

# D2.6 Architecture design and technical specifications document I

Authors: **Benjamin Heitmann**

May 2021



This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 871481.



## TRUSTS Trusted Secure Data Sharing Space

### D2.6 Architecture design and technical specifications document I

#### Document Summary Information

Grant Agreement No	871481	Acronym	TRUSTS
Full Title	TRUSTS Trusted Secure Data Sharing Space		
Start Date	01/01/2020	Duration	36 months
Project URL	<a href="https://trusts-data.eu/">https://trusts-data.eu/</a>		
Deliverable	D2.6 Architecture design and technical specifications document I		
Work Package	WP2 - Requirements Elicitation & Specification		
Contractual due date	31/05/2021	Actual submission date	31/05/2021
Nature	Report	Dissemination Level	Public
Lead Beneficiary	Fraunhofer (FhG)		
Responsible Author	Benjamin Heitmann		
Contributions from	FhG, SWC, EMC, G1, FNET, EBOS, LST, REL, FORTH, KNOW, RSA		

## Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete	Changes	Contributor(s)
v0.1	01/10/2020	5	Initial Deliverable Structure	Benjamin Heitmann (FhG)
v0.2	02/11/2020	20	Collection of requirements for high-level design principles of DMA, IDS and GAIA-X	Benjamin Heitmann (FhG), Stefan Gindl (RSA), Victor Mireles-Chavez (SWC), Christoph Lange-Bever (FhG)
v0.3	18/12/2020	50	Collection of architecture requirements from the WP3 tasks	Benjamin Heitmann (FhG), Steffen Biehs (FhG), Stefan Gindl (RSA), Victor Mireles-Chavez (SWC), Nikos Fourlataras (REL), Dominik Kowald (KNOW), Ohad Arnon (EMC), Andreas Trügler (KNOW)
v0.4	26/03/2021	75	Definition of components for the architecture	Everybody from v0.3, Gianna Avgousti (EBOS), Xavi Olivares (LST)
v0.5	30/04/2021	95	Overview of architecture, additional explanations, design considerations.	Benjamin Heitmann (FhG), Stefan Gindl (RSA), Victor Mireles-Chavez (SWC)
v0.6	05/05/2021	99	Additional explanations for architecture requirements.	Everybody from v0.3
v1.0	10/05/2021	100	Deliverable ready for peer review	Benjamin Heitmann (FhG)
v1.1	19/05/2021	100	Peer review	Ioannis Markopoulos (FNET), Bert Utermark (G1), Victor Mireles-Chavez (SWC), Robert David (SWC)
v1.2	27/05/2021	100	Revisions according to peer review	Benjamin Heitmann (FhG), Mehdi Akbari Gurabi (FhG)
v1.3	31/05/2021	100	Final version	Benjamin Heitmann (FhG)

## Disclaimer

The content of the publication herein is the sole responsibility of the publishers and it does not necessarily represent the views expressed by the European Commission or its services.

While the information contained in the documents is believed to be accurate, the authors(s) or any other participant in the TRUSTS consortium make no warranty of any kind with regard to this material including, but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the TRUSTS Consortium nor any of its members, their officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the TRUSTS Consortium nor any of its members, their officers, employees or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

## Copyright message

© TRUSTS, 2020-2022. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the

work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.

## Table of Contents

Executive Summary .....	9
1 Introduction.....	10
1.1 Mapping Projects' Outputs .....	11
1.2 Deliverable Overview and Report Structure .....	12
2 Technical Requirements for the Architecture .....	13
2.1 Architectural Requirements .....	13
2.1.1 Architecture requirements from alignment with Data Market Austria (DMA) components .....	14
2.1.2 Architecture requirements from alignment with International Data Spaces components .....	15
2.1.3 Architecture requirements from future alignment with Gaia-X .....	17
2.1.4 Architecture requirements related to smart contracts .....	19
2.1.5 Architecture requirements related to the interoperability of data marketplaces .....	20
2.1.6 Architecture requirements related to data governance.....	21
2.1.7 Architecture requirements related to platform development and integration .....	24
2.1.8 Architecture requirements related to brokerage and profiles for users and corporates .....	26
2.1.9 Architecture requirements related to privacy enhancing technologies .....	27
2.1.10 Architecture requirements related to anonymization and de-anonymization .....	30
2.1.11 Architecture requirements following from the usage of Comprehensive Knowledge Archive Network (CKAN) software .....	32
2.2 Summary of Architecture Requirements .....	33
3 Technical architecture of the TRUSTS platform .....	37
3.1 Overview of the technical architecture.....	37
3.1.1 Types of nodes.....	39
3.1.2 Data sets / services / applications .....	40
3.1.3 Connections within the TRUSTS architecture .....	41
3.1.4 Interoperability with External Data Markets and EOSC Initiatives .....	42
3.2 Components of the architecture.....	43
3.2.1 Summary of connections between functional requirements and components .....	55
3.2.2 Summary of connections between architecture requirements and components.....	58
4 Design considerations for the architecture of the TRUSTS platform .....	60
4.1 Functional requirements which are addressed by the architecture as a whole .....	60
4.1.1 Functional requirements coming from the preparation for the use case trials .....	61
4.1.2 Handling of portable applications on the TRUSTS Platform .....	62
4.1.3 Handling of services on the TRUSTS platform .....	63
4.2 The role of the components in enabling trust between participants .....	64
4.3 Planning for the evolution of the architecture with agile methods.....	65
5 Conclusions and Next Actions .....	67
6 References.....	68
7 Annex: Smart Contract FR revision.....	69

## List of Figures

Figure 1: GAIA-X high-level architecture [3] .....	18
Figure 2: Illustration for AR 4.1: Computation using distributed and controlled execution environments .....	28
Figure 3: Illustration for AR4.2: Machine Learning using distributed and privacy preserving technologies .....	28
Figure 4: Illustration for AR4.3: Option of execution of distributed and privacy preserving technologies on servers provided by the TRUSTS operator .....	29
Figure 5: Illustration for AR4.4: Option of execution of distributed and privacy preserving technologies on servers provided by a TRUSTS participant themselves .....	29
Figure 6: Architecture requirements related to anonymisation and de-anonymisation .....	31
Figure 7: Diagram of the technical architecture .....	38
Figure 8: Overview of networks involved in the TRUSTS platform .....	39

**List of Tables**

Table 1: Adherence to TRUSTS GA Deliverable & Tasks Descriptions .....11

Table 2: Summary of Architecture Requirements .....33

Table 3: Summary of connections between functional requirements and components .....55

Table 4: Summary of connections between architecture requirements and components.....58

## Glossary of terms and abbreviations used

Abbreviation / Term	Description
GA	Grant Agreement
HTTP	Hyper Text Transfer Protocol
FR	Functional Requirement
AR	Architectural Requirement
ARS	Architectural Requirement Summary
ID	Identifier
IDS	International Data Spaces, see [2]
DMA	Data Market Austria, see [1]
DAPS	Dynamic Attribute Provisioning System
ACME	Automated Certificate Management Environment
UC	Use Case
CKAN	Comprehensive Knowledge Archive Network
API(s)	Application Programming Interface
MVP	Minimum Viable Product, see [4,5]
PII	Personally Identifiable Information
DM	Data Marketplace



## Executive Summary

This deliverable reports the status of the architecture design and the collection of technical specifications. The technical architecture of the TRUSTS platform is based on three pillars: (1) the vision for the architecture as a consensus of the technical experts in the project; (2) the functional requirements collected from experts inside and outside of the project; and (3) the architecture requirements collected from the project participants which are working on tasks related to the implementation of the platform.

The TRUSTS platform builds on the experiences and best practices from two previous initiatives for supporting data markets. These are the Data Market Austria (DMA) [1] and the International Data Spaces (IDS) [2]. Both initiatives have strongly influenced the architecture design of the TRUSTS platform. On a conceptual level, the roles of participants and the structuring of the federated architecture into several nodes are based on a hybrid of the related concepts from both DMA and IDS. On a technical level, the architecture design prescribes the reuse of infrastructure from both DMA and IDS for the implementation of the platform towards a true hybridized implementation. Taken together, the hybrid architecture will result in a platform implementation which aims to combine the best of both previous initiatives while mitigating strategically important weaknesses.

Towards specifying the architecture design and the technical specifications in this deliverable, the following steps are described in this deliverable: First, the architecture requirements are collected from the project participants which are working on tasks related to the implementation of the platform. Then the architectural requirements and functional requirements are summarized. Then software components for the architecture are specified in order to address the collected functional and architectural requirements. Taken together, these components form the technical architecture of the TRUSTS platform. In addition, we describe the design considerations of the architecture to document the aspects of the architecture which are represented in the interplay of multiple components, instead of being implemented in a single component.

The resulting technical architecture follows all the design principles of the DMA, IDS and GAIA-X, which we have listed as architecture requirements in this deliverable. Of specific strategic significance are the following high-level characteristics of the architecture:

- Security- and privacy-by-design, in order to provide a trusted and secure platform for data markets which empowers consumers with measures to protect their privacy, while simultaneously enabling the emerging data economy.
- Innovative beyond the state-of-the-art, as not just data sets can be traded, but in addition access to services and applications can be monetized while maintaining security and privacy of all involved participants.
- Enabling federation, distribution and decentralization through an architecture which emphasizes distributed operation of infrastructure components, decentralized grouping of participants into federations, and communication between decentralized instances of security related infrastructure.
- Scalability and extensibility, by providing technical interfaces to allow performance critical infrastructure to be replicated horizontally, and to allow participants to extend the platform with their own components, services and applications.
- Future proof, as the technical architecture already in principle is compatible with all high-level architecture principles of GAIA-X. GAIA-X [3] is a project for the development of an efficient and competitive, secure and trustworthy federation of data infrastructure and service providers for Europe.

Taken together, the strong foundation of the architecture of the TRUSTS platform which is represented by the IDS and DMA initiatives, and the strategic significance of the high-level characteristics of the platform, will enable the TRUSTS platform to support new forms of innovation and the development of new business models. To exploit the technical architecture, the experts for business models and innovation in the project will collect business requirements, which will shape the second version of this deliverable.

# 1 Introduction

The architecture design of the TRUSTS platform represents the blueprint for the technical results of the TRUSTS project. As such, it also represents the foundation for instances of the TRUSTS platform which will be provided by one or more TRUSTS operators after the duration of the project. The technical specifications provide the details which are required by technical experts in order to instantiate the platform infrastructure and built on top of it or to extend it with their own components, services and applications.

The work in this deliverable is the result of a collaborative process between all technical experts in the project. The architecture represents the conceptual foundation for the implementation of the TRUSTS platform, and therefore reaching a consensus on the architecture enables all project partners with a technical view to agree on the most important abstract decisions, before realizing them in their implementation. In addition, the architecture also allows the project partners with a non-technical view to contribute with cross-cutting requirements of strategic importance, such as having future proof characteristics.

This deliverable summarizes the efforts accomplished so far and outlines insights gained throughout the process as well as conclusions from these insights. Of particular importance are the documented decisions for challenges of strategic importance, which have an impact on the project beyond the purely technical aspects of the platform:

- How can the technical architecture be **designed as an extension of both the Data Market Austria (DMA) [1] and the International Data Spaces (IDS) [2]** initiatives? This is required in order to enable the maximum uptake of the existing experiences and best practices from these two initiatives, which represent the state-of-the-art for providing support for the development of data markets at the start of the project.
- In addition, can the technical architecture be designed to be **future proof with regards to the emerging data economy in Europe**, by being compatible with the GAIA-X [3] initiative? We expect GAIA-X to set important impulses for the data economy in Europa by e.g., communicating with important groups of stakeholders to set the agenda and by setting standards for technical and organizational issues, such as certification.
- The technical specifications describe the details for instantiating the technical infrastructure of the TRUSTS platform. **How can infrastructure components from both DMA and IDS be used together?** Are there parts of the platform, such as communication between participants, for which only the approach of one legacy initiative has to be selected? Are there areas for which components can be freely mixed and matched?
- The TRUSTS platform aims to **enable trust between the participants of a data marketplace**, which might not have done any form of commerce before interacting on the platform. Which role is the technical infrastructure playing for this? Given the fact that TRUSTS will employ novel approaches to enable this, how can this be anchored in the design of the architecture?
- How can the design of the architecture **enable new forms of assets to be traded and monetized** in a data marketplace? Is there a way to go beyond trading of data sets, and enable for instance the monetization of access to services and apps as part of a data marketplace?
- Given the importance of the technical architecture for the implementation of the TRUSTS platform, which strategy should be used to enable **the evolution of the architecture** during the project duration?

## 1.1 Mapping Projects' Outputs

The purpose of this section is to map TRUSTS Grant Agreement commitments, both within the formal Deliverable and Task description, against the project's respective outputs and work performed.

Table 1: Adherence to TRUSTS GA Deliverable & Tasks Descriptions

TRUSTS Task		Respective Document Chapters(s)	Justification
T2.4 Architecture design and technical specifications	<p><i>Based on the market analysis and requirements elicitation outcomes that are performed in T2.1 and 2.2, as well as the legal and ethical frameworks and requirements generated in T6.2, this task deals with the specification of an architectural design of the TRUSTS platform. While existing specifications such as the Reference Architecture Model (RAM) of the Industrial Data Space (published by the IDSA, co-edited with FhG) and design documents of the Data Market Austria, will serve as a basis, the additional contributions of this Tasks are to (i) align the collected requirements with these specifications, (ii) identify areas that need to be improved and adapted, and (iii) develop and suggest concrete requests for changes and adaptations of the RAM. The Task involves IDSA as well as key contributors to the DMA platform as the main stakeholders of the specifications, leveraging their established processes (e.g., working groups, technical advisory board meetings) to make sure that the proposed changes are incorporated efficiently with wide support from the industry. The task not only focuses on the architectural level but also on making decisions about technologies and methods to actually implement the architecture. This encompasses the design of data structures, message formats, APIs and protocols, but also the definition of procedures, such as the deployment of new component instances and their integration into the running TRUSTS platform.</i></p>	Section 2	Requirements for the architecture are derived from: (1) IDS RAM and DMA as the foundation of the TRUSTS platform; (2) all technical experts in the project, who are working on tasks related to implementing the platform; (3) future proof compatibility with GAIA-X.
		Section 3	The technical architecture and its components are described. This includes the specification of technical details for the interplay of the components.
		Section 4	Any additional design considerations and decisions for the technical architecture are described.
TRUSTS Deliverable			

***D2.6 Architecture design and technical specifications document I***

*First version of the periodically updated report on the TRUSTS platform specification that is based on the results communicated in D2.1 and D2.2 and D2.3. The document describes the architectural decisions taken and their rationale. Furthermore, D2.6 will in addition report on the application of the testing and benchmarking framework provided by T2.3.*

## **1.2 Deliverable Overview and Report Structure**

This deliverable summarizes the activities in task 2.4. “Architecture design and technical specifications” and is the first of two versions of this deliverable. It documents the blueprint for the technical results of the TRUSTS project. The technical specifications provide the details which are required by technical experts in order to instantiate the platform infrastructure and built on top of it or to extend it with their own components, services and applications.

The deliverable is structured into the following sections:

- **Technical requirements for the architecture:** This section collects the requirements for the architecture from two different groups of stakeholders within the project. The first set of requirements are the functional requirements (FRs). They were collected mainly from the non-technical participants of the project, and are collected in deliverable D2.2 “Industry specific requirements analysis, definition of the vertical E2E data marketplace functionality and use cases definition I”. The second set of requirements are the architectural requirements (ARs). They were collected from the technical participants of the project, and are grouped by the different areas of concern for the TRUSTS architecture.
- **Technical architecture of the TRUSTS platform:** Based on the architectural requirements, we describe a software architecture for the TRUSTS platform. We give an overview of the architecture, then we specify which software components are needed in order to address the collected functional and architectural requirements. We also provide tables which list how the components address the identified requirements for the architecture.
- **Design considerations for the architecture of the TRUSTS platform:** We describe the design considerations of the architecture to document the aspects of the architecture which are represented in the interplay of multiple components, instead of being implemented in a single component.
- **Conclusions and next actions:** We conclude the deliverable by listing the project results which are described in this deliverable, and giving an overview of the next actions towards the second version of this deliverable.

## 2 Technical Requirements for the Architecture

The architecture of the TRUSTS platform has to accommodate the requirements and priorities of many different stakeholders. In order to accomplish this, requirements have been collected from two different groups of stakeholders within the project.

