The BRIGHT project is co-founded by the EU's Horizon 2020 innovation programme under grant agreement No 957816

Boosting DR through increased communIty-level consumer engaGement by combining Data-driven and blockcHain technology Tools with social science approaches and multi-value service design

Deliverable D4.1 Scalable Privacy Preserving Data Collection

Authors: Andrej Čampa (COM), Klemen Bregar (COM), Denisa Ziu (ENG), Andrea Iannone (CEL)

The project Boosting DR through increased communIty-level consumer engaGement by combining Data-driven and blockcHain technology Tools with social science approaches and multi-value service design (BRIGHT) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957816. The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the Innovation and Networks Executive Agency (INEA) or the European Commission (EC). INEA or the EC are not responsible for any use that may be made of the information contained therein.

Imprint

Title:	**Scalable Privacy Preserving Data Collection**
Contractual Date of Delivery to the EC:	31.10.2021
Actual Date of Delivery to the EC:	29.10.2021
Authors:	Andrej Čampa (COM), Klemen Bregar (COM), Denisa Ziu (ENG)
Participant(s):	COM, ENG, DuCoop, CEL
Project:	Boosting DR through increased community-level consumer engagement by combining Data-driven and blockchain technology Tools with social science approaches and multi-value service design (BRIGHT)
Work Package:	WP4 – Community and Customer Digital Twin Models
Task:	T4.1 – Scalable privacy preserving Data Collection
Confidentiality:	Public
Version:	1.0.0

Consortium - List of partners

Table 1. Consortium partner list

Partner no.	*Short name*	*Name*	*Country*
1	ENG	ENGINEERING - INGEGNERIA INFORMATICA SPA	Italy
2	TUC	UNIVERSITATEA TEHNICA CLUJ-NAPOCA	Romania
3	IMEC	INTERUNIVERSITAIR MICRO-ELECTRONICA CENTRUM	Belgium
4	COM	COMSENSUS, KOMUNIKACIJE IN SENZORIKA, DOO	Slovenia
5	SONCE	SONCE energija d. o. o.	Slovenia
6	ISKRA	ISKRAEMECO, MERJENJE IN UPRAVLJANJE ENERGIJE, D.D.	Slovenia
7	EMOT	EMOTION SRL	Italy
8	TNO	NEDERLANDSE ORGANISATIE VOOR TOEGEPAST NATUURWETENSCHAPPELIJK ONDERZOEK	Netherlands
9	CENTRICA	CENTRICA BUSINESS SOLUTIONS BELGIUM	Belgium
10	ASM	ASM TERNI SPA	Italy
11	DuCoop	DUCOOP	Belgium
12	CEL	CYBERETHICS LAB SRLS	Italy
13	DOMX	DOMX IDIOTIKI KEFALAIOUCHIKI ETAIREIA	Greece
14	APC	Asociatia Pro Consumatori	Romania
15	WVT	WATT AND VOLT ANONIMI ETAIRIA EKMETALLEYSIS ENALLAKTIKON MORFON ENERGEIAS	Greece
16	SUN	SunContract OÜ	Estonia

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Table 2. List of Acronyms and Abbreviations

AAL	Ambient Assisted Living
API	Application Programming Interface
B-DLT	Bright Distributed Ledger Technology
BRIGHT	Boosting DR through increased community-level consumer engagement by combining Data-driven and blockchain technology Tools with social science approaches and multi-value service design
CPU	Central Processing Unit
DLT	Distributed Ledger Technology
DR	Demand Response
EMS	Energy Management System
EV	Electric Vehicle
HTTPS	HyperText Transfer Protocol Secure
HVAC	Heating, Ventilation, and Air Conditioning
ID	Identifier
IoT	Internet of Things
JSON	JavaScript Object Notation
PV	Photovoltaic
REST	Representational State Transfer
SOC	State of Charge
URL	Uniform Resource Locator
VEC	Virtual Energy Community
QoS	Quality of Service
SOA	Service-Oriented Architecture
ZKP	Zero-Knowledge Proof

Executive Summary

The objective of this deliverable is to describe the initial Bright system architecture for data collection and data privacy preservation. Data sources in Bright are heterogeneous and produce great amounts of sensitive data that needs to be preserved and kept private.

The deliverable extends the contributions of previous tasks from WP2, especially the T2.3 in the form of *D2.4 Cross-Domain Data & Service Interoperability*, where the interoperability layer for use in Bright is conceptually defined. The proposed system architecture is based on currently known data sources and components of pilots. This document reports the initial testing of the interoperability components and performs the first experiment accessing the historical and live pilot data. The final system architecture for scalable privacy-preserving data collection will be reported and presented in an updated version of the current document scheduled for M30.

The current version of the document presents a scalable service interoperability layer that is a base for the Bright system and works as a universal connector between different data producers and consumers inside the Bright consortium. Data acquisition processes are defined by introducing currently known data objects and a data privacy preservation strategy is defined to secure all communications between the agents in the Bright system. An example of data acquisition and access is given by a specific Bright pilot with available historical data in Section 4.2.

1 Introduction

The report presents the first step and actions taken to integrate data from heterogeneous sources into a Bright ecosystem. This report analyses the proposed solution for creating an interoperable solution that is scalable and allows privacy protection and integration of existing data sources.

