reduces the estimation for the required number of concurrently running databases to a few hundreds. However, it adds an additional element, that of dynamic resource management, i.e. a pooling mechanism, which must be able to automatically start inactive databases and stop them after a period of inactivity. Although it is not in the scope of the current document to provide a detailed technical solution to this problem, a preliminary design is presented in the following diagram to highlight the overall complexity, but at the same time the technical feasibility of such a database infrastructure.



Deployment diagram

Database Cloud: Provides dynamic provisioning and pooling of PostgreSQL databases for Reportnet 3.0

Figure 23 – Database Cloud deployment diagram

The architecture of the above solution is based on a distributed system, which operates using two separate roles:

- **Manager Nodes:** Control the cluster and hold the system's state, which is distributed across all other manager nodes using an appropriate replication protocol, e.g. Raft. The system's state consists of the database inventory, which is managed through an exposed API and a list of coordinates of the active databases assigned on each worker node. Manager nodes are responsible to intercept new database connections and bootstrap inactive database instances and leverage an existing connection pooling technology to ultimately control the idle timeout for each database instance.
- **Worker Nodes:** They are the nodes hosting the actual database instances and are controlled by the cluster's managers. Workers are also responsible to detect and stop databases without active connections hosted on them and send periodical heartbeats on behalf of their active databases to the leading manager node.

2.4.2. Application infrastructure

Reportnet 3.0 application architecture poses some special requirements concerning the infrastructure that is required to support a microservices architecture.

2.4.2.1. Service deployment – Container orchestrator

Each service in Reportnet 3.0 is independent and self-contained, it might be written in a different language, framework or framework version and it may require to follow its own separate testing, release and deployment cycle. Moreover, services need to easily scale to multiple instances in order to achieve the necessary throughput and availability requirements of the system, or even scaled down during low periods for cutting down the operating cost of the system. At the same

time, the sheer number of services alone poses some interesting challenges pertaining to their deployment, management and monitoring which needs to be as reliable and cost-effective as possible.

Deploying one or even multiple services directly on a physical host or virtual machine (VM) is not considered to be an efficient way of managing microservice deployments let alone rolling out new services with such an infrastructure which usually entails higher turnaround times.

Thus, it is proposed to use a **container orchestrator** as a deployment platform for Reportnet 3.0. Such a platform can provide a higher level of abstraction compared to individual Hosts or VMs, it allows the definition and operation of services, a named set of highly available application processes. Services in such an environment are expected to be containerised (e.g. using Docker containers) and they can be independently deployed, scaled and monitored by the platform. A container orchestrator will take care of scheduling the services to the backing hosts or VMs, monitor them through health checks for liveness and proper operation and it will also automatically reschedule them if a service or its host is failing. Additionally, software defined networking capabilities offered on top of container orchestration platforms may assist in automating network access security-related aspects of service deployments.

Another advantage associated with automated containerised deployments is the fact that it contributes considerably towards achieving the creation of easily reproducible environments. Even though the ultimate goal for the system is to run in a single (production) environment, the requirement for additional similar environments will always be present, at least for the needs of the software development and release process (eg. DEV & TESTING environments). Maintaining, managing and rolling out changes on all these environments usually constitutes a hidden cost and may easily become an overhead for any development/devops team. However, by leveraging a container orchestrator to abstract the infrastructure layer, and investing in full automation of the development, release, configuration and deployment process through CI/CD pipelines, the creation and maintenance of multiple identical environments can be achieved quicker and with less recurring effort.

2.4.2.2. Application metrics and monitoring platform

Each microservice is expected to expose a set of application metrics, i.e. measurements, which provide insight into the service's usage and performance characteristics. Such metrics are valuable when troubleshooting a problem or when we try to measure, understand and analyse the runtime behaviour of the system. To monitor these metrics and be able to setup meaningful dashboards and alerts based on them, a **centralised metrics aggregation platform** is required to collect, visualise and provide appropriate alert management functionalities for the collected measurements. Such a metrics service shall be accessible by the support personnel or any other interested party. Regarding metrics aggregation, two models exist, the push model where each service is responsible for pushing its metrics to the aggregator and the pull model where the aggregation service is responsible for pulling the metrics from each individual service. The message bus can also be utilised as the transportation means of metrics to the aggregation service.

2.4.2.3. Log aggregation platform

Each service is expected to produce its own set of log messages containing errors, warnings, information or debug messages. These logs are critical to the management of the platform, especially for troubleshooting and correcting error situations. Logs can also be used to detect underlying errors early, before they degrade the overall health of the system and start affecting macroscopic metrics. As such, they can also be seen as an important source of system monitoring data. Nevertheless, logs are of low value if they are only available for the narrow

scope of a single service because services collaborate with each other and a log sequence will usually span multiple different services.