The first set of requirements are the functional requirements (FRs). They were collected mainly from the non-technical participants of the project, and are collected in deliverable D2.2 “Industry specific requirements analysis, definition of the vertical E2E data marketplace functionality and use cases definition I”. While this deliverable will reference the functional requirements from deliverable D2.2, they will not be included in this deliverable. Further, the functional requirements will be updated and revised in the second version of the same deliverable, which is identified as D2.3. These revisions will take into account the feedback and experiences of all project partners in the first half of the TRUSTS project. Towards this goal, several suggestions for revisions of functional requirements are included in the annex of this document.

The second set of requirements are the architectural requirements (ARs). They were collected from the technical participants of the project, and are grouped by the different areas of concern for the TRUSTS architecture. These areas of concern are as follows:

- Architecture requirements from alignment with Data Market Austria (DMA) components
- Architecture requirements from alignment with International Data Spaces (IDS) components
- Architecture requirements from future alignment with Gaia-X
- Architecture requirements related to smart contracts
- Architecture requirements related to interoperability of data marketplaces
- Architecture requirements related to data governance
- Architecture requirements related to platform development and integration
- Architecture requirements related to brokerage and profiles for users and corporates
- Architecture requirements related to privacy enhancing technologies
- Architecture requirements related to anonymization and de-anonymization
- Architecture requirements following from the usage of Comprehensive Knowledge Archive Network (CKAN) software

In this section, we first list the architecture requirements for the different areas of concern for the TRUSTS platform. This is followed by a summary of the architectural requirements.

### 2.1 Architectural Requirements

In addition to the functional requirements (FRs), which were mainly collected by the non-technical participants of the project, the TRUSTS architecture is based on architectural requirements (ARs). They were collected from the technical participants of the project, and are grouped by the different areas of concern with regards to the architecture.

For each area of concern, we list the requirements stemming from that area of concern. For each requirement we provide an ID, a summarizing title, and a description.

### 2.1.1 Architecture requirements from alignment with Data Market Austria (DMA) components

One of the two approaches for supporting data markets on which the TRUSTS platform builds, is the Data Market Austria (DMA) data market [1]. The DMA, which started from a flagship project founded by the Austrian Federal Ministry of Transportation, Innovation and Technology, was envisioned as a single point of entry to a federated network of data and data-service providers. It aimed at providing a user-friendly portal through which individuals interested in consuming datasets and data-services can see a list of available datasets, and then contact the respective provider to gain access to the infrastructure that hosts them. The DMA project finished in August 2019.

The DMA was envisioned as a distributed set of nodes hosting the different assets, and a central node hosting the metadata about the assets and other support services. Each node consists of a set of containers serving a series of core components, which include access control, metadata harvesting, asset ingestion UI and an Ethereum blockchain node. The central node additionally executes several metadata management pipelines, along with a user authentication and access control service. Together, these services and an orchestrated exchange of information (e.g. ip-addresses and public keys for verifying authentication tokens) realize the federation of the DMA.

The DMA model of a distributed set of independent nodes and an additional central node has been inherited in the TRUSTS platform architecture presented in this document. It will be further enriched by the security-enhancing technologies that have been developed by the IDSA, as well as a series of components deployed adhoc for the TRUSTS platform. Three important architectural commonalities exist between the two projects. First, the notion of independent nodes that host the assets offered by the organization. Second, the notion of a centralized metadata catalogue that, along with a set of controlled vocabularies, constitutes a knowledge graph on which applications such as search and recommendation are powered. Third, the idea of a set of components being developed and distributed to different organizations so that each organisation can set up their own participating node by running instances of the provided components.

Likewise, several outstanding differences can be identified between the two, which are introduced by the usage of software components from the International Data Spaces (IDS) [2]. The IDS will be described in the next section. On the one hand, the communication protocol between the different nodes is replaced in TRUSTS by the IDSCpV2 protocol, which incorporates an additional layer of security and trust by the use of cryptographic certificates and third party attestation. On the other, TRUSTS introduces the notion of a portable application, which in turn necessitates a more adaptable routing mechanism within each node. In the DMA, routing is configured in a reverse proxy configuration which only requires alterations when a new core component is installed. This contrasts with the TRUSTS routing mechanism which allows dynamically for services to appear, disappear or change names within a node. Finally, the TRUSTS platform envisions a federated user information system, distributed across all the nodes.

Since some of the DMA components are repurposed in the TRUST platform, along with many of the design principles and user scenarios, the following architecture requirements are inherited by the TRUSTS architecture from the DMA project:

Requirements for reuse of DMA components and concepts	
AR 3.D.1	<p>Ability to operate as a distributed set of nodes.</p> <p>The different providers and consumers of assets must remain capable of operating their own infrastructure in order to maintain data sovereignty. This infrastructure should be connected in a well-defined and easy to set-up manner in order to realize the business models.</p>
AR 3.D.2	<p>Asset metadata distribution and aggregation.</p> <p>Organizations should be able to offer existing assets on the platform. This requires mechanisms to ingest, map and distribute metadata about assets in a way that is searchable and actionable by the other participants of the platform.</p>
AR 3.DMA.3	<p>Components must be easily deployable and connected.</p> <p>The different nodes involved in the platform will each deploy a set of services. These can be developed or packaged by the platform operator, and they should be easily installable by every participating organization.</p>
AR 3.DMA.4	<p>Node infrastructure metadata accessible to all components in a node.</p> <p>All components running in a node must have access to a single and up-to-date source of basic metadata about the node, such as name, identification numbers, as well as metadata about other components it interacts with.</p>
AR 3.DMA.5	<p>Single source of truth for controlled vocabularies.</p> <p>In order to adequately manage metadata coming from different organizations, some of which might have been created with distinct purposes in hand, it is necessary to have a set of controlled vocabularies that are centrally maintained and which can be used in mapping of metadata schemas and items.</p>

### 2.1.2 Architecture requirements from alignment with International Data Spaces components

The second approach for supporting data markets on which the TRUSTS platform builds, is the International Data Spaces (IDS) [2], which provides a set of infrastructure components on which data markets can be built. The IDS is a decentral software architecture for exchanging data in a sovereign, secure and interoperable way. Data owners' sovereignty over their data is achieved by certifying the actors that participate in a data space as well as the technical components they operate to exchange data, and by technically controlling the usage of data on the data consumer's side according to metadata by which the data owner described data usage policies.

The IDS Reference Architecture Model is defined by the IDS Association (IDSA) and its 100+ member organizations. In a minimal IDS, participants exchange data peer-to-peer. They do so by operating a

standardized communication interface called Connector. A meaningful data space that allows for multiple participants that neither know nor trust each other initially and that is open for additional participants to join requires further infrastructure called *essential services*<sup>1</sup>. For convenience, further non-essential services make sense in a data space. The following table lists them all:

Service	What is it?	Essential?
Certification body	Governance body empowered to grant IDSA certification for components and participants	Yes
Certification authority	Authority that is in charge of the certification to make sure that only compliant organizations are granted access to the trusted business ecosystem	Yes
Dynamic provisioning service	Management of certifications and metadata for all components and participants	Yes
Participant information system	Registry of certified participants that is accessible to all participants	Yes
Dynamic trust management	Governance body empowered to enforce basic security rules of IDS as a whole	Yes
IDS (metadata) brokers	IDS connectors will register descriptions of data endpoints with IDS brokers. This allows data consumers to find the data they need.	Yes
App store	Outlets providing data apps that can be deployed in IDS Connectors to execute tasks like transformation, aggregation or analytics on the data. Provided by IDS members, certified under IDS standards.	No
Vocabulary provider	Offer “vocabularies” such as ontologies, reference data models and metadata elements, which can be used to annotate and describe datasets.	No
Clearing houses	These intermediaries will provide clearing and settlement services for financial and data exchange transactions in the IDS.	No

<sup>1</sup> <https://internationaldataspaces.org/adopt/essential-services/>



We imagine the TRUSTS data space to be an IDS-compatible data space with at least the Essential Services. The technical services for dynamic provisioning, participant information and (metadata) brokerage should be put into operation as soon as possible, and the participants required for demonstrating the TRUSTS use case should be equipped with IDS Connectors. At the end of the project, organizations should have been identified that take care of certification and trust management. Based on this, the following architecture requirements are inherited by the TRUSTS architecture from the IDS:

Requirements from future alignment with IDS	
AR 3.1.1	<b>Essential services (technical):</b> Services for dynamic provisioning, participant information and (metadata) brokerage shall be put into operation.
AR 3.1.2	<b>IDS Connectors:</b> All participants required for demonstrating the TRUSTS use cases shall be equipped with IDS Connectors and should be able to expose their data offerings through these connectors' interfaces, including self-describing metadata in terms of the IDS Information Model.
AR 3.1.3	<b>Essential services (operational):</b> Organizations that provide the essential services of certification (i.e., certification body and certification authority) shall be identified. A body in charge of dynamic trust management shall be established.

### 2.1.3 Architecture requirements from future alignment with Gaia-X

GAIA-X [3] is a project for the development of an efficient and competitive, secure and trustworthy federation of data infrastructure and service providers for Europe, which is supported by representatives of business, science and administration from European countries.

GAIA-X follows the principles of openness and transparency of standards, interoperability, federation, i.e., decentral distribution, as well as authenticity and trust (Technical Architecture [3, §1.2]).

These principles are translated into the following technical guidelines (Technical Architecture [3, §1.3]):

- Security-by-design
- Privacy-by-design
- Enabling federation, distribution, and decentralisation
- Usage-friendliness and simplicity
- Machine-processability
- Semantic representation

The GAIA-X ecosystem as a whole is structured into a Data Ecosystem and the Infrastructure Ecosystem (Technical Architecture [3, §1.4]). The Data Ecosystem enables Data Spaces as envisioned by the European Data Strategy, where not only data is exchanged, but also advanced smart services are provided. The Infrastructure Ecosystem comprises building blocks from hardware nodes to application containers, where data is stored and

services are executed, as well as networks, via which data is transmitted. Infrastructure itself may be provided as a service.

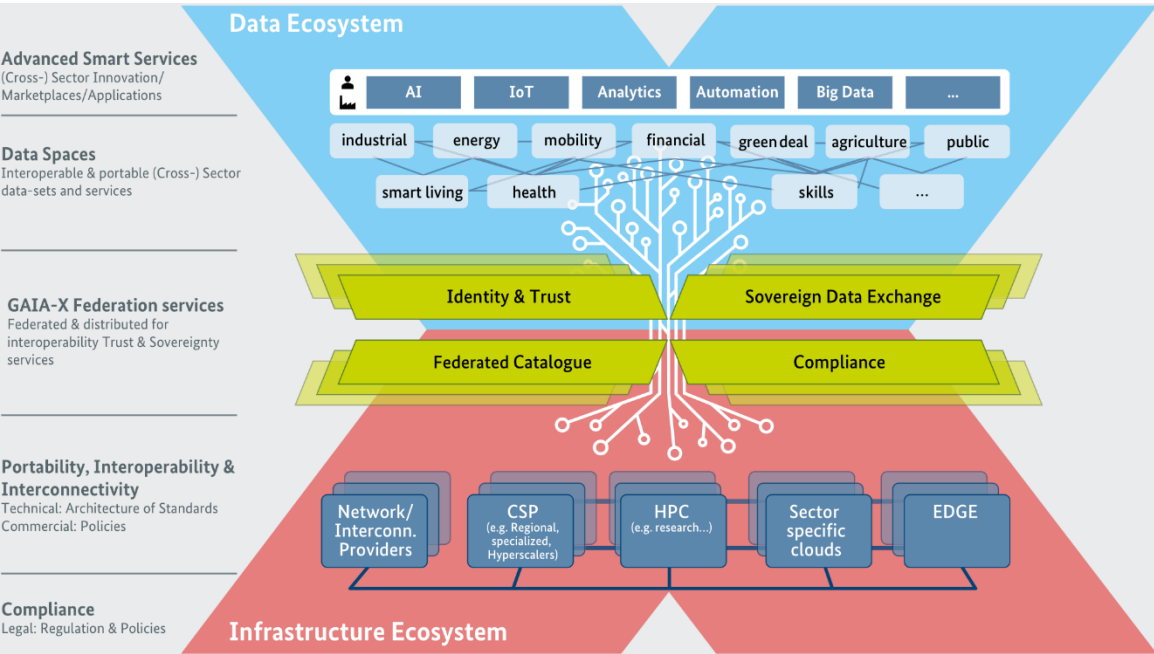


Figure 1: GAIA-X high-level architecture [3]

The Federation Services connect the Data and Infrastructure Ecosystems with concrete functionality that is in line with the architecture principles and technical guidelines. The Federation Services are grouped into the following domains, which are by no means specific to GAIA-X, but generally apply to data and service ecosystems based on cloud technology:

- Identity and Trust mechanisms, comprising federated identity management, trust management, and federated access.
- A Federated Catalogue, comprising offerings of assets (i.e., data and services) that describe themselves in a FAIR (findable, accessible, interoperable, reusable) way.
- Sovereign Data Exchange, ensured by policies, usage control, and security concepts.
- Compliance in a sense of organization and governance (defining the rights and obligations between service providers and consumers, and the process of onboarding participants and their assets into the ecosystem), but also supported technically via certification and continuous monitoring.

In being compatible with the Federation Services, providers and their assets can fulfil the architectural principles as follows:

Requirements from future alignment with Gaia-X	
AR 3.X.1	<b>Security by design:</b> Security considerations are addressed by secure and sovereign data exchange, as well as

	compliance concerns.
AR 3.X.2	<b>Privacy by design:</b> Identity and Trust provide the foundation for privacy. Compliance with privacy requirements is further ensured by the mechanisms for sovereign data exchange.
AR 3.X.3	<b>Enabling federation, distribution, and decentralisation:</b> The Federation Services themselves are designed to operate in a federated, distributed, and decentralised manner. The Federated Catalogue encourages providers to offer services on all layers of abstraction, from infrastructure as a service to data as a service. Their machine-processable and semantic Self-descriptions enable automated deployment.
AR 3.X.4	<b>Usage-friendliness and simplicity:</b> APIs, which can be found via the Federated Catalogue), enable the construction of human user interfaces.
AR 3.X.5	<b>Machine-processability:</b> This is ensured by Self-descriptions of assets in the Federated Catalogue, which comprise usage policies and may point to APIs. APIs further enable automation.
AR 3.X.6	<b>Semantic representation:</b> Self-descriptions in the Federated Catalogue make assets FAIR (Findable, Accessible, Interoperable, Reusable).

#### 2.1.4 Architecture requirements related to smart contracts

Smart contracts are one of the areas of concern for the TRUSTS architecture. The work in relation to this concern is being led by Fraunhofer ISST. The main goal with regards to this concern is to develop the necessary concepts for using smart contracts in the context of the European Data Market delivered by the TRUSTS project. This includes in particular the technical foundations for ensuring the integrity and authenticity of such contracts as well as the analysis of the legal challenges brought about by smart contracts, such as issues of validity, enforceability, and interpretation. Related to this, the technical challenges and the legal issues regarding the fact that smart contracts are written in executable code instead of natural language, will be examined by the partners with legal expertise in the project.

The result of this work is a comprehensive concept document related to all the aspects of the above paragraph. Additionally, a demonstrator is being developed by Dell. The demonstrator consists of a blockchain instance which is using the Hyperledger Fabric technology. Thus it is possible to execute smart contracts in this instance. The demonstrator will also connect to external payment systems. The demonstrator is logged as architecture component C24 - “smart contract executor”.

Based on the project work of the partners with expertise related to smart contracts, the following requirements for the architecture of the TRUSTS platform are specified. In addition, several suggestions for revisions of functional requirements related to smart contracts are included in the annex of this document.

T3.2 Smart Contracts	
AR 3.2.1	<p><b>Smart contracts based on data sharing policies:</b></p> <p>It should be possible to create smart contracts based on policies for data sharing. The policies define how data can be handled by a data consumer. When a data consumer accepts a policy, a contract is created between the data provider and the data consumer based on the accepted policy. From this contract, a transaction will be created by one of the parties of this contract. The transaction will be sent to the blockchain instance. Before this transaction will be stored into a new block, it triggers a smart contract which is related to the contract and proves it.</p>
AR 3.2.2	<p><b>Logging of all transactions:</b></p> <p>Every transaction between two participants should be logged. It should be possible to define which information is logged for which kind of transaction.</p>

### 2.1.5 Architecture requirements related to the interoperability of data marketplaces

The interoperability between TRUSTS and other data marketplaces is one of the concerns of the TRUSTS platform. This includes the definition and implementation of interfaces to ensure interoperability with other industrial data marketplaces as well as with EOSC initiatives. In addition, this is related to analysing and examining existing standards and interfaces with regards to their suitability for interoperability. This will ultimately result in the development of an interoperability solution, which provides the necessary technical functionality to interoperate with a selected set of existing data marketplaces and EOSC initiatives. The interoperability solution will help to include external resources in TRUSTS. The interoperability solution is envisaged as a client-server architecture consisting of a data exchange component residing on the premises of TRUSTS as well as another component residing on the premises of an external data marketplace or EOSC initiative, respectively.