1.1 Scope of the document

This deliverable *D4.1 Data Collection* is the first of the two reports for task 4.1. The first report is mainly concerned with the evaluation of the possible solutions to be integrated into the Bright architecture. The first report is reported in M12, while the second report is scheduled for M30. The second report will present the full solution that will be integrated. The complete solution will enable:

- Accessing existing data sets from all Pilots
- Providing access to historical data and live data for the creation of various services
- Retrofitting legacy data

This document uses the output of *D2.3 DR technologies and tools specification* where the requirement and specification of tools are defined. It also includes the Bright architecture that needs to be followed by all partners. Furthermore, this document also uses *D2.4 Cross-domain Data & Service Interoperability* output, where the interoperability layer is defined in detail. Finally, the evaluation of particular solutions is presented in this document that is aligned with the interoperability vision of the Bright project solution.

The output of this document is the evaluation of the technologies used to create the interoperability layer, in particular the Kafka streaming engine and the access to the existing data from Pilot 2 to create new models within the Bright project (WP4, WP5 and WP6) from different partners.

1.2 Structure

Deliverable is structured in the following sections:

- Section 2 describes the Bright platform, data acquisition process, the status of data models and the first version of identified data objects relevant for this deliverable.
- Section 3 presents the message queue system that will be used in the final Bright architecture to create scalable and privacy-enabled solutions for streaming data among different stakeholders. The Kafka streaming engine is evaluated as one of the possible solutions.
- Section 4 presents the solution to access the Pilot 2 live and historical data, which will be used in developing various models in WP4, WP5 and WP6.
- Section 5 wraps the report with conclusions resulting from sections 2, 3 and 4 and outlines the next steps required towards the second version of this document in M30.

2 Bright platform

The key aspect of the Bright platform is data interoperability, which enables the exchange of data between different components and actors within the platform by mapping various data sources to predefined data models. Data models by the semantics written in their structures enable seamless data mapping from many different data sources to the common data model and thus common understanding of the data received from the data source.

The key component of any service-oriented architecture (SOA) is a service compatibility layer that translates messages from each component in the SOA architecture and translates it to a message that the target service on the other side of the SOA understands it. The main functions of a service compatibility layer are:

- Message routing
- Message monitoring and control
- Resolving communication disputes between services
- Service provisioning and versioning control
- Quality of service (QoS) enforcement

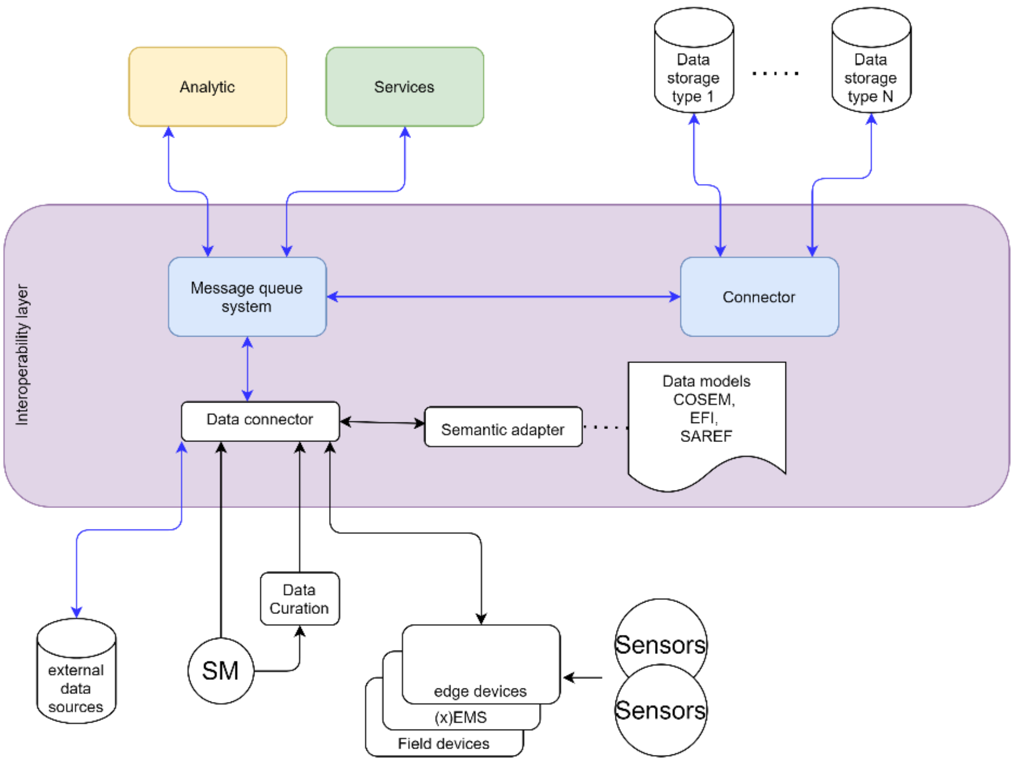


Figure 1: BRIGHT system interoperability architecture.

A conceptual architecture of a Bright interoperability layer is shown in Figure 1 where only the main logical components and interactions are depicted. It is responsible for data collection and management, semantic adaptation and data integration. The current development stage of the

Bright platform still doesn't enable us to cement the final architecture of the platform that will be used in the pilots. Current version of architecture of BRIGHT interoperability layer contains message queue system which is currently implemented by Apache Kafka. Another component is a group of data connectors that take care of translating the data between the historical data inside the databases and data streams inside the message queue system. Data interfaces to the data producers are implemented inside the data connector component which together with a semantic adapter and data models provide needed data translation and thus data interoperability inside the BRIGHT. Due to the implementation constraints during the development of the pilots and platform, the components and specifications will still slightly change and will be reported in the later deliverables (*D2.5 Cross-domain Data & Service Interoperability* in M19 and in the second version of this document *D4.1 Data collection* in M30).