In order to be in a position to understand the behaviour of the system and troubleshoot problems efficiently, a **centralised logging platform** must exist which will aggregate logs generated from all sources and allow the support personnel to search and analyse them. Furthermore, in the proposed architecture application logs are not the only logs emitted by the system, the commands and events that flow in the message bus also provide a meaningful trace of the behaviour of the system. Thus, leveraging the existing message bus as the transport vehicle also for application logs will contribute in following a common approach concerning how the system's log sources are being processed and aggregated by the logging service.

2.4.3. Candidate technologies

The following table summarises the candidate (open source) technologies to be considered for use by the various Reportnet 3.0 elements. The proposed technologies presented here are only indicative, since the selection of the final technology stack is expected to be defined at a later phase of the project and will be dependent on the competences of the service provider that will be awarded the implementation contract of Reportnet 3.0.

Element	Technologies	Notes
Message bus	<ul style="list-style-type: none">• Apache Kafka• RabbitMQ	
Record store	<ul style="list-style-type: none">• PostgreSQL	<i>Augmented with a custom-developed clustering/pooling solution, please see section 2.4.1 Database infrastructure.</i>
Container orchestrator	<ul style="list-style-type: none">• Rancher• Docker swarm• Kubernetes	
Microservices framework	<ul style="list-style-type: none">• Spring Boot• Microprofile (e.g. Thorntail, Open Liberty etc.)• Dropwizard	<i>Listed candidates include java-only frameworks. A microservice however may be developed in any programming language.</i>
User interface	<ul style="list-style-type: none">• Angular• React• Vue	
API Gateway	<ul style="list-style-type: none">• NGINX• Kong• Express Gateway• LoopBack	
Application metrics & monitoring platform	<ul style="list-style-type: none">• Prometheus• Telegraf, InfluxDB, Grafana (TIG stack)	
Log aggregation platform	<ul style="list-style-type: none">• Elasticsearch, Logstash, Kibana (ELK stack)• Graylog	

3. REQUIREMENTS MAPPED TO ARCHITECTURE

Id	Description	Mapping
F-001	Reportnet 3.0 will allow for user identity verification beyond Eionet, for example using EU login.	Application Architecture <ul style="list-style-type: none"> • 2.3.4.3 Account service
F-002	Reportnet 3.0 should allow for reporting agreements not requiring a nominated representative, where verified users self-register and report their information, for example industry reporters.	Data Architecture <ul style="list-style-type: none"> • 2.2.3 Data schema • 2.2.5.3 Data Collection and Reporting datasets
F-003	Reporters must be provided with functionality to declare what they will actually report under an agreement and validation will react accordingly, but still ensure technical requirements are met.	Data Architecture <ul style="list-style-type: none"> • 2.2.3 Data schema • 2.2.5.3 Data Collection and Reporting datasets
F-004	Reportnet 3.0 will have the capability for users to subscribe to alert notifications regarding submission agreements and submission status.	Application Architecture <ul style="list-style-type: none"> • 2.3.4.10 Notification service
F-005	Reportnet 3.0 must guide the reporter through the series of steps from initiation to completion in the delivery process. Reportnet 3.0 must manage the reporting sequence within the submission agreement.	Data Architecture <ul style="list-style-type: none"> • 2.2.5 Data flow
F-006	Reportnet 3.0 will support one-off and cyclic reporting periods, continuous reporting, e.g. competent authorities updates and up-to-date reporting, e.g. air quality.	Data Architecture

Id	Description	Mapping
		<ul style="list-style-type: none"> 2.2.5 Data flow
F-007	Reporters would be able to submit comments to the submitted data (e.g. where they cannot report mandatory data or where they need to provide a justification for resubmitting) and the opportunity to provide systematic feedback post-reporting (email questionnaire or similar).	Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface: Commenting 0 Commenting service
F-008	Reportnet 3.0 will provide process and functionality to facilitate integrated manual review of reported data, for example by ETCs.	Data Architecture <ul style="list-style-type: none"> 2.2.5 Data flow
F-009	Reportnet 3.0 will consider the whole dataflow to ensure validations checks are executed only once to be efficient and avoid redundancy.	Application Architecture <ul style="list-style-type: none"> 2.3.4.8 Validation service
F-010	Reportnet 3.0 must support collaboration between actors in the design and delivery processes to achieve common goals, share information and solve business problems more efficiently. Key functionality for a collaborative platform is sharing, rights management and inline communication tools	Data Architecture <ul style="list-style-type: none"> 2.2.5 Data flow Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface: Commenting 0 Commenting service

Id	Description	Mapping
F-011	Reportnet 3.0 must be centralised to enable all processes to be performed in a central point and future users will perform actions on their single computers without interaction with other systems. This requirement also includes the solution should have common web interface.	Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface
F-012	Reportnet 3.0 must allow flexibility for the user to self-decide the format of data delivery (e.g. Web forms, XML, CSV, JSON) including use of web services . The user will be able to gradually build the dataset before formal submission Must also enable snapshots of data as engine for submission mechanism Reportnet 3.0 must be able to handle spatial data from (but not limited to) shapefile, GML and OGC GeoPackage for supporting INSPIRE needs.	Data Architecture <ul style="list-style-type: none"> 2.2.2 Dataset Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface: Dataset Editor 2.3.4.6 Dataset service
F-013	Reportnet 3.0 must include the capability for inline commenting on data structure in dataflow design to facilitate collaboration.	Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface: Commenting 0 Commenting service
F-014	A capability of simple visualisation of data (maps, charts) as standard, and optional for more complex visualisation configuration. Users must be able to interact with the data (sort, filter, group)	Data Architecture <ul style="list-style-type: none"> 2.2.4 Data visualisation Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface: Data visualisations

