D1.5 Data Storage Repository

June 2020

Document Identification: D1.5 Data Storage Repository			
Status	Final	Due Date	30.06.2020
Version	1.3	Submission Date	30.06.2020

Deliverable Number	D1.5
Deliverable name	Data Storage Repository
Work Package number	Work package 1
Delivery due date	30.06.2020
Actual date of submission	30.06.2020
Dissemination level	Public
Lead beneficiary	41 - ViF
Beneficiaries	ViF
Responsible scientists	Elem Güzel Kalayci, Stefan Erlachner, Alois Steiner
Internal reviewer	Oliver Lah

Table of contents

Introduction

The Solutions+ project [1] aims to establish a world wide platform for public, shared and commercial e-mobility solutions, and to push the transition towards low-carbon urban mobility. The work plan of the project comprises city level demonstrations to test different types of innovative and integrated e-mobility solutions, complemented by a comprehensive toolbox, capacity development and replication activities. In the Solutions+ project [1], the consortium has established partnerships with cities in Europe, Asia, Africa and Latin America. The cities participating in this project represent different characteristics with regard to their socioeconomic and policy environment, geographic and structural factors, modal share and energy mix. The partner cities start from different levels of experience in sustainable urban transport and electric mobility. The partner cities have been selected as demonstration cases as they represent key factors relevant for the testing and replication of the innovative technologies and business models tested in this project. The Solutions+ data storage repository will reside as a centralized storage for the collected data in the demonstration actions.