2.1 Data Acquisition Process

Data collection starts with the measurement and acquisition devices, which are mostly connected to the edge devices such as home gateways, EMS and others. Those devices capable of data preparation and reporting can connect to the Bright interoperability layer and post the measurements in a predefined data format that is understood by the data connector and semantic adapter in the interoperability layer itself. The data models that are already defined inside the Bright platform and others that will be defined during the development and integration of the pilots and Bright platform, enable the seamless translation between the machine-specific messages to the universal ontologies that can be understood throughout the Bright platform.

2.1.1 Data models

Data models as conceptual, and in many cases, visual tools are used to describe the data elements and their relations inside the actual physical system. Thus, a data model is intended to represent an entity from the physical world and can be used to enable interoperability between different components in the information systems. The main objectives of using data models are:

- Accurate representation of data objects
- Definition of relations between the components of a data structure
- Provide a clear picture of the data
- Help in identifying missing data
- It unifies the IT infrastructure and makes it upgradeable and maintainable.

2.1.2 Identified Data Objects

Through collaboration between experts from the various stakeholders in the Bright consortium, we have created a list of data objects to be used in the proposed pilots and in the implementation of the Bright. Most of these data objects are conceptual representations of the devices identified in the pilots. These identified data objects are listed in Table 3. , where the conceptual structures of the data objects are represented by lists of device/object attributes.

Table 3. List of identified objects and list of attributes.

Object	Description	Attributes
Smart meter	Smart meter device, also advanced metering infrastructure (AMI) and any legacy electric energy meter	- ID - Timestamp - Active Energy/power - Voltage - Current - Tariff
Energy meter	Any other metering device that is not utility meter used by DSO	- Consumed Energy (day, night, total) - Current - Frequency - Voltage - Active power - Reactive power - Power factor - Timestamp
Energy meter heat	Heat generation, e.g. heat pump	- Operational state - Generated power - Input Temp. - Output Temp. - Energy - Active power - Timestamp
Generation	Generation of energy, such as PV.	- ID - Measurement Unit - Timestamp - Active Energy Import - Reactive energy - Current - Voltage - Frequency - Power - Total yield
Storage	Energy storage, mostly battery systems	- Timestamp - SOC - Charge/discharge Power - Operational state
Heat Storage	Accumulation of the heat in the water heat storage	- Temperature - Min. Temperature - Max. Temperature - Volume - Timestamp
EV	Electric vehicle	- ID - Timestamp - SOC
EV charging station	Electric vehicle charging	- ID (station and socket)

	station	- Max. power - Min. power - Current - Voltage - Charging power - Price - Timestamp
Load	Flexible or any other types of loads (e.g. HVAC, heating devices)	- Consumed electric energy/power - Current - ID - Timestamp
Weather	Historical and forecasted weather for a location of interest	- Date - Time - Temperature - Precipitation - Humidity - Wind direction - Wind speed - Solar radiation - UV index
Calendar	Local/National calendar with all types of events (national holidays, other free days and bank holidays)	- Working days - Weekend - Public holidays and bank holidays
Property and measurements	e.g. indoor parameters (temperature, humidity, CO2)	- ID - Timestamp - Unit - Value
Smart Heating Controller	Remote monitoring of heating system parameters with the aid of smart heating controllers attached with the boilers of pilot users enable control and access of boiler's parameters	- Indoor/outdoor temperature, - Space heating temperature setpoint, - Domestic hot water temperature Setpoint, - Boiler water temperature, - DHW temperature, Heating usage, - Hot water usage, - Boiler modulation, - Gas consumption, - Timestamp
Private living unit	Mostly dwellings and office spaces	- Temperature - Temp. set point - District heating Temp. - District heat return Temp.

		- District heating flow - Tapping Temp.
BEMS	Building Energy Management system	- Zones Temperature - Humidity levels - Technical setpoints - Timestamp

2.2 Data privacy

Encrypted communications are used in all interactions between the Bright platform and components involving data exchange to ensure privacy at all levels of data transport from source to analytics services.