Id	Description	Mapping
F-015	Reportnet 3.0 must provide immediate feedback to the user on data issues in clear language. Definition of quality rules should be on records, datasets and collections with outcomes stored at the same level. Configuration of quality rules should be widely understood to non-developers. Data visualisation tools should be integral to the QC checks.	Data Architecture <ul style="list-style-type: none"> • 2.2.3.1 Constraints Application Architecture <ul style="list-style-type: none"> • 2.3.4.1 User Interface: Status dashboards • 2.3.4.8 Validation service
F-016	Reportnet 3.0 must enable versioning and maintenance of codelists	Data Architecture <ul style="list-style-type: none"> • 2.2.3 Data schema
F-017	Reporting data must be available to public users, respecting confidentiality, the moment those are available in the system.	Application Architecture <ul style="list-style-type: none"> • 2.3.4.1 User Interface: Public views
F-018	Reportnet 3.0 should allow for reporting agreements not requiring a nominated representative, where verified users self-register and report their information, for example industry reporters.	Data Architecture <ul style="list-style-type: none"> • 2.2.5 Data flow
F-019	Reportnet 3.0 must have a test environment for reporters to gain understanding of the data requirements and test against the quality checks.	Data Architecture <ul style="list-style-type: none"> • 2.2.5.2 Design dataset

Id	Description	Mapping
F-020	Reportnet 3.0 must have the capability to create snapshots of reported data	<p>Data Architecture</p> <ul style="list-style-type: none"> • 2.2.2 Dataset • 2.2.5.3 Data Collection and Reporting datasets <p>Application Architecture</p> <ul style="list-style-type: none"> • 2.3.4.12 Record store manager
F-021	Reportnet 3.0 must be able to handle spatial data from (but not limited to) shapefile, GML and OGC GeoPackage for supporting INSPIRE needs.	<p>Data Architecture</p> <ul style="list-style-type: none"> • 2.2.3 Data schema <p>Application Architecture</p> <ul style="list-style-type: none"> • 2.3.4.1 User Interface: Dataset editor
F-022	Reportnet 3.0 must block deliveries when the data does not meet a specified minimum acceptable technical quality	<p>Data Architecture</p> <ul style="list-style-type: none"> • 2.2.3 Data schema • 2.2.5.3 Data Collection and Reporting datasets
F-023	Reportnet 3.0 should be able to generate a standard set of documentation, which describes the data model, QC steps and workflow without significant configuration by the workflow owner. The data model structure needs to be made available (exported) as scripts so MS can easily create the data model on their own systems	<p>Application Architecture</p> <ul style="list-style-type: none"> • 2.3.4.1 User Interface: Dataset editor • 2.3.4.6 Dataset service

Id	Description	Mapping
F-024	<p>"Requirement related to the Reportnet</p> <p>Reportnet must include the means to provide / identify the Inspire dataset(s)/services that contain the correct data related to the reporting obligation in non-ambiguous way. It shall be mandatory for MS to provide this information."</p>	<p>Data Architecture</p> <ul style="list-style-type: none"> 2.2.5.3 Data Collection and Reporting datasets: INSPIRE source
F-025	<p>"Requirement related to the Reportnet</p> <p>If an Inspire dataset contains features related to different reporting obligations, make sure it is possible to download only the part of the dataset related to that reporting."</p>	<p>Data Architecture</p> <ul style="list-style-type: none"> 2.2.5.3 Data Collection and Reporting datasets: INSPIRE source
F-026	<p>"Requirement related to the Reportnet</p> <p>If the Inspire service is indicated in the reporting data flow, testing the availability of that service should be part of the reporting workflow. Secondly, it must be tested if the returned data follows the expected schema"</p>	<p>Application Architecture</p> <ul style="list-style-type: none"> 2.3.4.11 INSPIRE harvester
F-027	<p>"Requirement related to the Reportnet</p> <p>Test of each reported entry can be reached in the Inspire dataset through the given service(s)"</p>	<p>Application Architecture</p> <ul style="list-style-type: none"> 2.3.4.11 INSPIRE harvester
N-001	<p>"Validation error messages must be indited in a human readable format to be easily understood and corrected by the reporters.</p> <p>The user shall also have access to a log of errors to keep track of the correction (possibility to export the log locally)."</p>	<p>Data Architecture</p> <ul style="list-style-type: none"> 2.2.3.1 Constraints <p>Application Architecture</p> <ul style="list-style-type: none"> 2.3.4.1 User Interface: Dataset editor