The successful implementation of the interoperability solutions requires a set of components. The architecture is based on a careful evaluation of both existing data marketplaces as well as EOSC initiatives. This involved the analysis of technical and operational features such as the provided resources (e.g. data assets or search functionality) or available APIs. It involves a metadata schema for data assets based on existing schemas such as the IDS Metadata Model, two components establishing metadata and/or data transfer between TRUSTS and external data marketplaces and EOSC initiatives, a registry of data marketplaces containing up-to-date information about external data marketplaces/EOSC initiatives, as well as a graphical user interface for the interoperability solution. The resulting architecture requirements are as follows:

Architecture requirements related to data marketplace interoperability	
AR 3.3.1	<p><b>Metadata schema for data assets:</b></p> <p>A documented metadata schema building on top of existing schemas, such as the IDS Metadata Model for describing resources and datasets, but extended to data assets.</p>
AR 3.3.2	<p><b>Data exchange client component:</b></p> <p>A module that can be integrated into a common data management platform which will enable the exchange of information about data products across a federated data markets network. An external data market should be able to integrate this software library with ease and be able to</p>

	expose information about its data products to TRUSTS as a result. The library should also serve as a basis for connecting TRUSTS with EOSC.
AR 3.3.3	<b>Data exchange TRUSTS component:</b> This component resides on the side of TRUSTS and communicates with the Data Exchange Client Component. It receives the input from the Data Exchange Client Component and converts it into the format required by TRUSTS, subsequently storing the received data in the TRUSTS data storages.
AR 3.3.4	<b>Registry of data markets:</b> This component will be essential for having always up-to-date information about external data markets (metadata schemas, APIs, platforms) currently used in the sector. The information in the registry will assist in identifying the external parties most suitable for the testing/piloting of the TRUSTS developed interoperability solution.
AR 3.3.5	<b>Administrative interface:</b> There should be an <b>administrative interface</b> for the TRUSTS operator that enables the operator to <b>add, update or remove external data markets supporting the developed interoperability solution</b> from the TRUSTS platform. This will be the mechanism by which the TRUSTS operator decides which data markets to connect to. The process can be facilitated by automatically linking data markets listed in the Registry of data markets.

### 2.1.6 Architecture requirements related to data governance

The TRUSTS platform will enable exchange of assets (services, applications, datasets) between participants. These assets, however, will remain in the hands of their respective owners and only be transferred to those who purchase them in accordance to contractual terms. To enable said exchange, the TRUSTS operator will collect metadata about the assets, their provenance, their location and their permitted usage and commercialization options.

Towards this goal, a set of ontologies and controlled vocabularies for expressing such metadata have been identified, as well as schemata and exchange protocols for their exchange and storage. A specification for using these ontologies, vocabularies, schemata and exchange protocols together will be documented in detail in deliverable D3.7. This specification will be utilized by almost all components of the TRUSTS platform in order to retrieve and store metadata about the assets being traded, the organization of the platform nodes, and the organization of components within nodes, among others. The semantics of this metadata will be explicitly described in a combination of machine readable formats (RDF) and human readable documentation, which will in turn guide i) the implementation of software artifacts specialized in metadata processing, as well as ii) the interpretation that other components have of said metadata and how they translate these into actions.

The exchange and use of metadata necessitates a combination of solid schemata, ontologies, controlled vocabularies, and identifier practices, as well as an underlying technical infrastructure for its processing and storage. It is this technical infrastructure which induces a set of architectural requirements described below.

A subset of these requirements aim at solidifying the joint evolution of the architecture and metadata layer that has taken place so far in the project. For example, the metadata definition envisions each asset to be

associated with a specific organization, and provisions have been made to describe from which of the many components that said organization operates an asset is served. This organization of metadata is thus compatible with the notion that each participant in the TRUSTS platform be the operator of a computing environment (AR 3.4.1) in which several software components are executed (AR 3.4.2). In addition, there exist mechanisms for locating an asset among these components given a metadata description (AR 3.4.4, AR 3.4.8). Other requirements aim at ensuring the technical means for the creation, storage and maintenance of metadata. For example, user interfaces should produce metadata whenever an asset is onboarded (AR 3.4.12), and an organization's existing metadata should be transformed to conform to the TRUSTS metadata schema (AR 3.4.9), having always in mind the existing repertoire of, for example, topics and keywords with which to classify an asset (AR 3.4.11).

Architecture requirements related to data governance, metadata, lineage and semantic layers	
AR 3.4.1	<p><b>Every TRUSTS platform instance is a distributed set of nodes:</b></p> <p>TRUSTS should be a distributed set of <b>nodes</b>. Each node can run in its own infrastructure, and its own network. Connection between nodes should be done through the IDSCP, or through other protocols that allow for certified communications and identity verification. Nodes must expose a self-description according to the IDS information model.</p>
AR 3.4.2	<p><b>Each node runs software components:</b></p> <p>Each node runs a series of <b>software components</b>, which can be installed into the node using docker containers, and which can communicate among each other. A software component can be certified by a certification authority.</p>
AR 3.4.3	<p><b>Each node runs an instance of the trusted connector:</b></p> <p>Each node must run a software component that controls communication with other nodes. This component, referred to as <b>Connector</b> can be an instance of, e.g. IDS Trusted Connector or any other connector compliant with the IDSCP and compatible with the IDS Trusted Connector. The Connector should be able to route messages between other nodes and other software components within the node. It must expose a series of ports through which other connectors can send or request data.</p>
AR 3.4.4	<p><b>Each node has an internal directory of running software components:</b></p> <p>Each node must be equipped with a machine readable <b>internal directory</b> of the software components running inside of it, their type, address and, if available, connector port that they respond to. This list must be the source of configuration for internal routing between connector and software components.</p>
AR 3.4.5	<p><b>A node can internally be a distributed node:</b></p> <p>Each TRUSTS node can, itself, enclose a distributed system (e.g. multiprocessing cluster) provided that:</p> <ol style="list-style-type: none"> <li>1. components of this system communicate only among themselves,</li> <li>2. communication to other software or hardware components must be done through the Connector.</li> <li>3. The above rules are enforced both in configuration of the distributed system itself, as</li> </ol>

	<p>well as through network configuration.</p> <p>Nodes which implement these three requirements are called <b>Distributed Nodes</b>.</p>
AR 3.4.6	<p><b>The TRUSTS platform has at least one Central Node:</b></p> <p>At least one of the nodes of TRUSTS, called <b>Central Node</b>, must maintain a centralized metadata store that includes connection details (IP addresses, ports) of every other node. It must receive this information in accordance with the IDS information model, check for its authenticity and keep it up to date. This node must also expose, through a Connector interface, updates to this metadata that can be leveraged by other components, described below. Furthermore, it must provide to other nodes information about the TRUSTS ecosystem that will allow for the different nodes to interoperate, in particular: location, identity and public keys of all nodes registered with it, as well as information (e.g. public keys) that can aid in validating signatures by specific services, such as the Authorization service or the Vocabulary Management service.</p>
AR 3.4.7	<p><b>A node must be able to run apps:</b></p> <p>Nodes must be capable of running arbitrary software components (<b>applications</b>) in a sandboxed environment, that is, having only connections to other components as configured by the node administrator, and to the outside world (including other nodes) through the Connector. For example, it should be possible to install in a node a Mapping Component app that, after configuration, can i) go through a set of metadata files in the node, ii) consult another node for some entity extraction, and iii) transform each metadata file into another format and iv) send this metadata onto the Broker Node.</p>
AR 3.4.8	<p><b>Managing of assets in a node:</b></p> <p>Nodes can run one or more apps called <b>Asset Servers</b> which must i) receive requests for assets (data / service access) coming to the node in the form of an AssetID, a requester ID and an authorization note, ii) check the validity of the authorization, iii) answer the request with the asset. These asset servers must specify the method for accessing their assets (e.g. files, REST endpoints, SCP connections), and make this metadata available through a REST call.</p>
AR 3.4.9	<p><b>Harvesting of metadata in a node:</b></p> <p>Nodes can run one or more apps called <b>Metadata Harvesters</b> which collect metadata about the node's assets, which can be in a variety of storage mechanisms (e.g. files, tables, OAI-PMH endpoints). If necessary, these metadata can be sent to an external node for mapping and then stored in this new mapped form. The metadata can be sent, through the connector, in the form of IDS messages to the central node and must also include information about the method for accessing assets, which can be retrieved from the corresponding asset server.</p>
AR 3.4.10	<p><b>Recommendations in a node:</b></p> <p>Nodes can run an application for <b>Recommendation</b> that will do analysis on the broker nodes' metadata repository. These nodes need to be authorized by each broker node to do this analysis.</p>
AR 3.4.11	<p><b>Managing vocabularies in a TRUSTS ecosystem:</b></p> <p>At least one node in the TRUSTS ecosystem must run application for centralized <b>Vocabulary</b></p>

	<b>Management</b> that is accessible to other nodes and provides i) a SPARQL interface for vocabularies used by the TRUSTS ecosystem, and ii) linked data descriptions of these vocabularies (using content negotiation). Nodes running this service must also run a <b>Metadata Mapping</b> service that, after case-per-case configuration, can map metadata in an existing JSON or XML schema into the IDS information model schema, serialized in RDF/Turtle.
AR 3.4.12	<b>User interfaces in a TRUSTS ecosystem:</b> At least one of the nodes of TRUSTS, called a <b>Portal Node</b> , must exhibit a set of user interfaces for searching the assets stored in a set of Broker nodes, generating configuration files for metadata mapping, as well as buying available assets. These UIs must integrate with an authorization service running also in this node.

### 2.1.7 Architecture requirements related to platform development and integration

In order to develop the TRUSTS platform and integrate existing software components into the platform, the architecture of the TRUSTS platform needs to accommodate the reuse of existing software components. These components are provided by the International Data Spaces (IDS) [2], Data Market Austria (DMA) [3] and Comprehensive Knowledge Archive Network (CKAN) software. In addition, newly developed components provided by the partners of TRUSTS also need to be integrated into the TRUSTS platform. The new platform should be easy deployable to widely used operating systems and cover as many possible potential participants in order to allow them to connect new or existing information systems to the TRUSTS platform. The platform should provide access to the TRUSTS ecosystem in a secure way for both corporate and individual users, as long as they satisfy the requirements for participation.

Towards this goal, an implementation of the TRUSTS platform will be developed and tested which integrates a selection of existing open source components with components newly developed in order to cover the functional requirements of the project. To achieve these goals we have concluded that the platform architecture should be built around flexible and secure components that can be easily deployed. In addition, infrastructure should be offered to provide flexibility with regards to the types of services or protocols used in order to maximise the types of existing solutions which can be offered by participating organisations.

The TRUSTS architecture should be independent from an operating system, easy to deploy, and simple to adapt. In addition, one of the most important considerations is the preservation of security and privacy by the platform. For this reason, the platform should be built as a network of secure and sovereign nodes which can communicate in a peer-to-peer fashion. In addition, each node should have the means to control how any data is used by any of the services and applications which are deployed on that node. This also allows the mitigation of security breaches by excluding potentially compromised nodes. Finally, the TRUSTS platform should enable the deployment of services and portable applications independent of the technology or development environment used to implement the service of applications, as long as a small set of API requirements is fulfilled.



The resulting requirements for the architecture are as follows:

<b>T3.5 Platform Development and Integration</b>	
AR 3.5.1	<p><b>Communication via a connector instance:</b></p> <p>Communication between TRUSTS nodes should follow a standardised and secure protocol. It should be possible to easily add new nodes to an existing ecosystem. For this purpose, a connector component should be used as a universal communication component.</p>
AR 3.5.2	<p><b>Running a connector instance:</b></p> <p>The source code of the connector component should be available for modification, if needed. The implementation should allow adding new nodes to an existing ecosystem. In addition, any required credentials or security certificates for running the connector, should be provided by the TRUSTS operator.</p>
AR 3.5.3	<p><b>Communication between a connector instance and an application:</b></p> <p>The Connector should provide means to communicate with any application inside the same node, using routing of incoming and outgoing flows and this routing should allow for dynamic changes. This communication should accommodate a large set of technologies used to implement an application.</p>
AR 3.5.4	<p><b>Securing a connector instance:</b></p> <p>The Connector should support the (existing) authentication, authorization and security system inside the node.</p>
AR 3.5.5	<p><b>Connecting an application to a connector via an adapter:</b></p> <p>It should be possible to create adapters for applications inside a node, which can not be directly connected to the connector.</p>
AR 3.5.6	<p><b>Running components via docker:</b></p> <p>All infrastructure which is provided by the TRUSTS platform in the form of shared components, should run on Docker-Compose. This should include the connector, adapters and other shared components.</p>
AR 3.5.7	<p><b>Usage control in a TRUSTS node:</b></p> <p>All components running in a TRUSTS node should be connected to the usage control component in order to check if any user initiated action is permissible.</p>
AR 3.5.8	<p><b>Metadata and security in a TRUSTS node:</b></p> <p>All infrastructure components provided by the TRUSTS platform should support the metadata and security infrastructure of TRUSTS.</p>
AR 3.5.9	<p><b>Configurability of TRUSTS nodes:</b></p> <p>All infrastructure components provided by the TRUSTS platform should allow any changes to TRUSTS functionality with minimum coding. These changes should be done easily and using</p>

	configuration files where possible.
--	-------------------------------------

### 2.1.8 Architecture requirements related to brokerage and profiles for users and corporates

Another area of concern for the TRUSTS platform is brokerage and profiles for users and corporates.

Specifically, the main objective related to this is the realization of a mapping between offerings and demands of users, datasets and services. To realize this, a recommender system will be developed with the following three goals: (i) extract, process and store user interactions (e.g., downloads) and metadata of users/corporates, services and datasets, (ii) develop and provide recommendation algorithms to interlink users/corporates, services and datasets in this tripartite recommendation setting, and (iii) collect user feedback (e.g., clicks) to evaluate and tune the recommendation algorithms.

Based on these goals, three requirements for the architecture of the TRUSTS platform were derived. The main requirement for a recommender system to work is data. Thus, the first goal deals with data handling, processing and storing, which requires a mechanism in the TRUSTS platform that informs the recommender system about new potential resources to be recommended (i.e., datasets and services) and also interaction data that the recommender system can use to train its algorithms (AR3.6.1). The second goal deals with the development and integration of recommendation algorithms. Here, the recommender system requires a mechanism in the TRUSTS platform to give context to the recommendation requests (e.g., the current user) and to present recommendation results (AR 3.6.2). Finally, the third goal is related to the evaluation and tuning process of the recommender system. This translates to the requirement of providing users of the TRUSTS platform with functionality for interacting with recommendation results (AR3.6.3). More details to these requirements are given below:

T3.6 User and Corporate Profiles and Brokerage	
AR 3.6.1	<p><b>Notification mechanism to provide data for the recommender system.</b></p> <p>In order to provide the recommender system with data for training its algorithms, the TRUSTS platform should provide a mechanism to transfer data to the recommender system. Therefore, the recommender system will provide REST-based services to add (i) metadata of datasets, (ii) metadata of services, (iii) metadata of users/corporates, and (iv) interactions between those entities (e.g., if a user downloads a dataset). The TRUSTS broker and the TRUSTS platform should use these services in order to notify the recommender system when new entities or interactions come into the platform or when existing entities are changed.</p>
AR 3.6.2	<p><b>User interface component to show recommendations.</b></p> <p>For visualizing recommendation results to users, the TRUSTS platform should provide a user interface component that is capable of showing an ordered list of recommendations. For this purpose, the recommender system will provide REST-based services for six recommendation settings: (i) recommend datasets to users/corporates, (ii) recommend services to users/corporates, (iii) recommend datasets to services, (iv) recommend services to datasets, (v) recommend datasets to datasets, and (vi) recommend services to services. The TRUSTS platform needs to use these services to query recommendations by providing parameters such</p>

	as the current user, the currently viewed dataset or service, one of the six mentioned use cases, the algorithm (e.g., collaborative filtering or content-based filtering) and the number of recommendations to generate (the default value is 10).
AR 3.6.3	<p><b>User interface component to interact with recommendations.</b></p> <p>When recommendations are shown to users, the TRUSTS platform should also allow them to interact with the recommendations, i.e., click on the recommendations to get additional information. Thus, for every recommendation request, the recommender system will generate a unique recommendation id that is provided with the list of recommendations. The TRUSTS platform needs to store this recommendation id and whenever a user interacts with a recommended entity informs the recommender system about this interaction, which is interpreted as feedback to the recommendation. With this, the recommender system is able to evaluate the quality of the recommendations and adapt the algorithms if necessary. Furthermore, this allows us to conduct A/B tests and compare the quality of two types of algorithms (e.g., collaborative filtering and content-based filtering).</p>