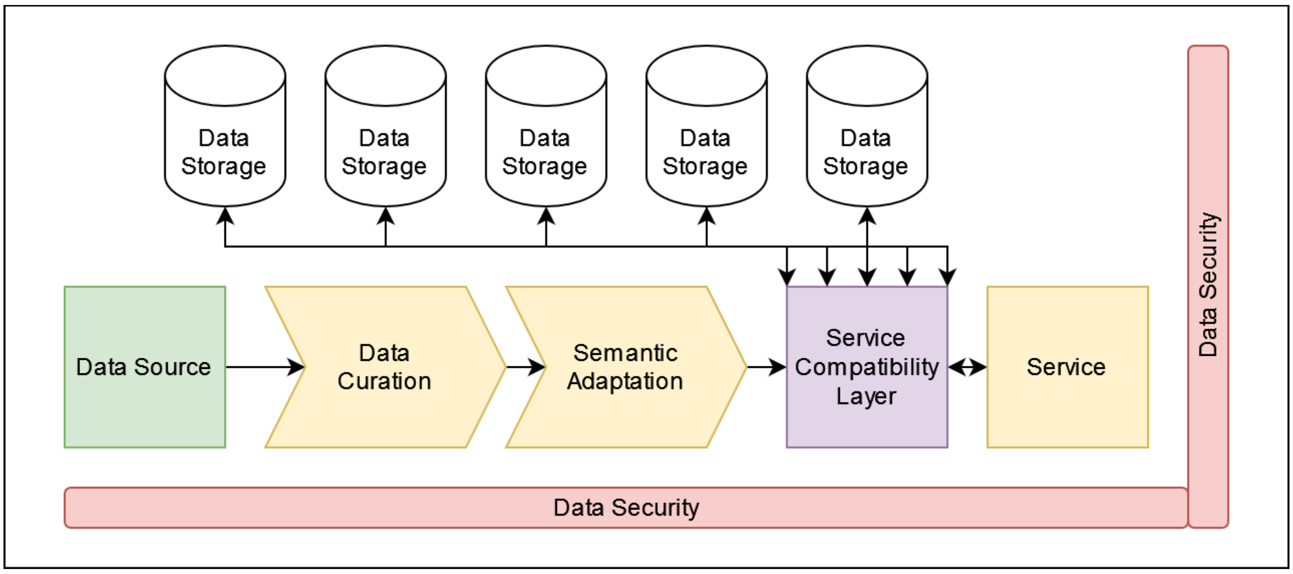


Figure 2: Data privacy chain in Bright.

In many cases, for example, when using MQTT as a communication interface, recommended authentication method is to use X.509 certificates in conjunction with Transport Layer Security (TLS) protocol to establish a secure encrypted communication channel between the client and the server. TLS cryptographic protocol use a handshake mechanism for different parameter negotiations. After the handshake is completed, the communication channel is secured and no third-party device or agent can eavesdrop on the communication anymore.

TLS protocol can be used on most of the communication channels between the Bright components, thus preventing unauthorized agents from accessing sensitive data and most importantly preventing unauthorized use of Bright platform resources (hardware and communication).

2.2.1 Privacy and security in Bright DLT solution for flexibility trading

Data privacy on the Bright DLT solution is guaranteed by storing any and all personal data off-chain, by the use of ZKPs, and by differential measuring. Indeed, data on chain is pseudo-anonymized: the address of the prosumer smart contract, which receives the ZKP, is on chain, while the physical address of the prosumer is not. What is shown on chain is not the actual energy usage, but the deviation from a baseline. Essentially, the ZKPs hide the monitored energy data and the requested flexibility profiles while registering on chain only the deviations. The ZK proof validation checks that the deviation is correctly computed. This entails that there is a minimum risk of linkability between the on-chain address and the actual prosumer in real life.

From a security perspective, the blockchain trading network might encourage one-off opportunistic behavior on the part of participants motivated by a high-enough prospective payoff. For instance, a malicious actor could potentially alter n users' flexibility baselines to artificially inflate the price of flexibility in order to sell at that higher price. To counter this risk, though price is considered equilibrium price of bids and offers submitted on chain at the end of each trading session, it is also compared to previous trading sessions. So, if there is a sudden, artificially inflated price, the system would detect it.

Lastly, the risks of stealing / tampering data prior to the storage on the blockchain is a well-known problem. Most proposed solutions are hardware-based. However, an analysis of these solutions has been considered to be out of scope for the Bright project.

3 Message queue system

Data acquisition is a critical aspect for Bright. Sensors, edge devices and external data sources are just some of the producers that can supply information to the system. Due to its nature, data that flows inside the system can be categorized as big data and, for this reason, a conventional way to collect it cannot be the best solution for the project's purposes. In order to address this limitation, communication inside of Bright is managed with the use of a message queue system. This type of tools not only provides an asynchronous and efficient way to collect a great amount of data, but also allows the other components to access it independently. To be able to share data, all applications or tools need to connect to the Bright's message queue system. Since communication between data producers and consumers is not implemented as a single point-to-point transition, scalability of each tool can be achieved in an easier way. The selected implementation for Bright's system is Apache Kafka [1].

3.1 Scalability and Privacy

Data sources' availability and efficiency are subject to many different factors. For example, a pilot might have access to a different set of sensors or devices and the number of those can even vary over time. This dynamism makes it hard to forecast the amount of information that flows inside of the system and, even after that, the value might not be consistent in time. The same argument applies to tools and other applications that need to access the data in order to offer specific services to the final user. For instance, a tool might not be available in the beginning but can be added later.

To make the system reliable and able to sustain the fluctuating amount of data, the Bright's communication mechanism (i.e., the message queue system) needs to be able to scale whenever necessary. One of the key and strong aspects of Kafka is precisely this. From the beginning, the creators of the platform focused on creating a scalable and efficient way to handle a huge quantity of messages. To achieve that, Kafka is designed to operate with a cluster of brokers, each one able to manage many topics. Those brokers can be spread over different servers and even regions or data centers, making Kafka able to scale over distributed machines. Performances can be improved by common approaches such as increasing CPUs or memory for a specific server, but other than that, scalability in Kafka can be achieved by [1,2]:

- Increasing the number of brokers: if the number of messages received by the system are too many and brokers can't keep up with this quantity, a possible approach is to add more brokers to the cluster. Kafka can handle more than one broker and they can be distributed over the network.

- Organizing topics: in Kafka, topics can be partitioned according to a system configuration. Each partition holds a set of messages and if needed, producers can insert them in a specific partition. For example, a partition might hold all the messages sent by the same application. Even if partitions refer to the same topic, they are distributed over different brokers. This optimization enables a system to dedicate a broker with a more powerful setup (e.g., more CPU, memory) to handle the more resource-consuming partitions.