Id	Description	Mapping
N-002	The defined countries or companies in the reporting obligation should be notified by the system as soon as reporting cycle of a specific year is open and as the deadline is approaching. Notifications should also be sent for updates in the reporting data model even though reporting hasn't started yet.	Application Architecture <ul style="list-style-type: none"> 2.3.4.10 Notification service
N-003	History of the submissions and the included files per reporting obligation should be logged by the system and be displayed to the users.	Data Architecture <ul style="list-style-type: none"> 2.2.5.3 Data Collection and Reporting datasets
N-004	Dashboards could be provided to the users having relevant access to the system allowing for easier monitoring of the reporting status highlighting the pending actions. A capability of simple visualisation of data (maps, charts) as standard, and optional for more complex visualisation configuration will be available. Users must be able to interact with the data (sort, filter, group).	Data Architecture <ul style="list-style-type: none"> 2.2.5.3 Data Collection and Reporting datasets Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface: Status dashboards
N-005	<p>"The new system should follow EU styleguides to be more user-friendly, modern and output oriented. Indicative styleguides found under the following URLs and should be explored upon implementation:</p> <p>https://ec.europa.eu/info/resources-partners/guidelines-websites-under-eceuropaeu_en</p> <p>http://blogs.ec.europa.eu/eu-digital/design-principles_en</p> <p>https://eui.ecdevops.eu/screen/app/prototypes-opsys"</p>	Principles <ul style="list-style-type: none"> 2.1.1 Follow EU design guidelines
N-006	Non-active reporting obligations could be organised to enable the users to search and find active obligations in a quick way. Moreover, active reporting obligations could be organised thematically and by frequency/ deadlines.	Application Architecture

Id	Description	Mapping
		<ul style="list-style-type: none"> 2.3.4.1 User Interface: Legislation management
N-007	A centralised point of storage should exist in the system for the supporting documents of the reporting process (guidelines, specifications, schemas, ...) which should be accessible by everyone.	Data Architecture <ul style="list-style-type: none"> 2.2.5.1 Documents dataset
N-008	<p>"Navigation process could be facilitated for the reporters by having a structured landing page with direct redirections to actions-pending-to-be-done and explanatory messages.</p> <p>The landing page is different from the dashboard and can be different depending on the roles of the users."</p>	Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface
N-009	The system should allow a pre-filling functionality (i.e. with past data) to support reporting process and allow data comparison.	Data Architecture <ul style="list-style-type: none"> 2.2.5.6 Prefilling data
N-010	The system must be available for use 24/7 and it must achieve 99% up time. Maintenance breaks must be scheduled outside working hours and the system shall present appropriate user notifications before becoming unavailable.	Application Architecture <ul style="list-style-type: none"> 2.3.1 Microservices 2.3.2 Asynchronous Messaging 2.3.4.2 API Gateway Technology Architecture <ul style="list-style-type: none"> 2.4.2.1 Service deployment

Id	Description	Mapping
N-011	The system's user facing components must achieve response times with their 98th percentile not exceeding 300ms as measured from the user interface.	Application Architecture <ul style="list-style-type: none"> • 2.3.1 Microservices • 2.3.2 Asynchronous Messaging
N-012	The system must be able to sustain its performance characteristics during usage peaks resulting from external factors such approaching deadlines. For that purpose, it shall be possible to increase the system's load capacity horizontally without disrupting its operation (zero downtime).	Application Architecture <ul style="list-style-type: none"> • 2.3.1 Microservices Technology Architecture <ul style="list-style-type: none"> • 2.4.2.1 Service deployment
N-013	<p>"The system must be developed following the OWASP secure coding practices and the system's security mechanisms shall be verified regularly through a well-established security testing process.</p> <p>Also, Reportnet 3.0 will comply to the relevant requirements for the COMMISSION DECISION 2017/46 on the security of communication and information systems in the European Commission</p> <p>The security plan shall be documented and accessible to the users."</p>	Principles <ul style="list-style-type: none"> • 2.1.2 Secure by design
N-014	The new system must be developed following a behaviour-driven development approach (BDD) where the expected software behaviours (scenarios) are to be specified in a logical language that everyone can understand and verified during the software delivery process by automated acceptance tests.	Principles <ul style="list-style-type: none"> • 2.1.3 High quality of delivered software software
N-015	The system must continue operating properly in the event of the failure of some (one or more) of its components.	Application Architecture <ul style="list-style-type: none"> • 2.3.1 Microservices