### 2.1.9 Architecture requirements related to privacy enhancing technologies

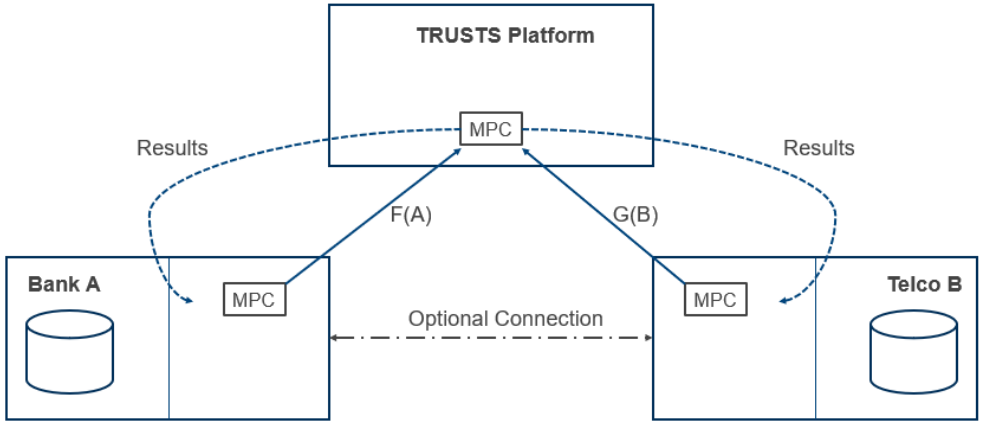
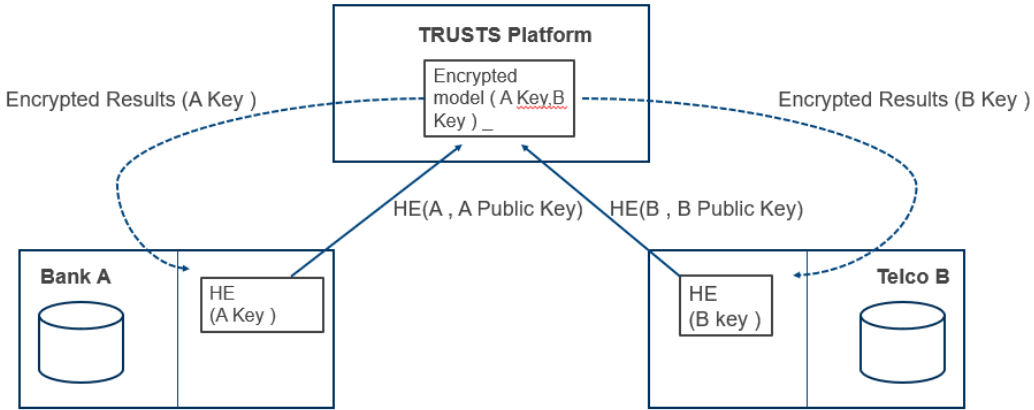
Privacy enhancing technologies (PETs) are an additional concern of the TRUSTS platform. Related to this are the objectives of investigating, designing and improving cryptographically secure protocols that enable data analysis of privacy-sensitive data. Consequently, the focus will be on practical aspects of cryptographic building blocks such as, but not limited to, secure multiparty computation and homomorphic encryption.

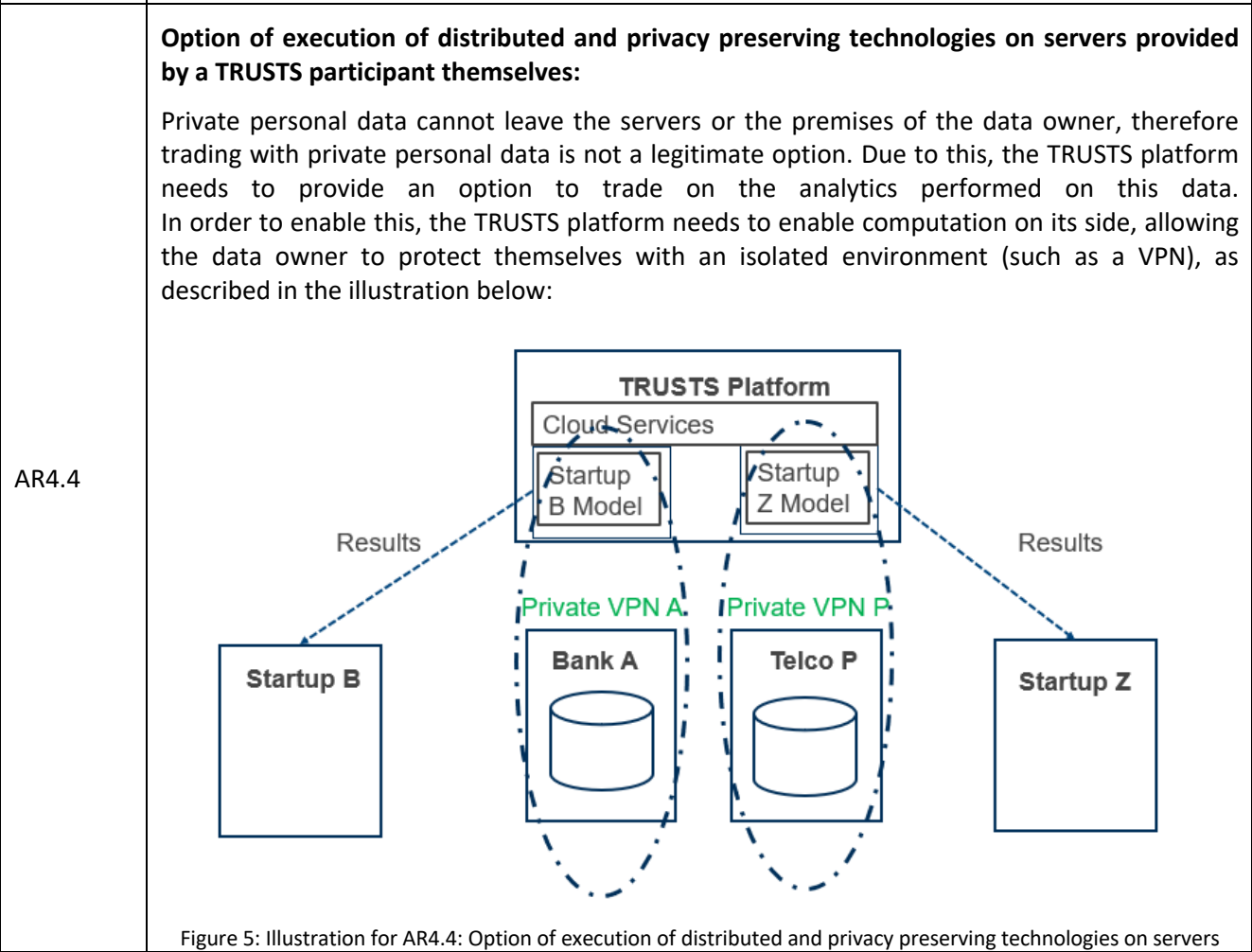
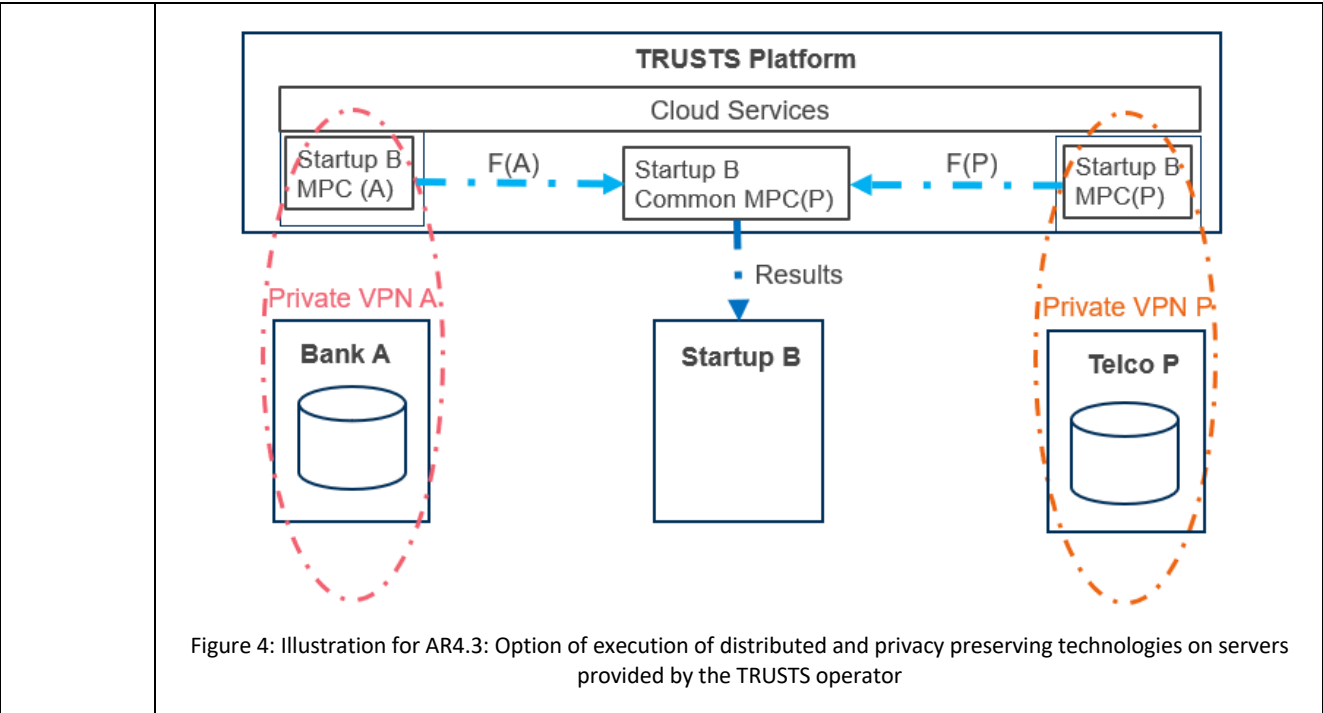
The results of this research will enable parties to collaborate over their private sensitive data and run advanced analytics on multi-party datasets while preserving the data privacy.

The requirements below support several methods to preserve data privacy. These requirements can be fulfilled by using data encryption methods such as Homomorphic encryption and sharing that encrypted data. Alternatively, other related methods can be used, like Ensemble Modeling that enables advanced analytics over multiple datasets. Each of the data owners will execute the analytics separately on their premises while sharing only the models' algorithm, which will preserve the data privacy.

The work on privacy enhancing technologies for the TRUSTS platform results in the following architecture requirements:

WP 4 - Privacy preserving technologies	
AR 4.1	<p><b>Computation using distributed and controlled execution environments:</b></p> <p>Since private personal data cannot leave the premises of the data owner, TRUSTS must provide an option to run analytics on the data while preserving the privacy of the users contained in the data. In order to collaborate over private personal data, TRUSTS should supply the kind of cloud services that will be part of the computation as described in the illustration below.</p>

	 <p>Figure 2: Illustration for AR 4.1: Computation using distributed and controlled execution environments</p>
AR4.2	<p><b>Machine Learning using distributed and privacy preserving technologies:</b></p> <p>In order to execute Machine Learning models in TRUSTS, the platform must provide a way to develop, test and execute ML models, while supporting MPC /HM/Federated learning and similar services.</p>  <p>Figure 3: Illustration for AR4.2: Machine Learning using distributed and privacy preserving technologies</p>
AR4.3	<p><b>Option of execution of distributed and privacy preserving technologies on servers provided by the TRUSTS operator:</b></p> <p>As described in the previous requirements, in order to collaborate over private sensitive data, the data owner must run the relevant ML model on their own servers. In order to give the data owner an alternative to using their servers for computation, TRUSTS should enable the option to execute the relevant computation on TRUSTS servers, while enabling access only to the data owner. This can be done for example using a virtual private network (VPN) created by the data owner.</p>



	provided by a TRUSTS participant themselves
AR4.5	<p><b>Development of distributed services using privacy preserving technologies:</b></p> <p>The TRUSTS platform needs to provide an option to develop, test, and execute services using privacy preserving technologies. In addition, capabilities for logging and monitoring of these technologies needs to be provided. These capabilities need to take account of the distributed nature of privacy preserving technologies on the TRUSTS platform.</p>
AR4.6	<p><b>Scalability of the TRUSTS platform:</b></p> <p>The TRUSTS platform needs to provide scalability for technologies which are dependent on computation and network throughput for their performance.</p>
AR4.7	<p><b>Ensemble Modeling:</b></p> <p>Since private personal data cannot leave the premises of the data owner, TRUSTS must provide an option to run analytics on the data while preserving the privacy of the users contained in the data.</p> <p>One of the suggested methods is to use Ensemble Modeling. The outcome of this method will be trading of pre-trained models that were trained to solve the same problem.</p> <p>There should be a kind of application store that will enable download and update of pre-trained models.</p>

### 2.1.10 Architecture requirements related to anonymization and de-anonymization

Data controllers prior to sharing their data need to become aware of the privacy risks in their data and apply the appropriate anonymisation measures. The privacy risks arise because of de-anonymisation: the process of identifying individuals in a dataset that does not contain any personally identifiable information (PII), such as full name and address. To confront the privacy risks, anonymisation methods have been introduced. Anonymisation methods distort a dataset in such a way, so that it conforms to a privacy model. In relation to this, the aim is to provide an application that offers de-anonymization risk analysis modules for different kinds of data, as well as anonymization methods that offer compliance to various privacy models. The application's aim is to raise awareness on the privacy risks of a data controller's dataset, aid in the decision of the anonymization measures and their extent, and help data controllers comply with the general data protection regulation (GDPR) by protecting the privacy of the individuals in their datasets.

The figure below gives an overview of the usage of the application. Given two actors – a data seller and a data buyer – the usage flow is as follows:

1. The data seller and the data buyer download the application from the TRUSTS platform to their premises. The application needs to be executed on the TRUSTS users' premises because non-anonymised, privacy-sensitive, personal data are processed by the application, and such data should not be uploaded to the platform. Additionally, to ensure that only verified TRUSTS users are using the application, the platform needs to provide means of remote (i.e., from the users' premises) authentication.
2. The data seller imports their non-anonymised, privacy-sensitive, personal data to the application.
3. The data seller uses the de-anonymisation risk analysis modules.

4. The data seller uses the anonymisation modules.
5. The data seller transfers the anonymised data to the data buyer. This will be feasible using the capabilities of the IDS Trusted Connector.

The application will be packaged as a docker container to facilitate smooth execution and minimal installation and configuration by the TRUSTS users.