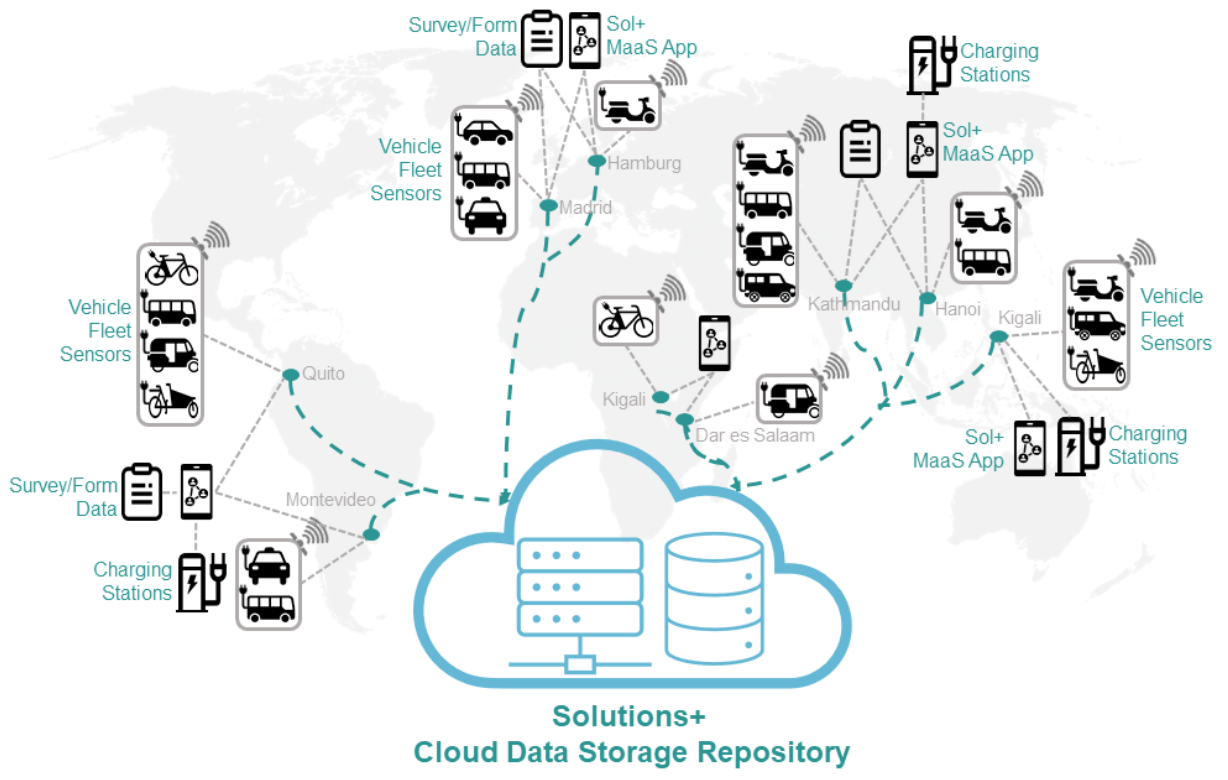


Figure 1 presents an overview of the data collection from planned demonstration actions in the nine partner cities that receive direct funding from the Solutions+ project for implementation and equipment. The demonstration actions support the introduction and integration of electric buses, mini-buses, taxis, 2- and 3-wheelers in partner cities. They will test the development, upgrading and retrofitting of electrified 2 or 3 wheelers, and the operations, charging and integration of different types of electric buses, mini-buses and e-moto-taxis. The vehicles will be bundled into fleets and they will be monitored with various sensors measuring the speed, vehicle position, acceleration etc. The project also aims at contributing to the development of elaborate charging solutions and the setup of interoperable, high power charging infrastructure in the partnering cities. The goal is to establish a charging infrastructure that offers connectivity to various types of vehicles and supply energy for a whole electric bus fleet and other heavy-duty vehicles in a city. The data that is collected from the charging stations and the sensors monitoring the demo vehicles will be stored in Solutions+ data storage repository. In addition to the data that will be collected in the demonstrations, the relevant data that is already available in the partner cities will be also integrated into the Solutions+ data repository. The collected data will be complemented by classical surveys on quality, subjective data collection, and user satisfaction etc. Moreover, Mobility as a Service (MaaS) solutions will feature in all demonstration actions. The innovations to be tested will include the digital applications running some crucial operations like multimodal trip planning, payment and ticketing systems and the provision of information to the users. The logs of these operations will be imported into the Solutions+ data storage repository.

As the integrated data source, the Solutions+ data storage repository will serve the Solutions+ toolbox, which contains capacity building materials, summaries of business plans and models, summaries of innovations tested in the demonstration actions, operations and management tools for e-mobility solutions, information on financing and funding options, etc. The repository will also provide data support on the impact assessment tools, which will eventually guide the demonstration activities, and focuses on the aspects such as technical feasibility and impacts on energy system and energy security, financial viability, social impacts, per service unit, quality of life, vulnerable group needs, environmental and health impacts, and replication and scale-up potential.

The data repository has to ensure interoperability of the integrated data with easy to use data providing/access interfaces as well as it has to guarantee the security and privacy of the data with regards to the concerns of the data owners on storing locations, data access grants, etc. For these reasons we focus on outlining the design of the cloud data storage platform that will serve as the integrated Solutions+ data storage repository. In this deliverable, we first introduce in section 1 the characteristics of the data that will be collected and the storage options for it, in section 2 we briefly discuss the data repository survey, in section 3 we present the architecture of the cloud solution we plan to provide, and in section 4 we conclude the deliverable with an outlook and a discussion on future work.

1. Basic Options for Data Storage Repository

1.1 Data Quality & Quantity

1.1.a Characteristics of the Data

Task 1.3 [1] actively supports the demonstration actions by providing comprehensive impact assessments and evaluations. The task coordinates, supports and assures the quality of the data collected from sensors and traditionally collected data through questionnaires and interviews collected and of the data storage and management. Purpose and scope of the collected data will feed into the impact assessment. In this subsection we present the characteristics of the data that will be collected during demonstration actions.

A big amount of the data that will be collected within the scope of demonstrations is expected to have a fixed structure. Structured data is data whose elements are addressable for effective analysis. It can be described as predefined data that is formatted to a set structure before being stored in a data repository, which is usually referred to as schema-on-write. It concerns all data that can be stored in database SQL[2] in a table with rows and columns having mostly built-in data types. They usually have relational keys and can easily be mapped into pre-designed data fields. In structured data, entities can be grouped together to form new relations (‘vehicles’ that are grouped in a new fleet). This makes structured data easy to store, search and analyze. It is the simplest way to manage information and until the last decade was the only data easily usable for businesses.