Id	Description	Mapping
		<ul style="list-style-type: none"> • 2.3.2 Asynchronous Messaging <p>Technology Architecture</p> <ul style="list-style-type: none"> • 2.4.2.1 Service deployment
N-016	<p>The system's components must expose measurable behaviour indicators (metrics) which shall be managed (collected, stored, parsed and visualised) centrally and monitored by the technical support personnel.</p>	<p>Application Architecture</p> <ul style="list-style-type: none"> • 2.3.1 Microservices <p>Technology Architecture</p> <ul style="list-style-type: none"> • 2.4.2.2 Application metrics and monitoring
N-017	<p>"Reportnet 3's components must produce all the necessary audit trails of actions performed by its users following uniform logging patterns and all produced logs shall be managed (collected, stored, parsed and visualised) centrally.</p> <p>The logs shall be inline with standards already used at the EEA."</p>	<p>Application Architecture</p> <ul style="list-style-type: none"> • 2.3.1 Microservices <p>Technology Architecture</p> <ul style="list-style-type: none"> • 2.4.2.3 Log aggregation
N-018	<p>The new system must be break into components to be able to manage easier and extend. Moreover a modular architecture will enable the new system to be upgraded (e.g. add or remove any component) with the minimum impact to the rest system.</p>	<p>Application Architecture</p> <ul style="list-style-type: none"> • 2.3.1 Microservices • 2.3.2 Asynchronous Messaging

Id	Description	Mapping
		Technology Architecture <ul style="list-style-type: none"> • 2.4.2.1 Service deployment
G-001	All deliveries in Reportnet 3.0 must be linked to a submission agreements (e.g. obligation or request for national delivery).	Data Architecture <ul style="list-style-type: none"> • 2.2.5 Data flow
G-002	Use existing standards (e.g. INSPIRE) to assure reuse and interoperability of data .	Data Architecture <ul style="list-style-type: none"> • 2.2.3.2 INSPIRE Mappings
G-003	Reportnet 3.0 could be multilingual and even provide a framework for easy translation from the countries side.	Application Architecture <ul style="list-style-type: none"> • 2.3.4.1 User Interface
G-004	Reportnet 3.0 will need to be compliant to the applicable parts of GDPR and also the regulations for EC institutions	<i>Intangible requirement</i>
G-005	Reportnet 3.0 should be designed to handle confidential information appropriately with access managed through rights.	Data Architecture <ul style="list-style-type: none"> • 2.2.2 Dataset • 2.2.5 Data flow Application Architecture

Id	Description	Mapping
		<ul style="list-style-type: none"> 2.3.4.3 Account service
G-006	<p>"Reportnet 3.0 will provide a catalogue API that provides the metadata about the dataflows and the data that is being collected. This metadata should preferably be based on a simple and widely used open standard (as required by the EEA Data policy).</p> <p>A good candidate is the DCAT profile https://ec.europa.eu/isa2/solutions/dcat-application-profile-data-portals-europe_en, which is used for the EU Open Data portal. The data provided for dissemination must respect the confidentiality requirements."</p>	<p>Data Architecture</p> <ul style="list-style-type: none"> 2.2.4 Data visualisation 2.2.5.4 EU dataset
G-007	<p>The system will provide data access through API, for download or integration into downstream dissemination platforms and products.</p>	<p>Application Architecture</p> <ul style="list-style-type: none"> 2.3.5 Integration and Interoperability
T-001	<p>An application must be provided to enable and assist the thematic experts to define, design and maintain the data models (with minimum technical resources). A common vocabulary must be established and reused on data model definition.</p>	<p>Data Architecture</p> <ul style="list-style-type: none"> 2.2.2 Dataset <p>Application Architecture</p> <ul style="list-style-type: none"> 2.3.4.1 User Interface: Dataset editor
T-002	<p>"A universal platform with an easy-to-use interface must be available to all reporting parties for all obligations to streamline the reporting preparation process.</p> <p>The same system (with customization) has to be made available for other economic operators as well."</p>	<p>Application Architecture</p> <ul style="list-style-type: none"> 2.3.4.1 User Interface

Id	Description	Mapping
T-003	The data could be reported in open data formats (e.g. JSON) which can be created from rapid turn-around design tools. For a reason of User-friendliness, other proprietary format shall be considered if possible.	Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface: Dataset editor
T-004	<p>"Reporters should be able to validate the data early enough in the delivery preparation process through visualisation techniques (e.g. maps).</p> <p>The system has to be clear and easy-to-use and available on-line to both MS and other economic operators."</p>	Data Architecture <ul style="list-style-type: none"> 2.2.4 Data visualisation Application Architecture <ul style="list-style-type: none"> 2.3.4.1 User Interface: Data visualisations
T-005	<p>"The system must be able to handle efficiently reported files of at least 3GB.</p> <p>Various options can be considered such as asynchronous process and/or delivering files in multiple time. In the last case, the system has to be able to handle zip files with up to 15k files inside.</p> <p>The file system folder use must be able to handle more than 100k files (as it was the case with the old envelopes)"</p>	Data Architecture <ul style="list-style-type: none"> 2.2.2 Dataset Application Architecture <ul style="list-style-type: none"> 2.3.4.6 Dataset service
T-006	The system must be able to support data load coming from real time data reporting	Data Architecture <ul style="list-style-type: none"> 2.2.5.7 Up-to-date data
T-007	The system must provide a set of well-defined, documented and versioned public Restful APIs allowing third-party systems to easily integrate with it (read/write). Technological Standards shall be use to ensure that.	Application Architecture