- Increasing the number of consumers: if the message elaboration time is slowing down the system, the scalability of brokers will not help in this scenario. To improve the performances then, more consumers are needed. To help with that, Kafka provides an approach based on the concept of consumer groups. The platform enables the creation of a set of consumers and the possibility to assign specific topics to this group. For each of those topics, every message received that fits in it will be assigned to only one consumer. In other words, replicating the same consumer and assigning them to the same group will ensure that a received message is elaborated only once.

Another concerning aspect of the data transmission regards the privacy of the information that travels through the system.

Kafka cares about data security by implementing different policies and measures. Components inside the platform can interact with each other in a secure way by enabling authentication with SSL or SASL. Brokers and clients (both producers and consumers) can be set to use one or any of those authentication means as defined in the cluster configuration. Thanks to that, security can be shaped according to system or tool needs. Additionally, Kafka allows data encryption with SSL and enables authorization for specific actions (such as write or read) on every single client.

Other than that, thanks to the wide variety of Kafka's plugins, standard behaviour can be extended or modified to adapt to Bright's necessities. For example, another approach to data protection might be the introduction of a specific connector that can alter information before sending them to the actual consumers. Plugins such as Privitar [3] can apply a custom set of functions to each produced value to redact it, substitute with a predefined value or, in general, manipulate the data to protect it. Consumers will then receive the altered value and data privacy will be preserved when the information is accessed by the final user.

3.2 Implementation example

The sequence diagram shown in Figure 3 represents how data travels in Bright. Kafka has a central role in the schema because information will be sent to all systems connected to the corresponding topic, in a publish/subscribe approach. The steps are:

1. IoT devices, or other forms of data generation such as an external database, send data to the system. From a Kafka perspective, all those clients take the role of producer. The data is sent to a *Data Connector* by different means, depending on the implementation offered by every single connector.

2. The *Data Connector* receives data from outside of the system. To make Bright able to operate properly, data need to be normalized. Specifically, normalization is done with the support of an external component, called a *Semantic adapter,* that transforms data according to the chosen universal ontologies.

3. The *Semantic Adapter* converts information from the raw data generated by the producers to the system structure with the support of energy-specific standards and ontologies. This operation ensures that all the data that travels on the message queue system is normalized.

4. After receiving the data back from the *Semantic Adapter*, the *Data Connector* publishes a new event on the *Kafka server*, on a specific topic. Topics are created according to each pilot’s custom system configuration and can be based on the data type or available

services. Moreover, this operation might consider publishing data on specific partitions if it's a viable optimization strategy in the pilot's environment.

5. *Kafka server* now allows data to be pulled by the consumers; in this example, consumers are identified by *Services* (e.g., tools) and *Data storage* (e.g., common or shared databases). To be able to pull information, a consumer must be subscribed to the specific topic on which the data is published. Then, it will be able to retrieve the message and proceed with its elaboration.

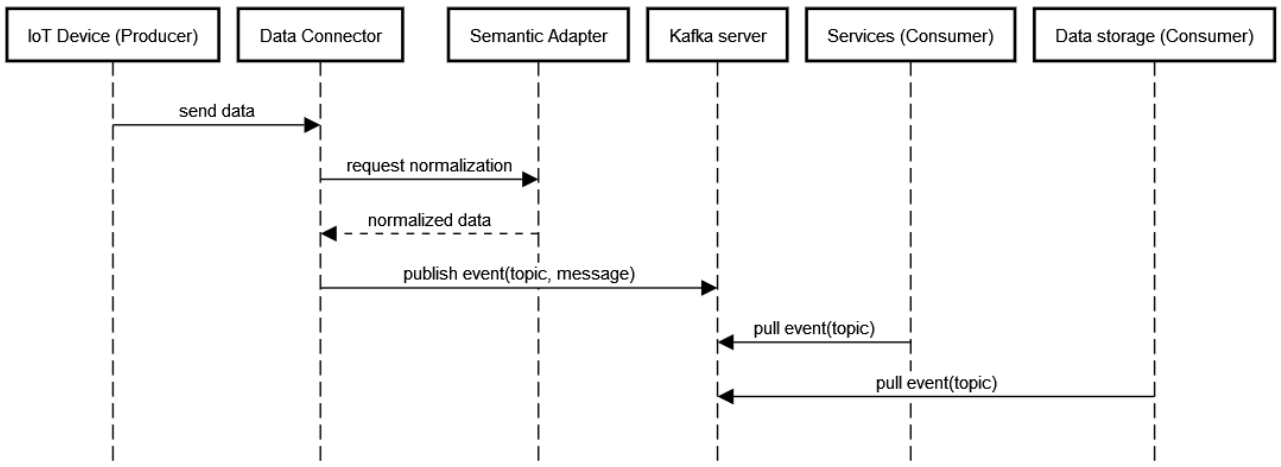


Figure 3: Data exchange in Bright.

4 Data access

For the needs of data analysis and services in the Bright project, the historical data and streamed live data are accessible through defined application programming interfaces (APIs) that are encrypted and authenticated to prevent eavesdropping by third parties. Each Pilot has come to the project with its own solution for external partners to access their data. Some data can be accessed online in different data formats and some of them can be for now accessed offline (e.g. historical data in CSV). This is more in detail described in D2.4. We will explain the data access on an example in Pilot 2 from Slovenia.