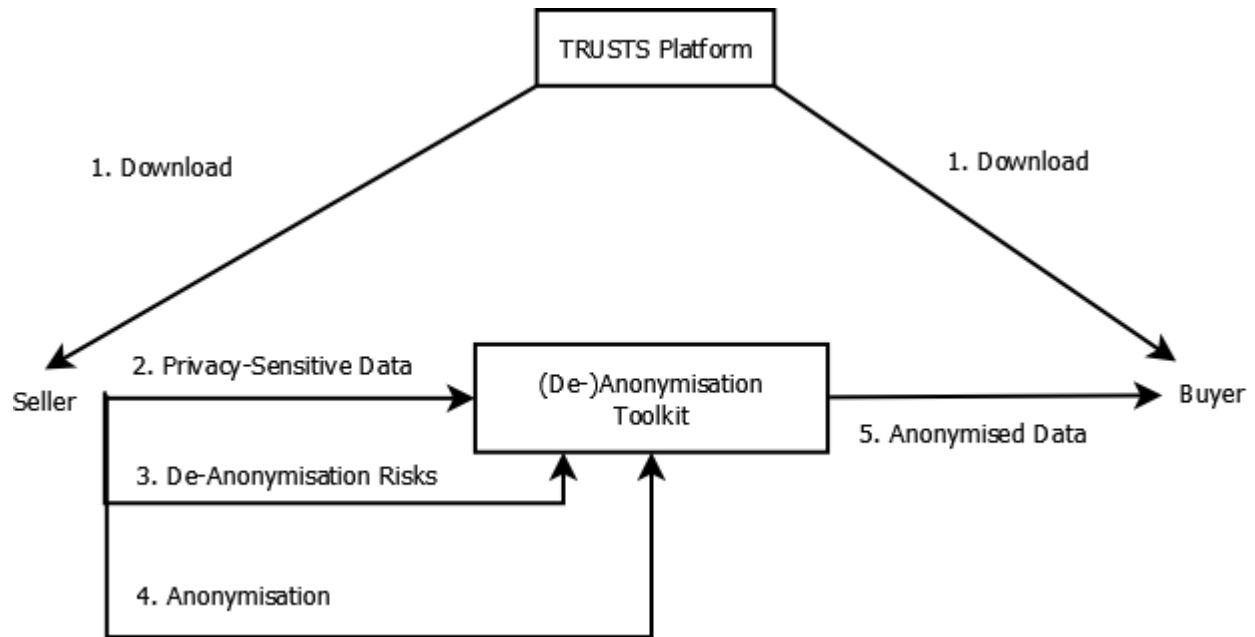


Figure 6: Architecture requirements related to anonymisation and de-anonymisation

The resulting requirements for the architecture are as follows:

Architecture requirements related to anonymisation and de-anonymisation	
AR 4.3.1	<p><b>Applications need to be available to TRUSTS platform users for download</b></p> <p>The task's output is an application with de-anonymization risk analysis and anonymisation modules that will be used both by data sellers and buyers. Since the application needs to process non-anonymised, private, personal data of a data seller, the application needs to be available for download, in order for it to be executed at the users' premises</p>
AR 4.3.2	<p><b>Applications need to be hosted on the platform in a form that allows execution on the users' premises</b></p> <p>In order to facilitate smooth execution and minimal installation and configuration, the application will be packaged as a docker container. Such a package, however, could be rather sizable (hundreds of MBs to a few GBs). The platform needs to be able to host such sizable files/packages.</p>

AR 4.3.3	<b>Login and Authentication for Applications</b> In order to make sure applications are used only by the TRUSTS platform users, logging in to the application and authentication should require validation by the platform.
AR 4.3.4	<b>IDS Connector</b> The transfer of anonymised data from a seller to a buyer has to be done through IDS connectors installed on the premises of the users. The platform should be able to support the installation of IDS connectors on the users' premises.

### 2.1.11 Architecture requirements following from the usage of Comprehensive Knowledge Archive Network (CKAN) software

Based on the collection of functional requirements in deliverable D2.2 “Industry specific requirements analysis, definition of the vertical E2E data marketplace functionality and use cases definition I”, it became clear that the TRUSTS platform requires a user facing component for data management and related activities.

A common best-practice in software engineering is to reuse existing software libraries in order to benefit from the stability and maturity of the library. Furthermore, existing libraries are ideally well-maintained and have a community around them taking care of removing bugs or adding new features. Consequently, reusing existing solutions increases robustness of the own solution. In TRUSTS, we aimed to follow this best-practice and examined different existing solutions from the area of data management platforms. This resulted in the selection of CKAN<sup>2</sup> as the main user facing component for data management and related activities on the TRUSTS platform.

CKAN is an open-source library to run self-hosted data management platforms. It provides a rich feature set such as a built-in data catalog, search facilities, as well as user management for fine-grained role assignment. CKAN has a dedicated extension mechanism to increase flexibility and add further functionality. The off-the-shelf features of CKAN overlap with functional requirements identified in deliverable D2.2. CKAN has been selected to become a central component during the establishment of the TRUSTS platform given the following advantages:

- Significant overlap of existing CKAN features with TRUSTS requirements.
- Availability of data cataloging functionality.
- Included search functionality using Solr.
- Deployable using container technology.
- Fine-grained adjustment of publicity levels for assets, i.e. public or private.
- Robust and mature solution. Thousands of productive instances are deployed worldwide.
- Widely used by a plethora of data providers, e.g. governments (e.g. Australian Federal Open Data Portal<sup>3</sup>) or other players (e.g. AQUACROSS<sup>4</sup>)

<sup>2</sup> <https://ckan.org/>, last accessed April 23, 2021

<sup>3</sup> <https://data.gov.au/>, last accessed April 23, 2021

<sup>4</sup> <http://dataportal.aquacross.eu/>, last accessed April 23, 2021



- Active maintenance by and responsiveness of the CKAN community
- A plethora of existing CKAN extensions that fulfil the requirements of TRUSTS, e.g. a harvesting extension, a DCAT extension, or an OAUTH2 extension.
- The availability of a convenient extension mechanism facilitating the creation of features specific to TRUSTS requirements

CKAN serves as a ready-to-use solution to provide an initial version of the user facing functionality of the TRUSTS platform. Deployed on the TRUSTS cloud services, stakeholders within TRUSTS can access and utilize the platform, register themselves and their companies, as well as upload datasets. This procedure helps to familiarize with the concept of data markets, unveils hidden obstacles and opportunities for improvement, as well as to detect flaws and bugs in the software. This initial version of the platform will continually undergo improvements. Additional features identified via the functional requirements will either be implemented from scratch or integrated from existing projects such as the DMA or the IDS. Consequently, the out-of-the-box implementation of CKAN will be extended with regards to features, scope, and functionality and will also be used to integrate with software modules from other work packages.

For the selection of a suitable out-of-the-box solution, several other existing libraries were examined as well. However, they were not found to satisfy some of the requirements of TRUSTS. For example, Dataverse does not provide an extension mechanism or an ecosystem of existing extensions, and also does not have fine-granular adaptation of the publicity level of data assets. The latter two also apply to Fedora Commons, in addition to the fact that it is not well-maintained. Lastly, Invenio lacks maturity and an extension ecosystem, and requires significant setup effort, making it ineligible as an out-of-the-box solution.

Requirements following from the usage of CKAN	
AR 3.3.9	<p><b>CKAN extension mechanism</b></p> <p>TRUSTS components should ideally be integrated using CKAN's built-in extension mechanism. Developers need to consider this and design and develop their solutions in a way that they can be easily integrated into CKAN.</p>

## 2.2 Summary of Architecture Requirements

The following is a summary of all previously listed architecture requirements. We have grouped all architecture requirements into architecture requirement summaries, labelled ARS, and provided a short description of each such summary. The original architecture requirements (ARs) which are summarised by an ARS, are listed in the right most column. Every original architecture requirement is included in at least one ARS.

Table 2: Summary of Architecture Requirements

ARS ID	Short description	Original AR numbers

ARS 1	<b>Smart contracts, which are based on data sharing policies, are supported.</b>	3.X.1 3.X.5 3.2.1
ARS 2	<b>All transactions on any node, which is part of a TRUSTS platform instance, are logged.</b>	3.X.1 3.X.5 3.2.2
ARS 3	<b>All data assets / data products in a TRUSTS platform instance, use metadata schema or vocabularies.</b>	3.X.5 3.X.6 3.3.1 3.4.11 3.5.8
ARS 4	<b>External data marketplaces can be integrated into a TRUSTS platform instance.</b>	3.X.3 3.3.2 3.3.3 3.3.4
ARS 5	<b>The operator of a TRUSTS platform instance can use an administrative interface. Users of the platform also have an interface available.</b>	3.X.4 3.3.5 3.4.12
ARS 6	<b>Every TRUSTS platform instance is a distributed set of nodes.</b>	3.X.3 3.4.1
ARS 7	<b>Each node runs software components, which are distributed as docker containers, and can be configured via configuration files.</b>	3.X.4 3.X.5 3.4.2 3.5.6 3.5.9
ARS 8	<b>Each node needs to run an instance of the trusted connector. It is required for communication between nodes and within a node</b>	3.X.1

		3.X.2 3.X.3 3.4.3 4.3.4 3.5.1 3.5.2 3.5.3 3.5.5
ARS 9	<b>Each node has an internal directory of running software components</b>	3.X.3 3.X.5 3.4.4
ARS 10	<b>A node can internally be a distributed node itself.</b>	3.X.3 3.4.5
ARS 11	<b>Every TRUSTS platform instance has at least one node which runs an instance of the Broker component.</b>	3.X.3 3.X.6 3.4.6
ARS 12	<b>A node must be able to run apps on premise.</b>	3.X.3 3.4.7 4.3.1 4.3.2
ARS 13	<b>Assets (such as data assets) in a node can be managed locally.</b>	3.X.3 3.X.5 3.4.8
ARS 14	<b>A node is able to harvest metadata from its local assets.</b>	3.X.3 3.X.5 3.4.9
ARS 15	<b>Recommendations in a node are integrated into the TRUSTS platform, which</b>	3.X.4

	<b>provides data, delivers recommendations and returns feedback.</b>	3.4.10 3.6.1 3.6.2 3.6.3
ARS 16	<b>Instances of the trusted connector support existing security infrastructure inside of a node, and connect it to the usage control of the TRUSTS platform.</b>	3.X.1 3.X.2 3.5.4 3.5.8 4.3.3
ARS 17	<b>Computation using distributed and controlled execution environments is possible in every instance of the TRUSTS platform.</b>	3.X.1 3.X.3 4.1
ARS 18	<b>Machine Learning using distributed and privacy preserving technologies is possible in every instance of the TRUSTS platform.</b>	3.X.1 3.X.2 3.X.3 4.2
ARS 19	<b>Every instance of the TRUSTS platform provides the option of execution of distributed and privacy preserving technologies on nodes provided by the operator of the instance.</b>	3.X.1 3.X.2 3.X.3 4.3
ARS 20	<b>Every instance of the TRUSTS platform provides the option of execution of distributed and privacy preserving technologies on servers provided by a TRUSTS participant themselves.</b>	3.X.1 3.X.2 3.X.3 4.4
ARS 21	<b>Development of distributed services using privacy preserving technologies is possible in every instance of the TRUSTS platform. (This refers to Devops.)</b>	3.X.1 3.X.2 4.5

ARS 22	<b>Every instance of the TRUSTS platform provides scalability for services and apps running on the platform instance.</b>	3.X.3 4.6
ARS 23	<b>The user interfaces of all TRUSTS components are integrated into the CKAN extension mechanism.</b>	3.X.4 3.3.9

### 3 Technical architecture of the TRUSTS platform

The architecture of the TRUSTS platform represents the blueprint for the results of the TRUSTS project. As such, it also represents the foundation for instances of the TRUSTS platform which will be provided by one or more TRUSTS operators after the duration of the project.

In this section, we provide a description of the architecture of the TRUSTS platform, in terms of component descriptions, interconnections and other technical details of the architecture. First, we provide a general overview of the architecture and explain the most important concepts of the architecture. This is followed by a description of each component of the architecture, as well as how components are connected within the platform, as well as to external data marketplaces for the purpose of interoperability and federation. The component descriptions are followed by an overview of the mapping between functional requirements (FRs) and components, and the mapping between architectural requirements (ARs) and components.

#### 3.1 Overview of the technical architecture

The TRUSTS platform is a set of interconnected nodes, each owned and operated by different organizations, on which the trusted exchange of digital assets is possible. Each node is a computing environment (e.g., a virtual machine) which runs a set of standardized components to support the different functionalities of the platform. Along these components, common to all nodes, organizations can execute their in-house developed applications and services. These applications and services can then be traded through the TRUSTS platform, and can also, in turn, consume and produce other assets through the platform.

Components deployed within a node are connected among themselves through a network environment provided and secured by the node's operator. The connections between different nodes are all done using the IDSCPv2 communication protocol, and are handled by the set of standardized components. In particular, connections coming from outside the node are first received by the Trusted Connector component and then distributed to other components. The IDSCPv2 protocol, apart from the standard asymmetric-key encryption of modern HTTPS connections, allows for conveying information specific for the functionalities of the platform. This includes user identification tokens, names and ports of the different nodes involved, as well as IDs of the components they are destined to. A schematic of these network connections is shown below.

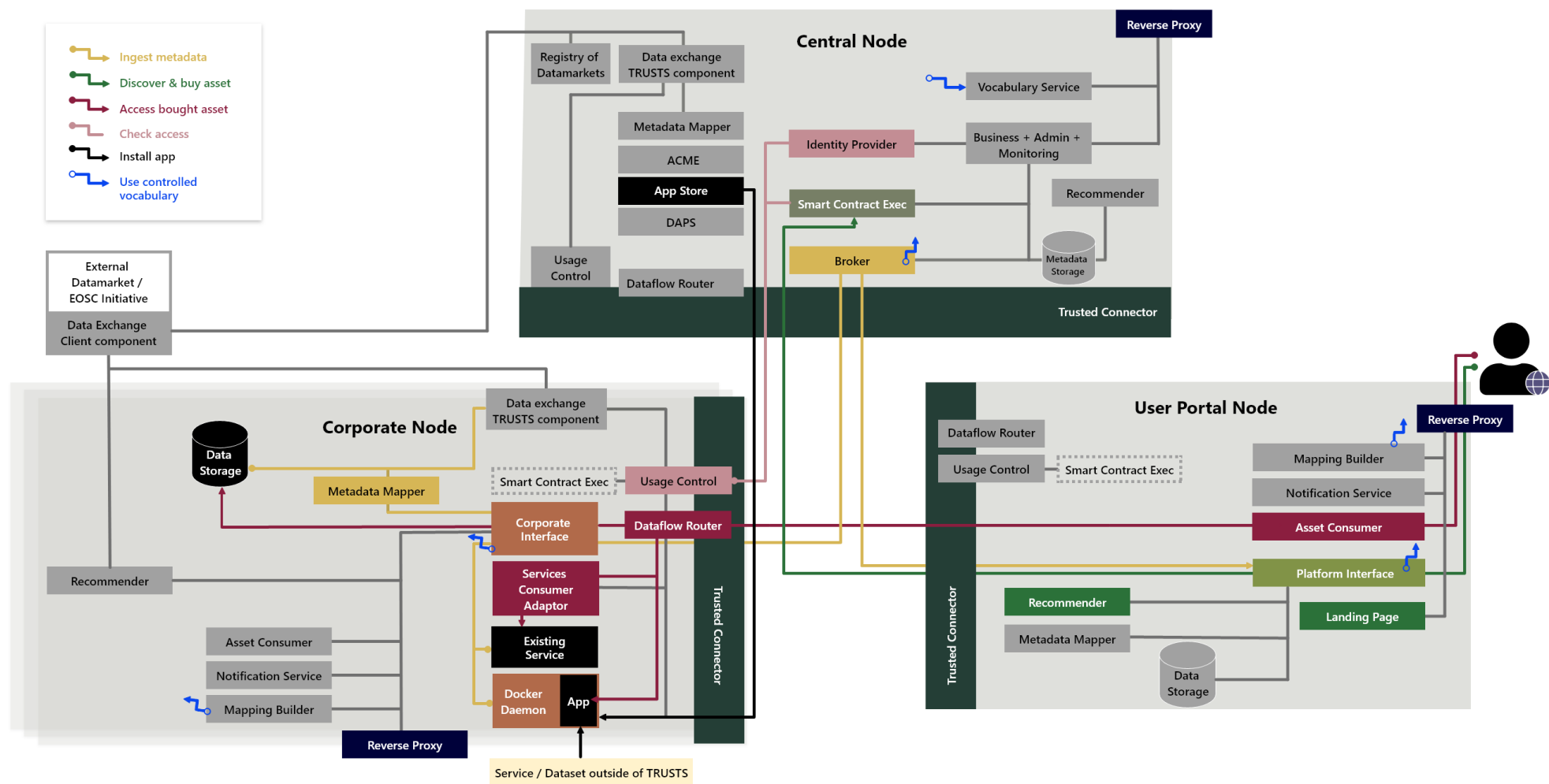


Figure 7: Diagram of the technical architecture

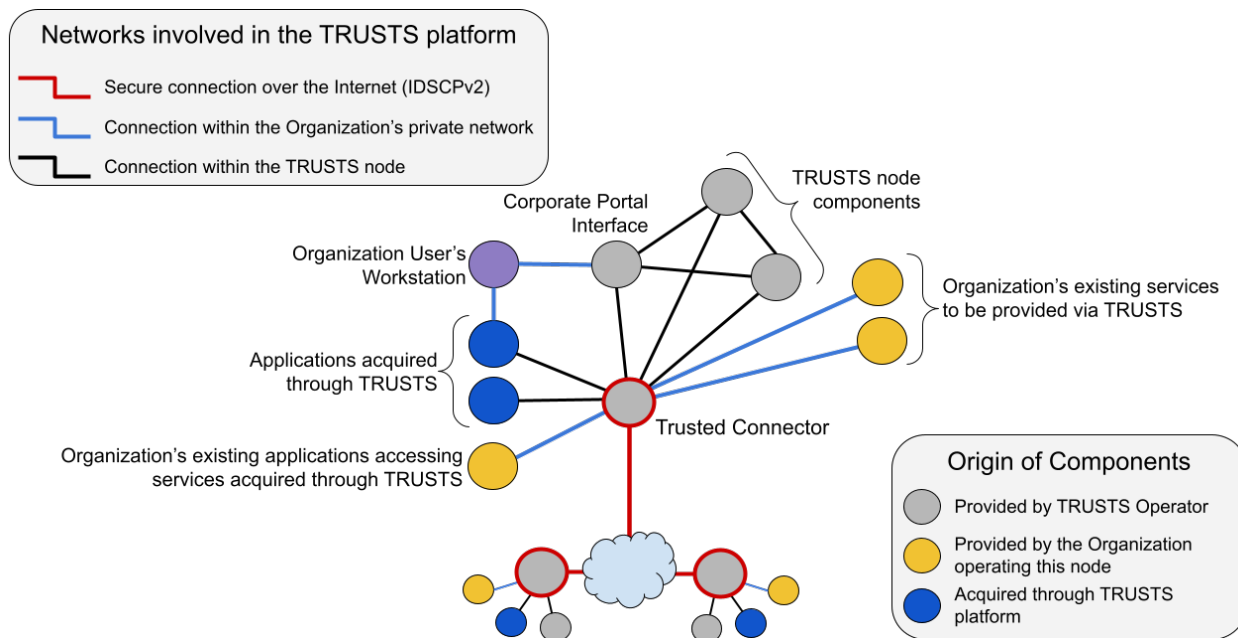


Figure 8: Overview of networks involved in the TRUSTS platform

The entire list of components within a node are listed in section 3.2. Using this standardized set of components in a node allows for a consistent set of functionalities to be available to all participants in the TRUSTS platform, as well as enforcing a set of constraints on the functioning, trading and access to assets within nodes. These constraints and functionalities are themselves the implementation of the functional requirements specified for the TRUSTS platform, as listed in deliverable D2.2. A description of how these components are used, and how they are related to the functional requirements and the architecture requirements can be found in section 3.2 below.