Id	Description	Mapping
		<ul style="list-style-type: none"> • 2.3.4.3 Account service • 2.3.4.4 Legislation service • 2.3.4.5 Data flow service • 2.3.4.6 Dataset service • 0 • Commenting service
T-008	<p>"Relational database must be used in the new solution for the following major reasons:</p> <ol style="list-style-type: none"> 1. Ability to have data integrity checks; 2. Ability to relate stored data; 3. Handling of a lot of complicated querying, database transactions and routine analysis of data." 	<p>Application Architecture</p> <ul style="list-style-type: none"> • 2.3.3 Record Store <p>Technology Architecture</p> <ul style="list-style-type: none"> • 2.4.1 Database infrastructure
T-009	<p>The system should support the duplication of existing dataflows or system capabilities to be used as a new starting point for the design of new dataflows.</p>	<p>Data Architecture</p> <ul style="list-style-type: none"> • 2.2.5.5 Repeating data flows
T-012	<p>"Requirement related to the Reportnet Reportnet workflows should continue to accommodate manual uploading of datasets where services are not available."</p>	<p>Data Architecture</p> <ul style="list-style-type: none"> • 2.2.2 Dataset <p>Application Architecture</p> <ul style="list-style-type: none"> • 2.3.4.1 User Interface: Dataset editor

Id	Description	Mapping
		<ul style="list-style-type: none"><li data-bbox="1509 363 1805 389">• 2.3.4.6 Dataset service