4.1 Introduction to the Pilot 2

The goal of the Slovenian pilot is to provide one-stop solutions for active consumers and citizens who participate in innovation and create value together with other actors (all together forming a virtual community). The solution of the DR program will be based not only on energy services, but also on non-energy services such as detection of inappropriate or sudden deviations in consumption of appliances or detection of environmental sensors outside the normal range in living spaces.

The specificity of the Slovenian pilot project is that it does not have strict geographical boundaries, as it takes place in a virtual community. A virtual community is formed by users of the platform: consumers, prosumers and indirectly involved citizens. The awareness of the variable energy price differs from one user group to another, which is reflected in a different understanding of comfort management. Through the peer-to-peer exchange of tokens between platform users on the B-DLT platform, the new energy services and non-energy services for personal security and AAL in the market can be leveraged.

For the development of the services, the incorporation of historical data is of great importance. Since the data comes from heterogeneous sources and extensive historical data was available before the project started, Pilot 2 developed a unified API to access the available data. Furthermore, in the course of the Bright project, some additional sensors will be installed that will be used to improve the services related to the different programs of DR defined by the use cases in D2.1.

The information objects already available for prototyping and developing initial services are summarized in Table 4 for Slovenian Pilot.

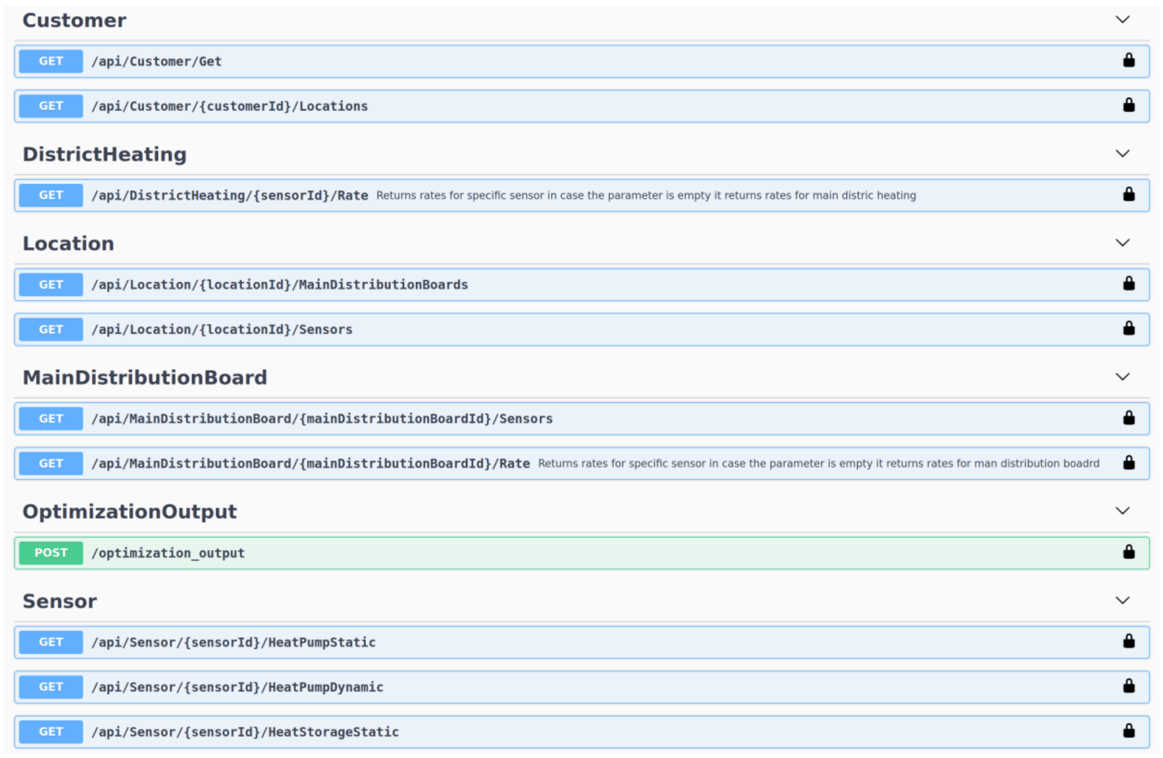
Table 4. List of objects and attributes for Pilot 2 available historical data.

Information object	Attributes	Granularity	Communication frequency (range)	Communication protocol	Data format and information model
Smart meter	Voltage, Active power, Current, 1 & 3 phase, ID, Timestamp	15 min	On demand	HTTP	JSON
Energy meter	Input Temperature, output Temperature, Energy, Active	1 s	On demand	HTTP	JSON

	power Timestamp				
Generation	Energy/power, ID, Timestamp	15 min	On demand	HTTP	JSON
Storage	Temperature, Min. Temp., Max. Temp., Volume Timestamp, ID	1s, 15 min	On demand	HTTP	JSON
Property and measurements	ID, Timestamp, Unit, Value	15 min	On demand	HTTP	JSON
Load	Voltage, Active pow- er, Current, 3 phase, ID, Timestamp	15 min	On demand	HTTP	JSON

4.2 Example Pilot 2

The Slovenian pilot deals with virtual communities and implicit and explicit DR programs. Due to heteregoneus nature of the input data sources it is crucial that the well-defined interoperability layer that supports scalability and privacy is defined. One such system is Kafka message queue system as shown in Section 3. To access the live and historical data, Pilot 2 exposes data through the REST (Representational State Transfer) API. The API has predictable, resource-oriented URLs and uses HTTP response codes to indicate API errors. The API uses HTTP authentication and HTTP verbs which are understood by standard off-the-shelf HTTP clients. All API requests are served by a response in JSON format which can be easily interpreted and converted to other formats inside the microservices and analytics components.