For an organization to install a TRUSTS node, they must enter a subscription agreement with the TRUSTS operator, after which they will be provided with a series of software artifacts, cryptographic certificates and instruction manuals for the set-up. This will involve, among other steps, creating a dedicated computing environment with a controlled set of open ports and other firewall rules. These rules shall imply, for example, that the only connection between the node and the internet will be through the Trusted Connector. Furthermore, it shall allow for other connections inside the organization's private network, in order for applications acquired through the TRUSTS platform to be accessible to the organization's users, for example. Finally, a set of technical information exchanges should be carried out between the TRUSTS operator and the node operator, for example, regarding a set of IP addresses and ports which are to be contact points between the nodes.

### 3.1.1 Types of nodes

Among the nodes of the platform, there are two which are of special importance since they host a set of services necessary for the operation of the platform as a whole. These set of components, akin to the IDS essential services, are to be hosted by one special organization, referred to as the TRUSTS Operator, which holds a position of special trust with the other organizations hosting nodes. The two nodes owned by the

TRUSTS Operator are the **User Portal Node** and the **Central Node**, while every other node is referred to as a **Corporate Node**.

All three types of nodes have infrastructure built around the Trusted Connector

- **Corporate Node** Is used by partners of TRUSTS that have complex infrastructure and participate in TRUSTS. They can have many users and many services and applications that are provided or consumed via the TRUSTS platform. They can set any authorization system and communication structure inside their nodes. The TRUSTS Operator is not responsible for setting up or supporting instances of corporate nodes, but will provide detailed instruction manuals and all necessary software. Among the components being set up, is the Corporate Interface, a web application through which the organization's users can interact with the TRUSTS platform.
- **User Portal Node** Is created to cover the needs of individual users of the TRUSTS platform. It is set up and maintained by the TRUSTS Operator. By accessing this node, individual users can benefit from some of the functionalities of the TRUSTS platform, without the complexity of setting up a corporate node. Additionally, it is the entry point for all organizations intending to join the TRUSTS platform, as it provides landing pages, legal information, and setup instructions. One of the components included in this node is the platform Interface, which acts as the main point of access to the above mentioned functionalities. In principle, the TRUSTS platform could have more than one User Portal Node.
- **Central Node** Exists to support the operation of the whole TRUSTS platform, playing the role of authorization, monitoring, smart contract executor, catalog, application repository, among others. This node is created and maintained by the TRUSTS Operator.

### 3.1.2 Data sets / services / applications

Three types of assets are envisioned to be traded among the participants of the TRUSTS platform. The specific choice of these types of assets was made after careful analysis of the requirements collected in Deliverable 2.1 as well as further discussion with the different stakeholders of the three use cases of the project. These three types of assets are listed and defined below. Section 3.2 describes how each of these is handled by the TRUSTS platform, whose components are described below.

**Dataset.** These are static files which can be traded. They are equivalent to the notion of DCAT:Dataset, in terms of metadata and scope. These files are transmitted directly from provider to consumer, and once they have been received, they are, from the technical point of view, mere files in the consumer's file system.

**Service.** These are computer programs which are executed in the provider's node. They can, upon contractual agreement, be executed on request of a consumer with the input of their choice. Services are envisioned to function as do standard web services: a server is running and accepts connections, processes the input, and returns the output to the requester. There is no prescription on the stateful-ness of services, but they have to expose an HTTP interface.

**Application** These are computer programs which are executed in the consumer's node. They are distributed as file bundles, which consist of a combination of container images, deployment scripts and configuration files. The provider of the application has total freedom regarding implementation and functionalities, and can



choose to make the application accessible only through the consumer node's Trusted Connector instance, thus enabling the use of TRUSTS provided functionalities, such as user authentication.

### **3.1.3 Connections within the TRUSTS architecture**

In the general architecture diagram (Figure 7), six specific data flows are pictured. These are meant as examples of the interactions between the different components in different nodes, and aim to specify how the choice of components and their arrangement relate to the functional requirements.

#### **Communication protocols used by components within the TRUSTS platform**

To properly and fully integrate components and services within the TRUSTS platform, those components and services need to provide a description of existing interfaces based on the best practices which are summarised by the REST (representational state transfer) paradigm. The REST paradigm for describing services is the most widely used web service interconnection approach and the most common standard in interaction between distributed software components. Additionally, services using REST are much easier to test compared to other service specifications, such as MQTT and SOAP. RESTful services have no restriction on the size of messages and are a lightweight approach to integration. RESTful services are supported by the trusted connector component, which is used in the TRUSTS platform.

#### **Ingest metadata**

When an organization wishes to make an asset it possesses available through the TRUSTS platform, it must make metadata about the asset available. This process is referred to as onboarding of assets in the requirements documents.

The origin of the metadata can be a metadata storage about datasets (e.g. a directory in a file system, a csv file, a data management platform), a machine-readable description of a web service, or a standardized description of deployment and configuration of an application. These three sources are all accessed, as per input by the user in charge of onboarding, by the Corporate Interface running in the organization's node, which collects the metadata and adds to it metadata about the node itself. This process can be assisted by the Metadata Mapper component which can convert from several metadata schemas into the one being used by the TRUSTS platform. Finally, the Corporate Interface also provides the user with the means to specify the contracting options by which the asset in question will be made available.

When the user is ready to make the asset available for purchase and access through the TRUSTS platform, they can request the Corporate Interface to send information about it to the Broker component present in the Central Node. The broker then stores this metadata in the authoritative central metadata repository, which the Platform Interface in the User Portal Node, or the Corporate Interface in any Corporate Node, will query.

#### **Discover and buy asset**

When a user wishes to acquire an asset through the TRUSTS platform, they must first present credentials either in the Platform Interface or in any Corporate Interface to which they have access. This interface will present them with the tools necessary to search, select an asset and select access options (including contracting and

billing). Once the process is complete, a record of the transaction is entered into the smart contract executor component in the Central Node.

### **Access bought asset**

For a user to access an asset they have acquired, they will request access through an Asset Consumer component located in the node in which they have credentials. This request can be, for example, the result of clicking a “Download Dataset” link in either the Platform or the Corporate Interface, an HTTP request made by a software application controlled by the user and wishing to access an asset on their behalf, or an application-deployment mechanism (e.g. Docker pull) requesting download of a docker image. In any case, the asset consumer will convert this request into an IDSCPv2 message requesting access, coupling it with the user’s authentication token, signing it with the appropriate cryptographic certificate, and forwarding to the Trusted Connector of the node providing the asset. This Trusted Connector will receive the request, consult with the node’s Usage Control component (see below) and, if appropriate, forward the request to the component which hosts the asset, such as a running HTTP server (in the case of services) or the Corporate Interface (in the case of datasets).

### **Check access**

When a Trusted Connector receives a request for an asset hosted in its node, it will forward certain metadata about the request to the Usage Control component. This component will, in turn, request information from the smart contract executor (running in the Central Node) which will return the status of the requester’s contract for the asset in question. If the status is favourable, the Trusted Connector will be notified and the Dataflow Router will forward the request appropriately (see above), the smart contract executor will be notified of the access operation for it to be recorded, and the Smart Contract Execution module will execute any associated smart contract.

### **Install application**

Installing an application that has been acquired through the TRUSTS platform will be done by downloading the applications configuration bundle from the provider’s node (see above) into the consumer’s node. This will be followed by a *pull* operation by the Docker Daemon running in the consumer’s node. This operation will also be validated (see above) for compliance with the contract pertaining to the application, before the image is transferred. Once installed, the application will be executed according to the configuration files that are part of the bundle.

### **Use controlled vocabularies**

Several components within the TRUSTS platform make use of controlled vocabularies for, e.g. storing and displaying labels describing assets, or enriching search results. This will be done by accessing a centralized Vocabulary Service, running in the Central Node, which will be kept up-to-date by the TRUSTS operator.

## **3.1.4 Interoperability with External Data Markets and EOSC Initiatives**

One of the objectives of the TRUSTS project is a connection to third party data markets and EOSC initiatives. TRUSTS will develop a component dedicated for this task, the so-called interoperability component. This

component will help external stakeholders to connect with TRUSTS and share and exchange their resources on TRUSTS. The interoperability solution consists of a component residing on the premises of the external data market or EOSC initiative as well as a TRUSTS component interacting with the client component. The components exchange information such as metadata, which in turn will be stored in the storages of TRUSTS.

A significant challenge of the development of this component is the strong diversity of the technological characteristics of external data markets and EOSC initiatives. Consequently, TRUSTS will not develop a solution for each of them. Instead, TRUSTS aims to build a solution exposing the requirements of TRUSTS, i.e., the TRUSTS metadata schema. This schema is an extension of the IDS Information Model. It is an open, non-proprietary, based on a widely accepted model (such as DCAT<sup>5</sup>) and covers a wide variety of technological requirements. External stakeholders will turn their metadata and data assets into a format understood by TRUSTS. The interoperability solution will be developed based on the requirements of a select set of external data markets and EOSC initiatives.

The two mentioned components, i.e., the Data Exchange Client Component and the Data Exchange TRUSTS Component, are supported by the Registry of Data Markets. This component contains up-to-date information about existing data markets and will help to identify data markets suitable for building the interoperability prototype.

## 3.2 Components of the architecture

For each component we list the following attributes:

- Component number.
- Component name.
- Short component description.
- Input dependencies: the names of other components which provide input for the component.
- Output dependencies: the names of other components which require output from the component.
- Requirements addressed: references to the functional and architectural requirements which are addressed by the component.

<b>C1</b>	<b>Trusted Connector</b>
Interface to nodes in the TRUSTS platform. Provides certified connections between pairs of nodes, serving as a proxy between the different services running on a node, and other nodes. It is based on Apache Camel and has functionalities for transaction management / usage control.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Dataflow router</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Corporate Interface</li> </ul>

<sup>5</sup> DCAT: <https://www.w3.org/TR/vocab-dcat-2/>, last accessed April 23, 2021

<ul style="list-style-type: none"> <li>• Corporate Interface</li> <li>• Usage Control</li> <li>• Automated Certificate Management Environment (ACME)</li> <li>• Dynamic Attribute Provisioning System (DAPS)</li> </ul>	<ul style="list-style-type: none"> <li>• Platform Interface</li> <li>• Usage Control</li> <li>• Broker</li> </ul>
<b>Requirements addressed:</b> FR27, FR28, FR31, FR38, FR39, FR40, FR41, FR42, FR43 ARS2, ARS7, ARS8, ARS12, ARS16, ARS17, ARS18, ARS19, ARS20, ARS21	

<b>C2</b>	<b>Dataflow Router</b>
Add-on to the trusted connector that initializes and destroys routes within a node as required by incoming and outgoing connection. For example, if there are N services running in a node (e.g. a CKAN instance, some APIs) and the Trusted Connector receives a request a access one particular service, this component creates a route in the Apache Camel configuration of the connector.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Corporate Interface</li> <li>• Platform Interface</li> <li>• Usage Control</li> <li>• Trusted Connector</li> <li>• Services Consumer Adaptor</li> <li>• Operation Converter</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Corporate Interface</li> <li>• Platform Interface</li> <li>• Usage Control</li> <li>• Trusted Connector</li> <li>• Services Consumer Adaptor</li> <li>• Operation Converter</li> </ul>
<b>Requirements addressed:</b> FR 27 ARS8, ARS9, ARS10, ARS16, ARS22	

<b>C3</b>	<b>Reverse Proxy</b>
For connections from a web-browser into nodes (e.g. for accessing portals), this component distributes the requests among the different services. Implementation can be Nginx, Traefik, HAProxy or similar.	

<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• None.</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Corporate Interface</li> <li>• Platform Interface</li> <li>• Mapping builder</li> <li>• Notification Service</li> <li>• Asset consumer</li> <li>• Business Support Services</li> </ul>
<b>Requirements addressed:</b> ARS5, ARS23	

<b>C4</b>	<b>Recommender</b>
Provide four recommendation settings: (i) recommendation of services to users/corporates, (ii) recommendation of datasets to users/corporates, (iii) recommendation of datasets to services, and (iv) recommendation of services to datasets.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Broker + Metadata Storage</li> <li>• Platform Interface</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Platform Interface</li> </ul>
<b>Requirements addressed:</b> FR6, FR7, FR8 ARS15	

<b>C5</b>	<b>Platform Interface</b>
This component should support users with opening accounts, searching for datasets and services (free and chargeable) and consuming them. It should provide means to sign contracts with providers of datasets and services. Additionally it allows consumers to rate dataset and service providers.	
<b>Input dependencies:</b>	<b>Output dependencies:</b>

<ul style="list-style-type: none"> <li>• Dataflower Router</li> <li>• Recommender</li> <li>• Vocabulary Service</li> <li>• Metadata Mapper</li> <li>• Broker</li> </ul>	<ul style="list-style-type: none"> <li>• Dataflower Router</li> <li>• Trusted Connector</li> </ul>
<b>Requirements addressed:</b> FR1, FR3, FR4, FR5, FR25, FR27, FR29, FR30, FR31, FR32, FR33, FR34, FR36, FR37, FR44 ARS5, ARS23	

<b>C6</b>	<b>Landing Page</b>
A website for non-registered users to learn about TRUSTS platform, legal terms, contact points, etc.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Reverse Proxy</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Platform interface</li> </ul>
<b>Requirements addressed:</b> ARS5	

<b>C7</b>	<b>Asset Consumer</b>
Satisfies the use case when a user logs into platform node X using their web browser and wants to access an asset on node Y. Given that node Y only serves assets through the Trusted Connector, this component, executed in node X, routes the browser's request through the Trusted Connector, into node Y.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Reverse Proxy</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Trusted Connector</li> </ul>
<b>Requirements addressed:</b> FR27, FR29 ARS13, ARS14	

<b>C8</b>	<b>Notification Service</b>
This component informs Consumers about changes in datasets and services that they already have access too.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Broker + Metadata Storage</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Notification service</li> </ul>
<b>Requirements addressed:</b> FR35 ARS5, ARS13	

<b>C9</b>	<b>Metadata Mapper</b>
This component transforms existing metadata that organizations might have in their pre-existing data stores, into the IDS-IM specification. Once this metadata is converted, it can be ingested into the local Metadata Registry and then broadcast into the Central Catalogue for advertising to other participants.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Mapping Builder</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Platform Interface</li> <li>• Corporate Interface</li> </ul>
<b>Requirements addressed:</b> FR23 ARS3, ARS14	

<b>C10</b>	<b>Usage Control</b>
This ad-on to the Trusted Connector provides Access and Usage Control to Datasets and Services. When a	

request to access an asset is received, this component checks if there is a contract between provider and consumer for said asset, and if that contract is in a valid state. If so, invokes the Dataflow Router for further processing of access.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Smart Contract Execution</li> <li>• Identity Provider</li> <li>• Trusted Connector</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Dataflow Router</li> <li>• Smart Contract Executor</li> </ul>
<b>Requirements addressed:</b> FR29, FR38, FR44 ARS16	

<b>C11</b>	<b>Mapping Builder</b>
A user interface that helps providers builds Mapping files to convert their metadata into the IDS-IM, which is TRUSTS metadata model.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Vocabulary Services</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Metadata Mapper</li> </ul>
<b>Requirements addressed:</b> FR23, FR33 ARS3	

<b>C12</b>	<b>Corporate Interface</b>
This component allows corporate inside TRUSTS Corporate node to manage the assets that they wish to provide through the TRUSTS platform. Also it should provide means to sign contracts with consumers of datasets and services. Additionally it allows consumers to rate dataset and service providers.	
<b>Input dependencies:</b>	<b>Output dependencies:</b>



<ul style="list-style-type: none"> <li>● Usage Control</li> <li>● Dataflower Router</li> <li>● Metadata Mapper</li> <li>● Recommender</li> <li>● Services Consumer Adaptor</li> </ul>	<ul style="list-style-type: none"> <li>● Dataflower Router</li> <li>● Trusted Connector</li> <li>● Usage Control</li> <li>● Recommender</li> </ul>
<b>Requirements addressed:</b> FR1, FR3, FR4, FR5, FR27, FR29, FR30, FR31, FR32, FR34, FR36, FR37, FR44 ARS5	

<b>C13</b>	<b>Services Consumer Adapter</b>
This component allows for services that are deployed by a participant in their Corporate Node to be offered through the TRUSTS platform. Thus, all pre-existing web APIs (e.g. REST) can be accessed through a unified gateway that incorporates usage control and user authentication synchronized with the TRUSTS platform.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>● Trusted Connector</li> <li>● Identity Provider</li> <li>● Usage Control</li> <li>● Pre-existing (legacy) services</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>● Trusted Connector</li> </ul>
<b>Requirements addressed:</b> ARS6, ARS10	

<b>C14</b>	<b>Data Exchange TRUSTS Component</b>
This component is deployed in TRUSTS nodes and ingests assets from third-party data-markets and EOSC and its related initiatives into the TRUSTS catalog. It receives input from the Data Exchange Client Component and converts it, if necessary, into the format required by TRUSTS. This component is also connected to the Registry of Data markets, receiving information about data markets and EOSC and its related initiatives connected with TRUSTS. Based on this information, the component connects to the respective Data Exchange Client Components and acquires their metadata or data assets.	