4. APPENDIX 1: FULL DOMAIN MODEL

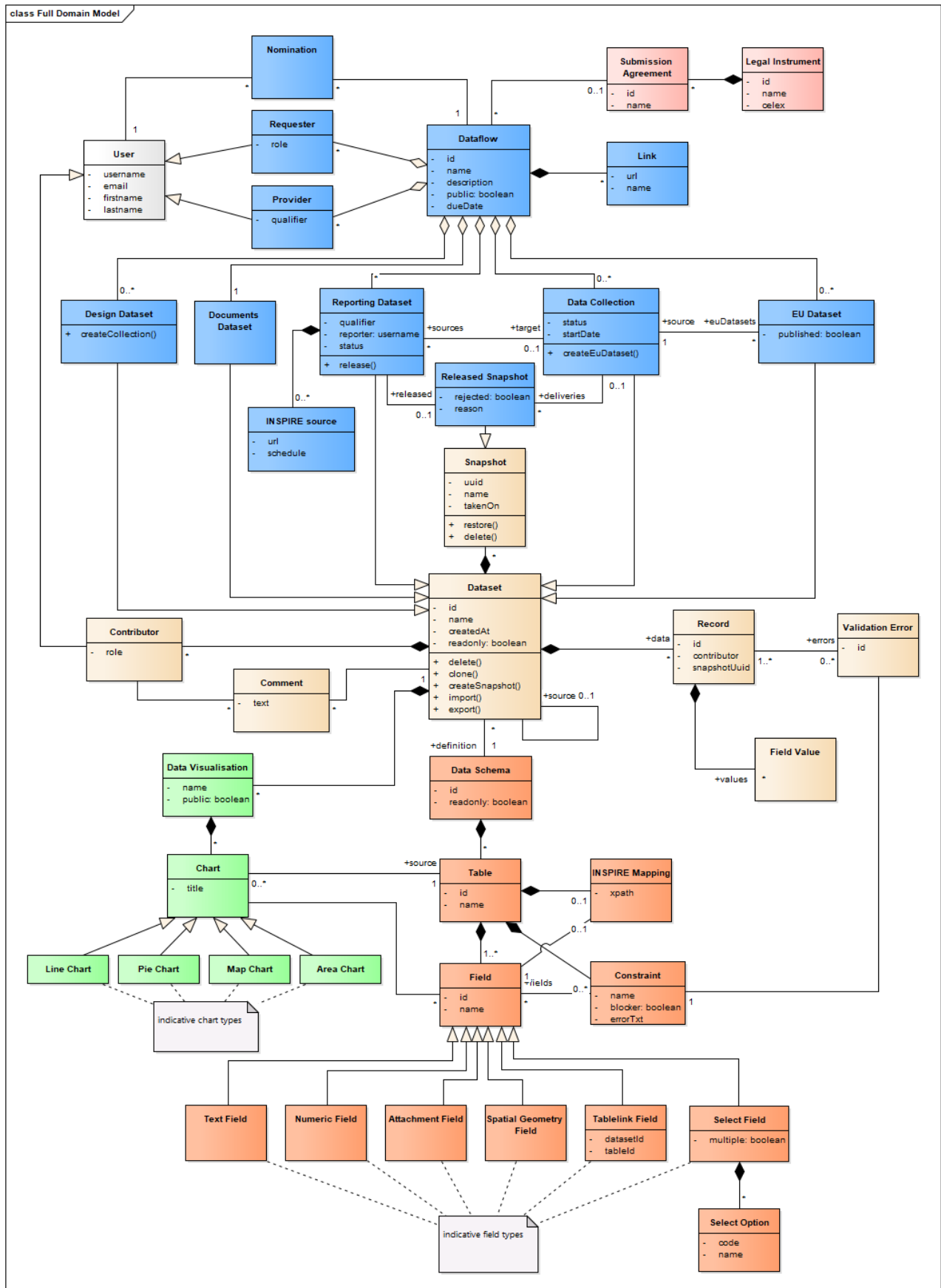






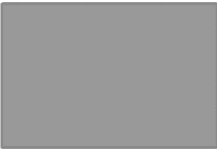



Figure 24 – Full domain model diagram




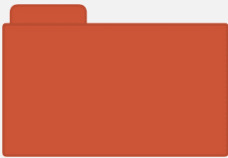
5. APPENDIX 2: C4 DIAGRAM KEY/LEGEND

5.1. Persons





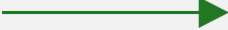

	Public Person with public, unauthenticated access to Reportnet 3.0.
	Data Requester Person requesting data to be reported and participating in the definition of a data flow.
	Data Provider Person providing data to a data flow, acting on behalf of either a country or a company.

5.2. Boxes (systems & containers)

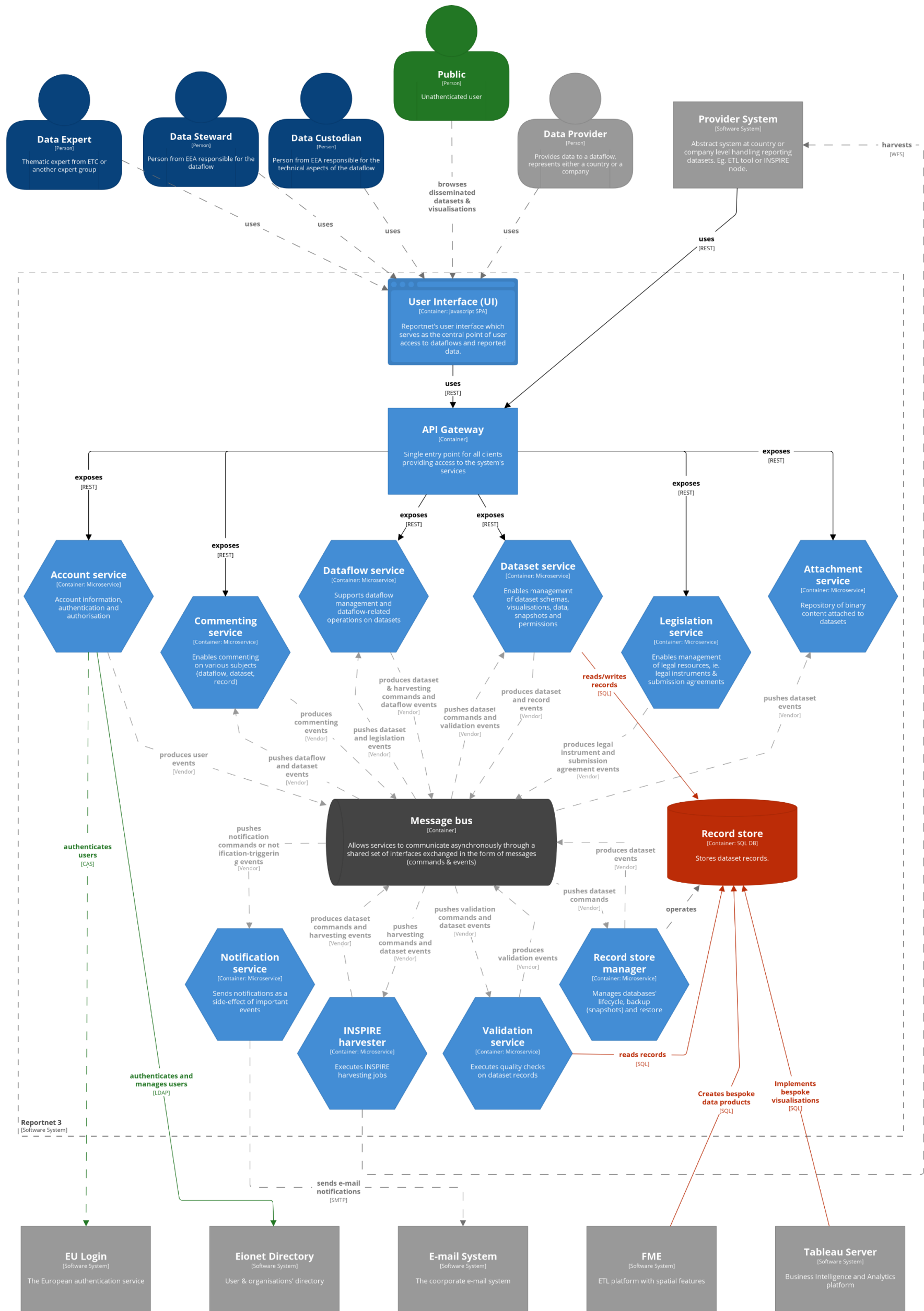
	Context Bounds Elements contained inside the dashed bounding box belong to the same context, which, depending on the diagram, may describe a container, a software system or the whole EEA enterprise.
	External Software System Software system external to Reportnet with which Reportnet interacts.
	Message Bus Allows Reportnet's containers (aka. services) to communicate asynchronously.
	Record Store Relational (SQL) database holding dataset records.
	Container A standard container. A container represents something that hosts code or data. A container is something that needs to be running in order for the overall software system to work.