Customer
GET /api/Customer/Get
GET /api/Customer/{customerId}/Locations

DistrictHeating
GET /api/DistrictHeating/{sensorId}/Rate Returns rates for specific sensor in case the parameter is empty it returns rates for main distric heating

Location
GET /api/Location/{locationId}/MainDistributionBoards
GET /api/Location/{locationId}/Sensors

MainDistributionBoard
GET /api/MainDistributionBoard/{mainDistributionBoardId}/Sensors
GET /api/MainDistributionBoard/{mainDistributionBoardId}/Rate Returns rates for specific sensor in case the parameter is empty it returns rates for man distribution boadrd

OptimizationOutput
POST /optimization_output

Sensor
GET /api/Sensor/{sensorId}/HeatPumpStatic
GET /api/Sensor/{sensorId}/HeatPumpDynamic
GET /api/Sensor/{sensorId}/HeatStorageStatic

Figure 4: Slovenian pilot API in a Swagger UI.

The API is used to query the historical sensor data of a specific asset or sensor. The API can be used from the 'Swagger UI' endpoint or by directly sending the requests to the '/api/*' methods from a standard HTTP client.

4.2.1 Customer API

The customer API is used to query the data regarding the specific customer and locations inside the particular pilot. API endpoints are the following:

- /api/Customer/Get:
    - GET will return a JSON with a list of available customer ids and names:

```
[
  {
    "id": "string",
    "name": "string"
  }
]
```

- /api/Customer/{customerId}/Locations:
    - GET will return a JSON with a list of available location ids and names:

```
[
  {
    "id": 0,
    "name": "string"
  }
]
```

4.2.2 District Heating API

The district heating API is used to query the district heating heat energy buying rates:

- /api/DistrictHeating/{sensorId}/Rate:
    - GET will return a JSON with an information on heating energy rates for a district heating unit:

```
[
  {
    "id": 0,
    "start": "2021-10-11T14:23:34.648Z",
    "end": "2021-10-11T14:23:34.648Z",
    "buyRate": 0,
    "sellRate": 0,
    "maxPowerBuy": 0,
    "minPowerBuy": 0,
    "maxPowerSell": 0,
    "minPowerSell": 0
  }
]
```

4.2.3 Location API

Location API is used to query the list of available sensors and the list of available main distribution board ids:

- /api/Location/{locationId}/MainDistributionBoards:
    - GET will return a JSON with main distribution boards ids:

```
[
  {
    "id": 0
  }
]
```

- /api/Location/{locationId}/Sensors:
    - GET will return a JSON with a list of available sensors at the selected location. Each list item contains a sensor's ID, sensor type and sensor name fields:

```
[
  {
    "id": 10,
    "type": "Main",
    "name": "main_meter"
  },
  ...
  {
    "id": 30,
    "type": "HeatPump",
    "name": "TČ1 OGR"
  }
]
```

4.2.4 Main Distribution Board API

The main distribution board contains the information of energy rates and available sensors on the selected main distribution board:

- /api/MainDistributionBoard/{mainDistributionBoardId}/Sensors:
    - GET will return a JSON with available sensors connected to the selected main distribution board:

```
[
  {
    "id": 10,
    "type": "Main",
    "name": "main_meter"
  },
  ...
  {
    "id": 30,
    "type": "HeatPump",
    "name": "TČ1 OGR"
  }
]
```

- /api/MainDistributionBoard/{mainDistributionBoardId}/Rate:
  - GET will return a JSON with a list of rates for the selected main distribution board:

```
[
  {
    "id": 0,
    "start": "2021-10-11T14:46:00.897Z",
    "end": "2021-10-11T14:46:00.897Z",
    "buyRateMT": 0,
    "buyRateVT": 0,
    "sellRateMT": 0,
    "sellRateVT": 0,
    "maxPowerBuy": 0,
    "minPowerBuy": 0,
    "maxPowerSell": 0,
    "minPowerSell": 0
  }
]
```

4.2.5 Sensor API

Sensor API is used to query data related to selected sensors in the particular pilot case:

- /api/Sensor/{sensorId}/HeatPumpStatic:
  - GET will return a JSON with static heat pump info for the selected heat pump device:

```
[
  {
    "id": 0,
    "turnOnCost": 0,
    "turnOffCost": 0,
    "isZeroOneAsset": true,
    "coefficientOfPerformance": 0,
    "fixedRunningConst": 0,
    "variableRunningConst": 0,
    "minUpTime": 0,
    "maxUpTime": 0,
    "minDownTime": 0,
    "maxDownTime": 0
  }
]
```

- /api/Sensor/{sensorId}/HeatPumpDynamic:
  - GET will return a JSON with dynamic info for the selected heat pump device-specific:

```
{
  "id": 0,
  "energy": [
    {
      "start": "2021-10-11T14:51:56.491Z",
      "end": "2021-10-11T14:51:56.491Z",
      "value": 0
    }
  ],
  "power": [
    {
      "dateTime": "2021-10-11T14:51:56.491Z",
      "value": 0
    }
  ],
  "calorimeterEnergy": [
    {
      "dateTime": "2021-10-11T14:51:56.491Z",
      "value": 0
    }
  ]
}
```