<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Third Party Marketplaces</li> <li>• EOSC and its related initiatives</li> <li>• Registry of Data markets</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Metadata Mapper</li> <li>• Usage Control</li> </ul>
<b>Requirements addressed:</b> FR2, FR4, FR23 ARS4	

<b>C15</b>	<b>Data Exchange Client Component</b>
This component resides at the location of a third party data market or EOSC initiative. It provides an interface to specify and select data assets and to map their metadata schema into a format understood by TRUSTS. The component communicates with the Data Exchange TRUSTS Component. Furthermore, it communicates with the Registry of Data markets and updates in case of relevant changes.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Third Party Data market</li> <li>• EOSC and its initiatives</li> <li>• Registry of Data markets</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Registry of Data markets</li> <li>• Data Exchange TRUSTS Component</li> </ul>
<b>Requirements addressed:</b> FR2, FR4, FR23, ARS4	

<b>C16</b>	<b>Registry of Data markets</b>
This component lists existing third party data markets and relevant initiatives of EOSC. On the one hand, this helps to form a community around TRUSTS. On the other hand, the component serves as an address book routing the communication between the Data Exchange TRUSTS Component and the Data Exchange Client Components installed on the premises of third party data markets and EOSC initiatives.	

<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Data Exchange Client Component</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Data Exchange TRUSTS Component</li> </ul>
<b>Requirements addressed:</b> FR2, FR4, ARS4	

<b>C17</b>	<b>Business Support Services</b>
This component will be used for monitoring and administration TRUSTS activities. It allows for the managing the list of connected organizations, monitoring of logs, and managing of fees collectable by the TRUSTS operator.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• smart contract executor</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Reverse Proxy</li> </ul>
<b>Requirements addressed:</b> FR33	

<b>C18</b>	<b>Broker + Metadata Storage</b>
The broker, and its accompanying metadata store, serves a central repository of all metadata regarding assets, participants and resources, as specified in the IDS-IM. It is a web application that serves as a wrapper of a Knowledge Graph that is compliant with the IDS-IM ontology. It collects metadata from the different corporate nodes and provides it to the smart contract executor and the platform interface in the User Portal Node (regarding assets), as well as to the Business Support Services (regarding organizations).	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Corporate Interface</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Smart contract executor</li> <li>• Business Support Services</li> </ul>
<b>Requirements addressed:</b>	

FR1, FR3, FR5, FR18, FR21, FR22, FR23, FR25, FR26,  
ARS11

C19	App Store
<p>Apps are docker images that participant organizations can, after signing a contract with the provider, pull and execute in their premises. The app store is a docker registry that is coupled with the TRUSTS platform to ensure that (push or pull) operations are compliant with a contract.</p>	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• None</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Trusted Connector</li> </ul>
<b>Requirements addressed:</b> FR19, FR24, FR38, FR39, FR40, FR41, FR42, FR43, ARS12	

C20	Identity Provider + Key Distribution System
<p>This component should provide users means to be a member of TRUSTS Ecosystem in a secure way. It will issue cryptographically signed authentication tokens. It should also distribute the public keys to verify the authenticity of said tokens.</p>	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Platform Interface</li> <li>• Corporate Interface</li> <li>• Usage Control</li> <li>• Business Support Services</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Usage Control</li> <li>• Platform Interface</li> <li>• Corporate Interface</li> </ul>
<b>Requirements addressed:</b> FR29, FR36, FR37, FR38, FR43, FR44, ARS16	

<b>C21</b>	<b>Vocabulary Services</b>
<p>All assets, organizations and components which are part of the TRUSTS platform is described as much as possible in terms of controlled vocabularies. These include, for example, taxonomies of keywords, types of assets, types of components, etc. These are maintained in this central location which provides other components with access to the list and definition of the terms in the vocabularies.</p>	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>● Reverse Proxy</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>● Mapping Builder</li> <li>● Corporate Interface</li> <li>● Platform Interface</li> <li>● Broker</li> </ul>
<b>Requirements addressed:</b> FR19, FR20, FR21, FR22, FR24, FR25, ARS3	

<b>C22</b>	<b>Dynamic Attribute Provisioning System (DAPS)</b>
<p>This is a component to support secure communication, authentication and authorization in TRUSTS infrastructure.</p>	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>● Trusted Connector</li> <li>● Automated Certificate Management Environment (ACME)</li> <li>● Business Support Services</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>● Trusted Connector</li> </ul>
<b>Requirements addressed:</b> FR38, FR43, FR44 ARS16	

<b>C23</b>	<b>Automated Certificate Management Environment (ACME)</b>
This component enables automation for management of certificates in the TRUSTS infrastructure.	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Dynamic Attribute Provisioning System (DAPS)</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Dynamic Attribute Provisioning System (DAPS)</li> <li>• Trusted Connector</li> </ul>
<b>Requirements addressed:</b> FR38, FR43, FR44 ARS16	

<b>C24</b>	<b>Smart Contract Executor</b>
<p>When access to a dataset is registered in a node, this action can result in triggering smart contract rules. The Smart Contract Executor is the component responsible for receiving information about this transaction, entering it into an appropriate logging mechanism, and performing the operations specified by the smart contract. The smart contracts can interact with other components related to security and privacy to check the ability of the consumer to get/use a dataset/service from a provider. The result then is documented with a logging mechanism and returned to the party which triggered the smart contract. This component stores smart contracts which have customizable input parameters which can be derived from (1) asset metadata, (2) the offering, and (3) the contract participants to enable a given contract's reuse for different transactions. A standard human readable description is attached to the smart contract.</p>	
<b>Input dependencies:</b> <ul style="list-style-type: none"> <li>• Business Support Services</li> <li>• Usage Control</li> <li>• Smart Contract Executor</li> </ul>	<b>Output dependencies:</b> <ul style="list-style-type: none"> <li>• Usage control</li> </ul>
<b>Requirements addressed:</b> FR10, FR11, FR12, FR13, FR14, FR15, FR16, FR17, FR30, FR38, FR43, ARS1, ARS2	

In summary, the following components are considered part of the TRUSTS technical architecture:

- C1 - Trusted Connector
- C2 - Dataflow Router
- C3 - Reverse Proxy
- C4 - Recommender
- C5 - Platform Interface
- C6 - Landing Page
- C7 - Asset Consumer
- C8 - Notification Service
- C9 - Metadata Mapper
- C10 - Usage Control
- C11 - Mapping Builder
- C12 - Corporate Interface
- C13 - Services Consumer Adapter
- C14 - Data Exchange TRUSTS Component
- C15 - Data Exchange Client Component
- C16 - Registry of Data markets
- C17 - Business Support Services
- C18 - Broker + Metadata Storage
- C19 - App Store
- C20 - Identity Provider + Key Distribution System
- C21 - Vocabulary Services
- C22 - Dynamic Attribute Provisioning System (DAPS)
- C23 - Automated Certificate Management Environment (ACME)
- C24 - Smart Contract Execution

### 3.2.1 Summary of connections between functional requirements and components

In the following we provide an overview showing which components need to address which functional requirements:

Table 3: Summary of connections between functional requirements and components

ID of functional requirement (FR)	IDs of components, which address the requirements
FR 1	C5, C12, C18
FR 2	C14, C15, C16
FR 3	C5, C12, C18
FR 4	C5, C12, C14, C15, C16

FR 5	C5, C12, C18
FR 6	C4
FR 7	C4
FR 8	C4
FR 9	C19
FR 10	C24
FR 11	C24
FR 12	C24
FR 13	C24
FR 14	C24, C24
FR 15	C24
FR 16	C24
FR 17	C24
FR 18	C18
FR 19	C21
FR 20	C21
FR 21	C18, C21
FR 22	C18, C21



FR 23	C9, C11, C14, C15, C18
FR 24	C19, C21
FR 25	C5, C18, C21
FR 26	C18
FR 27	C1, C2, C5, C7, C12
FR 28	C1
FR 29	C5, C7, C10, C12, C20
FR 30	C5, C12, C24
FR 31	C1, C5, C12
FR 32	C5, C12
FR 33	C5, C11, C17
FR 34	C5, C12
FR 35	C8
FR 36	C5, C12, C20
FR 37	C5, C12, C20
FR 38	C1, C10, C19, C20, C22, C23, C24, C24
FR 39	C1, C19
FR 40	C1, C19

FR 41	C1, C19
FR 42	C1, C19
FR 43	C1, C19, C20, C22, C23, C24
FR 44	C5, C10, C12, C20, C22, C23

### 3.2.2 Summary of connections between architecture requirements and components

In the following we provide an overview showing which components address which architecture requirements:

Table 4: Summary of connections between architecture requirements and components

ID of Architecture requirement (AR/ARS)	IDs of components, which address the requirements
ARS 1	C24
ARS 2	C1, C24
ARS 3	C9, C11, C21
ARS 4	C14, C15, C16
ARS 5	C3, C5, C6, C8, C12
ARS 6	C13
ARS 7	C1
ARS 8	C1, C2
ARS 9	C2

ARS 10	C2, C13
ARS 11	C18
ARS 12	C1, C19
ARS 13	C7, C8
ARS 14	C7, C9
ARS 15	C4
ARS 16	C1, C2, C10, C20, C22, C23
ARS 17	C1
ARS 18	C1
ARS 19	C1
ARS 20	C1
ARS 21	C1
ARS 22	C2
ARS 23	C3, C5

## 4 Design considerations for the architecture of the TRUSTS platform

The technical architecture of the TRUSTS platform is influenced in three ways: (1) by the functional and (2) by the technical requirements, but also (3) by the vision of the technical experts in the project.

The functional requirements were collected mainly from experts inside and outside of the project, and are collected in deliverable D2.2 “Industry specific requirements analysis, definition of the vertical E2E data marketplace functionality and use cases definition I”. The technical requirements were collected from the technical experts in the project, and are listed in Section 3 “Technical Requirements for the architecture”.

The vision for the architecture is based on a consensus of the technical experts who are participating in the project. The vision is expressed in several design considerations for the technical architecture, which we will describe in this section.

In addition, the experts for business models and innovation in the project will collect business requirements, which will also shape the technical architecture and the technical implementation of the TRUSTS platform in the future.

In this section, we first will describe several functional requirements which have influenced the architecture as a whole. Then we describe functional requirements which are based on the use cases from work package 5. This is followed by a description of two unique selling points of the TRUSTS architecture: the ability to handle services as part of a data marketplace, and the ability to handle portable applications as part of a data marketplace. We then describe which components of the TRUSTS architecture enable the establishment of trust between participants of an instance of the TRUSTS platform. Finally, we close the section by giving an overview of how we are planning for the evolution of the architecture by employing an idea from agile development which is called “Minimum Viable Product (MVP)” [4, 5].

### 4.1 Functional requirements which are addressed by the architecture as a whole

The use of the components listed here could be summarized as follows: Each organization has a set of components which, along with those deployed centrally by the TRUSTS operator, ensure that the assets it is offering are accessed in a trusted manner and in accordance with the contract it assigns. Likewise, consuming assets that are provided in the TRUSTS platform requires interaction with a set of components, so that privacy, data ownership, and contractual requirements can be enforced.

Apart from the FRs that are addressed in individual components (as listed above), we highlight three that have been of special interest when defining the architecture proposed here.

**FR28: TRUSTS should be able to be deployed as a federation of distributed, interconnected and interoperable nodes.**

This FR is addressed by the creation of a set of standardized components that will be distributed to all participating organizations, as well as the adoption of a protocol for interconnection between them. In particular, the semantics of the IDSCPv2 protocol are enhanced in the proposed architecture by the activation of the C10 - Usage Control, and C2 - Dataflow Router components.

**FR5: The system should provide rich search mechanisms across all federated nodes for available datasets and services**

This FR is addressed by having a centralized metadata repository which can be queried from different nodes. The metadata about assets is transmitted to the central node, but the assets themselves remain fully in possession of the providing organization.

**FR11: The system should ensure the integrity and authenticity of the smart contracts transactions signed by its users**

This FR is central to the concept of *trust* as understood in this project. Every operation in the TRUSTS platform, onboarding an asset, acquiring one, accessing one, etc., is performed via a sequence of messages as specified by the IDSCPv2. Each of these messages, which are emitted and received by respective Trusted Connector instances, is

- i) cryptographically signed with an X.509 certificate in control of the message producer and issued by a centralized certificate authority.
- ii) accompanied with a signed, time-sensitive, token certifying the authentication and authorization of the user who triggered the message (as provided by component C20 - Identity Provider + Key Distribution System)
- iii) recorded by the centralized Smart contract executor which shall, at any time, be queried by the interested parties to verify the current status of a contract or the sequence of operations pertaining to an asset they control.

Furthermore, there are functional requirements that have been collected from further specification of the use-case scenarios treated in WP5. These are summarized below, and are addressed by the way that services and applications are traded and used in the TRUSTS platform, which is described in detail in the next section.

**4.1.1 Functional requirements coming from the preparation for the use case trials**

The three use cases that are to be executed in the frame of the TRUSTS project, as specified in Deliverable D5.1, will make use of several features of the architecture proposed in this document. All three use cases involve a provider and a consumer of an asset, be it an application or a service. In all three, privately-owned and potentially sensitive data is utilized, necessitating a careful choice of where different software components are to be executed, and which parties are to be part of the processing.

There are several components which are to be deployed as stand-alone applications to be executed on the consumer's premises, and which come accompanied with all necessary data. There are the AML Risk Assessment Application (UC1), AML Transaction Monitoring Application (UC1), Insights for Banks Application (UC2) and Deanonymization Risk Application (UC2). The fact that they are to be deployed in the premises of the consumer, follows from the fact that they all take as input private or sensitive data, which the different consumers hold the right to, but are not necessarily allowed to transmit. In these cases, what the TRUSTS platform provides is the ability to deploy applications easily, entering and enforcing contractual agreements for their use, and making these applications easily discoverable by different consumers. Finally, the possibility to certify the functioning of applications is technically possible, and would remain an option for any future TRUSTS operator.

Three further UC-specific components are to be deployed as services which are to be executed on the provider's premises, and accessed in a controlled and metered manner by the consumer. These are the AML Screening Service (UC1), the AroTRON NBA service (UC3) and the AroTRON Chatbot Model service (UC3). The first one makes use of data for which the provider has a license allowing use but not redistribution, which necessitates that it remains at all times in their premises. The other two are computationally demanding services which are executed only once (or a few times) by the consumer, who is primarily interested in the output of the service.