	<p>Microservice</p> <p>A special type of container describing a small, loosely coupled, autonomous service.</p>
	<p>Browser-based</p> <p>A special type of container describing a process running on a web browser, typically a javascript application.</p>
	<p>Persistent store</p> <p>Special type of container describing a data store, either SQL or NoSQL.</p>
	<p>Filesystem</p> <p>Special type of container describing a filesystem-based store.</p>

5.3. Lines (relationships & protocols)

	<p>Relationship (generic)</p> <p>Relationship indicating usage or dependency between two elements. The arrow indicates the target which is being used or is depended on.</p>
	<p>Vendor</p> <p>Communication achieved through a non-standard, vendor-specific protocol which depends on the final technology choice.</p>
	<p>REST</p> <p>Communication achieved through a Restful, HTTP protocol.</p>
	<p>SQL</p> <p>Communication achieved through SQL queries (on top of a vendor-specific wire protocol).</p>
	<p>LDAP</p> <p>Communication achieved through the LDAP protocol.</p>
	<p>CAS</p> <p>Communication achieved through the CAS single sign-on protocol.</p>

6. APPENDIX 3: MAIN CONTAINER DIAGRAM



Container diagram for Reportnet 3
Main containers

Figure 25 – Main container diagram

7. APPENDIX 4: PRELIMINARY GAP ANALYSIS

The main objective of the main body of this document was to present a fresh architectural vision for the new Reportnet 3.0 platform, without being restricted by constraints associated with the existing version of Reportnet and its current components. Naturally, in the next phase of the project, it is expected to define the appropriate strategy for transitioning from the baseline (Reportnet 2.0) to the target architecture (Reportnet 3.0), along with a plan containing any possible transition architectures.

Nevertheless, a preliminary gap analysis matrix has already been prepared and is presented at the end of this section. From this matrix, the following conclusions can be drawn:

1. Most of the components of the new system will need to be developed as new components.
2. With regards to components identified as potential matches (i.e. ROD and UNS) further analysis and assessment needs to be carried out during the next phase of the project in order to decide on the appropriate strategy.
E.g. concerning the Reporting Obligations Database (ROD), this component could be initially shared between Reportnet 2.0 and Reportnet 3.0. A future scenario might be to evolve it into a stand-alone component accessed via services. Stand-alone because it has its own governance and contains much more than the reporting catalogue which is handled by EEA.
3. Components marked as eliminated won't necessarily have to be decommissioned, since:
 - Obligations are expected to be gradually transferred from Reportnet 2.0 to Reportnet 3.0, so the old system will still have to remain operational for the transition period.
 - Some of the old components, such as CDR, may need to remain operational even beyond the final transition, with restricted (read-only) functionality for archival purposes.
4. Part of the development effort for the new Reportnet 3.0 platform will be spent for the development of the necessary migration tools which will transfer the necessary information from Reportnet 2.0 to Reportnet 3.0. E.g. Data Dictionary (DD) contains a lot of information on reporting structures and there has already been a huge investment in it.

Reportnet 3 Reportnet 2	User Interface	API Gateway	Account service	Legislation service	Data flow service	Dataset service	Commenting service	Validation service	Attachment service	Notification service	INSPIRE harvester	Record store manager	Eliminated Services
Reporting Obligations Database (ROD)				Potential match									
Data Dictionary (DD)													Replaced by schema management capabilities of Dataset service and relevant UI components
Central Data Repository (CDR)													Replaced by Data flow service and relevant UI components
Webforms													Replaced by Dataset service and relevant Dataset editor UI component
Conversion and Quality Assessment Service (XMLCONV)													Replaced by Dataset service (constraint definitions) and Validation service (constraint implementation)
Content Registry (CR)													Replaced by Record store and other services' internal stores
Unified Notification System (UNS)										Potential match			
Gap	To be developed	To be developed	To be developed	Option 1: Make necessary adaptations to ROD Option 2: To be developed as new module	To be developed	To be developed	To be developed	To be developed	To be developed	To be developed	Option 1: Make necessary adaptations to UNS Option 2: To be developed as new module	To be developed	To be developed