- /api/Sensor/{sensorId}/HeatStorageStatic:
  - GET will return JSON with static info for selected heat storage device-specific:

```
{
  "id": 0,
  "minTemperature": 0,
  "maxTemperature": 0,
  "lossCoefficient": 0,
  "heatPumpIds": [
    0
  ]
}
```

- /api/Sensor/{sensorId}/HeatStorageDynamic:
  - GET will return a JSON with dynamic info for a selected heat storage device:

```
{
  "id": 0,
  "heatStorageTemperature": [
    {
      "dateTime": "2021-10-11T14:54:20.804Z",
      "value": 0
    }
  ]
}
```

- /api/Sensor/{sensorId}/DistrictHeatingStatic:
  - GET will return a JSON with static info for a selected district heating device:

```
{
  "id": 0,
  "price": 0,
  "maxHeatPower": 0
}
```

- /api/Sensor/{sensorId}/DistrictHeatingDynamic:
  - GET will return a JSON with dynamic info for a selected district heating device:

```
{
  "id": 0,
  "currentConsumption": [
    {
      "start": "2021-10-11T14:57:13.291Z",
      "end": "2021-10-11T14:57:13.291Z",
      "value": 0
    }
  ]
}
```

- /api/Sensor/{sensorId}/SolarStatic:
  - GET will return a JSON with static info for the selected solar stations:

```
{
  "id": 0,
  "inclination": 0,
  "declination": 0,
  "latitude": 0,
  "longitude": 0,
  "peakPower": 0
}
```

- /api/Sensor/{sensorId}/SolarDynamic:
  - GET will return a JSON with dynamic info for the selected solar stations:

```
{
  "id": 0,
  "energy": [
    {
      "start": "2021-10-11T14:59:43.677Z",
      "end": "2021-10-11T14:59:43.677Z",
      "value": 0
    }
  ],
  "power": [
    {
      "dateTime": "2021-10-11T14:59:43.677Z",
      "value": 0
    }
  ]
}
```

- /api/Sensor/{sensorId}/TemperatureStatic:
  - GET will return a JSON with static info for a selected temperature sensor device-specific:

```
{
  "maxTemperature": 0,
  "minTemperature": 0
}
```

- /api/Sensor/{sensorId}/TemperatureDynamic:
  - GET will return a JSON with a list of temperature measurements with corresponding timestamps:

```
[
  {
    "dateTime": "2021-10-11T15:04:41.963Z",
    "value": 0
  }
]
```

- /api/Sensor/{sensorId}/ChargingStationStatic:
  - GET will return a JSON with static info for a selected charging station device:

```
{
  "id": 0,
  "maxPower": 0,
  "minPower": 0
}
```

- /api/Sensor/{sensorId}/ChargingStationDynamic:
  - GET will return a JSON with dynamic data for a selected charging station device-specific:

```
{
  "id": 0,
  "expectedChargingEndTime": "2021-10-11T15:06:25.096Z",
  "pricePerkWh": 0,
  "energy": [
    {
      "start": "2021-10-11T15:06:25.096Z",
      "end": "2021-10-11T15:06:25.096Z",
      "value": 0
    }
  ],
  "power": [
    {
      "dateTime": "2021-10-11T15:06:25.096Z",
      "value": 0
    }
  ]
}
```

- /api/Sensor/{sensorId}/MainMeter:
  - GET will return a JSON with data from the selected mains metering:

```
{
  "id": 0,
  "energy": [
    {
      "start": "2021-10-11T15:06:53.246Z",
      "end": "2021-10-11T15:06:53.246Z",
      "value": 0
    }
  ],
  "power": [
    {
      "dateTime": "2021-10-11T15:06:53.246Z",
      "value": 0
    }
  ]
}
```

- /api/Sensor/{sensorId}/MainMeterStatic:
  - GET will return a JSON with static info for a selected mains meter device:

```
{
  "id": 0,
  "power": 0,
  "expectedYearlyUsageMT": 0,
  "expectedYearlyUsageVT": 0
}
```

5 Conclusions

This deliverable reports on the data collection and processing from the pilot’s site as well as existing historical data. Since the Bright architecture and the interoperability layer are designed in parallel and defined in D2.3 and D2.4, the full Bright solution could not be evaluated yet. However, the two technologies streaming engine Kafka and Pilot 2 API were tested and evaluated for integration into the interoperability layer. The main advantage of Kafka, which is well suited for the Bright project, is that it allows easy integration of heterogeneous data sources by subscribing and publishing, is highly scalable, fits into a distributed system with low overhead, and is reliable with intra-cluster replications. In addition, pilot’s existing APIs could be easily integrated into the Kafka system, allowing legacy devices to be integrated more easily into the Bright solution.

As this is the first report, the final integration of data into the interoperability layer and the deployment of the fully functional interoperability layer to connect to heterogeneous data and services developed by partners will be shown in the second version of this report scheduled for M30.

References

[1] Apache Kafka. Apache Kafka n.d. https://kafka.apache.org/documentation/#intro_concepts_and_terms (accessed October 19, 2021).
[2] Scalability of Kafka Messaging using Consumer Groups. Cloudera Blog 2018. https://blog.cloudera.com/scalability-of-kafka-messaging-using-consumer-groups/ (accessed October 19, 2021).
[3] Data Streaming Privacy, Security, and Compliance in Kafka. Confluent n.d. https://www.confluent.io/blog/kafka-data-privacy-security-and-compliance/ (accessed October 19, 2021).