Finally, an interesting combination of application, service, and dataset is to be deployed as part of the PSI Application by UC2. In this case, three parties are involved: one providing an application, one providing a dataset, and one consuming a service based on the dataset. For reasons of privacy (as described in detail in Deliverable 5.1), the parties involved in dataset exchange must not have access to the totality of the other's data. In brief, this use-case involves five steps: 1) the offering of an application in the TRUSTS platform; 2) the acquisition of this application by the dataset provider; 3) the creation of a service using this application and the dataset by the dataset provider; 4) the acquisition of the application by the dataset consumer; 5) the consumption of the service created in step 3 by the dataset provider by means of the application previously acquired.

#### **4.1.2 Handling of portable applications on the TRUSTS Platform**

Applications are software components which will be executed in the premises of the consumer, without any data leaving their network. Their installation and execution by the consumer may be the subject of contractual agreements, and the TRUSTS platform will provide technical means to enforce some of their provisions.

Application providers will onboard their applications by means of the Corporate Interface in their node. The operation of onboarding involves providing the metadata for the application, the necessary files for its configuration and deployment, a machine-readable description of its interfaces, and the directions to a locally (within the provider's node) available application image. The Corporate Interface will create the appropriate bundle file and advertise its metadata to the Central Node, which shall include it into the catalog.

When a consumer searches in their own Corporate Interface or in the Platform Interface, they will see a description of the application and, upon entering a contractual agreement, will be able to download, on their premises, the corresponding bundle. Then, this bundle will be decompressed and the application image will be,

subject to verification of identity and contract, pulled from the appropriate image repository into the consumer's node. The messages mediating this pull operation will be inspected and acted upon by the provider's Usage Control component, which will in turn consult the central Smart contract executor and, optionally, execute any relevant smart contract operations in the Smart Contract Executor deployed there.

Once the application image has been pulled, the configuration and deployment instructions contained in the bundle will be executed and the application will be made available from within the consumer's corporate network, to be used either directly by human users or by other, pre-existing applications.

Optionally, if the application exposes an HTTP interface adequately described (see below), the provider can choose to make their application accessible only through the Services Consumer Adaptor deployed in the consumer's node. This would enable a fine-grained metering of application usage, and execution of smart contract operations upon every request.

#### **4.1.3 Handling of services on the TRUSTS platform**

Services are software components which are executed on the provider's premises. The access to these services by the consumer may be the subject of contractual agreements, and the TRUSTS platform will provide technical means to enforce some of their provisions.

Services will be served by components installed by the provider according to their respective technical guidelines. This will allow, for example, pre-existing applications already deployed to be made available through TRUSTS, regardless of the operating system on which they are deployed, as long as they provide HTTP interfaces.

The service provider will also be responsible for providing a machine readable description of the service (for example, using the OpenAPI 3 specification). This file, along with the metadata necessary to describe the service (e.g. name, description, example input), will constitute a bundle which will be assembled by the providers Corporate Interface as part of the onboarding process. The metadata will be then forwarded to the central metadata repository, which will allow it to be discovered by other nodes or through the Platform Interface.

When a consumer acquires the service, they will receive in their corporate node the corresponding bundle. This bundle will be processed by the Dataflow Router in order to make the service accessible via the TRUSTS platform.

A user in the consumer's network will then access the different endpoints of the service by making HTTP requests to the Services Consumer Adaptor deployed in their node, attaching to each of them an authentication token. These requests can, if the business model so requires, be logged into the Smart contract executor from this point. The requests will be forwarded to the provider's node, where the Usage Control component will check the accompanying token for authorization, possibly consulting with the Smart contract executor about the status of the contract. If deemed adequate, the request will be forwarded to the component in the Provider's infrastructure which is actually serving the service. The resulting access will be

notified to the Smart Contract Executor, in order to trigger any action specified in the corresponding smart contract.

## 4.2 The role of the components in enabling trust between participants

The Trusted Connector component allows for verification of the origin of any messages received. This means that the provider of an asset can be sure that any request they receive comes from within the TRUSTS platform and, furthermore, can reliably identify the originating node and the user within. With the architecture presented here, a provider can use these assurances to evaluate, on their premises, the contractual validity of a request. In particular:

1. Application providers will transfer applications only after the requesting messages have been validated by the smart contract executor. Thus, the provider can rest assured that only contractually-valid transfers of their applications take place.
2. If the application is to be made accessible only through the consumer's Services Consumer Adaptor (only possible in the case of applications providing adequately described HTTP interfaces), then the application provider can request from the smart contract executor a detailed log of all access to the application. Importantly, the application provider must develop in their application the security mechanism to ensure that only requests which are accompanied by an adequately signed authorization token are served. In this case, it is possible to utilize fine-grained contracts on an application, for example, based on the number of requests.
3. Services exposed by a provider will be configured to only accept requests being forwarded to them by the node's Trusted Connector. In this case, the node's Usage Control component will evaluate the request against information in the smart contract executor. Thus, the provider can rest assured that only contractually-valid requests are served.
4. The requests done to a service are logged also by the consumer's node (in particular, by the Services Consumer Adaptor). This disallows incorrect claims by the provider regarding access to the service, as the smart contract executor holds records from both parties regarding access operations.
5. Dataset providers, likewise, can rest assured that all requests served are contractually valid, since only then will they be forwarded to any serving component (e.g., the Corporate Interface).
6. All signatures and tokens that have been described will be forwarded to the destination component, thus enabling any provider to check and log transactions on their own terms. Likewise, all certificates used to issue these signatures and tokens will be known to all participants, as part of the organization onboarding process.

Importantly, the metadata of all assets will include signatures (hashes) of all traded assets (application images, service descriptions, datasets), which will allow the consumer to rest assured that the received asset has not been tampered with. These signatures can, furthermore, be compared directly with those included in the provider's node catalog, thus reducing the need for trusts in the TRUSTS operator.



### 4.3 Planning for the evolution of the architecture with agile methods

The architecture of the TRUSTS platform represents the blueprint for the results of the TRUSTS project. As such, it also represents the foundation for instances of the TRUSTS platform which will be provided by one or more TRUSTS operators after the duration of the project.

The architecture has to accommodate the requirements and priorities of many different stakeholders. As such, it is prudent to plan for the evolution of the architecture during the project. In the TRUSTS project, we will use agile methods to facilitate multiple iterations of the architecture. In particular, we will use the concept of a “Minimum Viable Product (MVP)” to involve relevant stakeholders in the evolution of the platform, at different points in time during the project.

A Minimum Viable Product [4, 5] can be thought of as a version of a product with just enough features to be usable by early customers who can then provide feedback for future product development.

In TRUSTS, each MVP version should be seen as a clearly specified hand-off point, at which a specific combination of platform components in specific versions is released to all partners who are working on the implementation of use cases and on the implementation of privacy enhancing technologies. These partners are the internal customers of the MVP within the TRUSTS project.

Every MVP version will be released to the internal customers together with a description of how to perform technical tests. The results of these technical tests will then be used as input for the next version of the MVP.

The goals for the MVPs are characterized by four dimensions: integration of APIs, functional integration, operational integration and quality control.

#### **Integration of Application Programming Interfaces (APIs):**

So-called “software mock objects” can be used to implement and test APIs in isolation, independent of the progress of the implementation of the API. This can be used to distribute the work on components which will be connected to the APIs, and for integrating the APIs of existing software with newly developed software. This dimension is also important for validating the interplay between components which are run on different hosts.

The testing of APIs will result in a better understanding of the component view on all software artifacts in the project and will enable better communication between the technical experts in the project. In addition, it allows for testing of essential and shared infrastructure software components which are reused from either the IDS or DMA. Finally, it will allow for testing of deployment mechanisms.

#### **Functional integration:**

The testing of functional integration is concerned with the details of the interplay between components on the level of data and functions. Instead of testing APIs, functional testing is concerned with testing if components

can correctly handle incoming data and return the correct kind of output data. In addition, any side effects or error conditions which can appear, have to be accounted for.

The testing of functional integration benefits from a complete documentation of the data models used by all data stores in the project. It is also desirable to connect all implemented functions to realistic data sources. In addition, functional integration is very important to confirm that components developed by different project partners are able to interact in a correct way, and that the handling of errors in the input data allows resilient operation of the platform components.

### **Operational integration:**

The testing of operational integration is concerned with providing functionality in a way which simulates the envisaged customers of the platform. A prerequisite of this is that the platform components are able to communicate with each other via the correct APIs, and that functions of components are able to handle input and output data correctly and to be resilient towards errors. Operational integration is also concerned with the user experience and the way in which the overall functionality of the platform is communicated and packaged to the user.

As a result of operational integration, all aspects of the platform which concern the TRUSTS operator during and after the project will be tested. In addition, the security mechanisms of the platform, including its authentication services and the usage control mechanisms will be tested. Finally, operational integration allows non-technical experts to confirm cross-cutting requirements, such as those stemming from decisions about business models and innovation specifications.

### **Quality control:**

This dimension of the MVPs is concerned with ensuring that all artifacts (i.e., software components, data models, user interface, documentation) of the platform are mature and of a high quality level. This dimension is concerned with how an artifact is perceived by a potential end user. Important considerations include the usefulness of documentation and the design of the different user interfaces of the platform. However, it also includes factors of how the platform is perceived which have a technical aspect. In particular, how the speed and throughput of the platform is perceived. In addition, high level questions, such as the usefulness of the platform for common tasks of potential end users should be considered.

## 5 Conclusions and Next Actions

This deliverable summarizes the activities in task 2.4. “Architecture design and technical specifications” and is the first of two versions of this deliverable. It documents the blueprint for the technical results of the TRUSTS project. The technical specifications provide the details which are required by technical experts in order to instantiate the platform infrastructure and built on top of it or to extend it with their own components, services and applications.

The architecture of the TRUSTS platform has to accommodate the requirements and priorities of many different stakeholders inside of the project. In order to accommodate this, we collected architectural requirements from relevant parties. First, we collected requirements for the architecture from the initiatives on which the TRUSTS platform is based, which are the Data Market Austria (DMA), International Data Spaces (IDS). In addition, we collected architectural requirements from the GAIA-X initiative, as future compatibility with Gaia-X is of strategic importance to the TRUSTS platform. We expect GAIA-X to set important impulses for the data economy in Europa by e.g., communicating with important groups of stakeholders to set the agenda and by setting standards for technical and organizational issues, such as certification.

Then we collected architectural requirements from the different technical participants of the project, grouped by areas of concern. We collected such requirements related to: smart contracts; interoperability of data marketplaces; data governance; platform development and integration; brokerage and profiles for users and corporates; privacy enhancing technologies; anonymization and de-anonymization; as well as from the usage of CKAN software. We collected a total of 57 architectural requirements, which we grouped into 23 architecture requirement summaries.

Based on the architectural requirements, we describe a software architecture. We give an overview of the architecture, then we specify which software components are needed in order to address the collected functional and architectural requirements. Our proposed architecture contains 24 components. We provide two tables which show how the components are connected to the functional requirements and the architecture requirement summaries.

Finally, we describe the design considerations of the architecture to document the aspects of the architecture which are represented in the interplay of multiple components, instead of being implemented in a single component. These aspects include: the enabling of trust between participants using the platform; the functionality related to the use case trials; the handling of both portable applications and services on the TRUSTS platform; and the planning for the evolution of the architecture with agile methods.

For the second version of this deliverable, the focus will be on iterating the architecture based on feedback from the use case partners and from the non-technical project partners. Together with the use case partners, the technical environment for testing the use case trials will be defined in multiple, iterative versions. Complementary to that, the technical impact of the requirements identified in the area of business models for data marketplaces and the legal requirements will be investigated and will be addressed, if necessary, through updates of the architecture. In addition, the project partners with a technical perspective will refine the architecture and document additional technical specifications based on their implementation experiences. The results of this task will also provide input for selecting the results of the project, which will be used as starting points for standardization efforts with relevant external stakeholder organizations, by the standardization experts in the project.

## 6 References

- [1] M. Traub, et al., "Broker and Assessment Technology Specification and Development Road", Data Market Austria, May 2017.
  
- [2] B. Otto, et al., "Reference Architecture Model Version 3.0", International Data Spaces Association, April 2019.
  
- [3] G. Eggers, et al. "GAIA-X Technical Architecture", Federal Ministry for Economic Affairs and Energy (BMWi), June 2020. Accessed via: [https://www.data-infrastructure.eu/GAIA/Redaktion/EN/Publications/gaia-x-technical-architecture.pdf?\\_\\_blob=publicationFile&v=5](https://www.data-infrastructure.eu/GAIA/Redaktion/EN/Publications/gaia-x-technical-architecture.pdf?__blob=publicationFile&v=5)
  
- [4] V. Lenarduzzi, D. Taibi, "MVP Explained: A Systematic Mapping Study on the Definitions of Minimal Viable Product", Euromicro SEAA, 2016.
  
- [5] J. Münch, et al. "Creating minimum viable products in industry-academia collaborations." International Conference on Lean Enterprise Software and Systems. Springer, Berlin, Heidelberg, 2013.

## 7 Annex: Smart Contract FR revision

Smart contracts are one of the areas of concern for the TRUSTS architecture. The work in relation to this concern is being led by Fraunhofer ISST. The main goal with regards to this concern is to develop the necessary concepts for using smart contracts in the context of the European Data Market delivered by the TRUSTS project. This includes in particular the technical foundations for ensuring the integrity and authenticity of such contracts as well as the analysis of the legal challenges brought about by smart contracts, such as issues of validity, enforceability, and interpretation. Related to this, the technical challenges and the legal issues regarding the fact that smart contracts are written in executable code instead of natural language, will be examined by the partners with legal expertise in the project.

The result of this work is a comprehensive concept document related to all the aspects of the above paragraph. Additionally, a demonstrator is being developed by Dell. The demonstrator consists of a blockchain instance which is using the Hyperledger Fabric technology. Thus it is possible to execute smart contracts in this instance. The demonstrator will also connect to external payment systems. The demonstrator is logged as architecture component C24 - “smart contract executor”.

Based on the project work of the partners with expertise related to smart contracts, the following suggestions for revising the related functional requirements (FRs) in the future are made. These are the FRs which are grouped under the label of “Purchasing and Billing”. The next version of the deliverable which collects the functional requirements is D2.3 “Industry specific requirements analysis, definition of the vertical E2E data marketplace functionality and use cases definition II”.

### Revision suggestions for the functional requirements grouped as “Purchasing and Billing”:

- FR10: “The system should provide smart contract mechanisms as a validation means of sellers/buyers agreements.”
  - No change is suggested.
- FR11: “The system should ensure the integrity and authenticity of the smart contracts signed by its users.”
  - Change to: “The system should ensure the integrity and authenticity of the smart contract transactions signed by its users”.
  - Explanation: The reformulation of FR11 is to increase semantic accuracy by encompassing the transaction operation.
- FR12: “The system should provide a human friendly representation of smart contracts (e.g., natural language).”
  - Change to: “Smart contracts in the system should be accompanied by a human friendly representation (i.e natural language).”
  - Explanation: The current description could imply the automatic derivation of human-readable text from executable smart contract code.
- FR13: “Signed smart contracts should be legally valid, enforceable and interpretable.”
  - Change to: “Mechanisms to make signed smart contracts be legally valid, enforceable and interpretable will be investigated”.
  - Explanation: The current form seems infeasible due to the necessary procurement of legal contracts to underpin the smart contracts.

- FR14: “The system should encompass mechanisms for keeping transactions performed ensuring that they cannot be infringed.”
  - Change to: “The system should encompass mechanisms for keeping performed transactions from being infringed”
  - Explanation: The new version rearranges the words to clarify the meaning of the sentence.
- FR15: “The system should provide billing mechanisms for enabling consumers to pay providers according to the agreed smart contract.”
  - Change to: “The system should provide the ability to connect to billing mechanisms for enabling consumers to pay providers according to the agreed smart contract.”
  - Explanation: The current wording suggests that TRUSTS will provide a component which implements a payment mechanism. Instead, as reflected by the new wording, TRUSTS will provide a component to connect to existing payment mechanisms.
- FR16: “The system must provide alternative and flexible pricing models taking into consideration the diversity of the available datasets and services.”
  - No change is suggested. However, the fulfillment of this functional requirement requires the expertise from most partners in the project and goes beyond expertise in the area of smart contracts.
- FR17: “The system should provide brokerage mechanisms for addressing the offerings and demands of the hosted datasets and services.”
  - No change is suggested. However, the fulfillment of this functional requirement requires the expertise from most partners in the project and goes beyond expertise in the area of smart contracts.