Although most of the demonstration data is expected to be structured, there might be some data such as images or videos that are taken during the demonstration drives, or documents in different formats containing unstructured free texts, which cannot fit in a particular schema. Unstructured data [3] (schema-on-read) is stored in its native format and not processed until it is needed to be used. It comes in various file formats, including email, presentations, chats, social media posts, binary files and satellite imagery. There are several advantages to storing data in an unstructured manner. Saving the data in its native format allows for a wider variety of file formats in the data repository. The data that can be stored is not restricted by a specific format. As there is no need to predefine the data, also the data collection and storing process becomes faster and easier. Another advantage would be to benefit from data lakes allowing massive storage and also offering pay-as-you-use storage pricing, which helps reduce costs.

In the context of the demonstrations, it is planned to collect survey data in various topics such as quality, subjective data collection, and user satisfaction. Survey data is expected to have a structure up to a certain point, however, oftenly it cannot be fully structured as it may contain free texts and allows to submit additional materials. Such data is called semi-structured data [3]. Semi-structured data is data that does not reside in a relational database. It lacks a fixed or rigid schema but it has some organizational properties such as tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. These properties make semi-structured data easier to manage and analyze. In semi-structured data, the entities belonging to the same category may have different attributes even though they are grouped together. Unlike the relational data, the order of attributes is not important, and even the type and the size of the same attribute in a group may differ. Due to lack of a well-defined data structure, semi-structured data can not be utilized immediately.

As the data repository will serve to store the data collected during the demonstration actions, it will contain time-series data collected from sensors mounted to vehicles and charging points and also from the Solutions+ MaaS App. A time-series [4] is a series of data points ordered in time. Time-series adds an explicit order dependence between observations. The order of observations provides a source of additional information that should be analyzed. Time series are generally “append-only" and usually assumed to be generated at a regularly spaced interval of time , and so are called regular time-series. The data that will be collected from sensors such as speed, vehicle position, acceleration etc. during the demos is usually considered to arrive on a regular basis. But the data in a time series doesn’t have to come in regularly, as some measurements might not occur at regular time intervals. Such series are called irregular time-series. For example, some of the fleet measurement time-series data such as energy/fuel consumption or charging time may not be recorded regularly. Time-series can have multiple variables that change over time. The time-series having more than one variable are called multivariate time-series**.** For example, a tri-axial accelerometer data that will be collected from accelerometer sensors is a multivariate time-series. In a tri-axial accelerometer, there are three acceleration variables, one for each axis (*x,y,z*) and they vary simultaneously over time.

Geospatial data [5] exists in a variety of formats and usually contains more than only location specific information. It could be a vector in the form of points, lines or polygons. Connecting points form a line, and connecting lines creates an enclosed area which is called polygons. Vectors are utilized to present generalizations of features or objects on the Earth’s surface. Another way of representing geospatial data is to represent it in rasters which is a grid of pixels. Each pixel within a raster has a value. This value could be a colour or a unit of measurement to communicate information about the entity of interest. Geospatial data usually contains more information than only a position on the surface of the Earth. Any additional non-spatial data that describes a feature is called an attribute. Geospatial data can have any amount of additional attributes accompanying the geolocation information. It is likely that Solutions+ demonstrations will also involve collection of geospatial data associated with attributes.

1.1.b Storing the Data

The most known way of storing structured data is storing it in a relational database: relational databases store data in tables, which are organized into columns. The data, such as SSID, credit card numbers or address, has been formatted into precisely defined columns in which one datatype (e.g. integer, real number, string, date) is assigned. Each row in tables represents an instance of the table. In relational databases, tables can be related to each other. The integrity and consistency of individual instances in tables and of relations between tables are maintained with key and integrity constraints. Relational databases are queried with the structured query language SQL. Cloud service providers support most of the major relational databases such as PostgreSQL, Microsoft SQL Server, Oracle DB, SAP HANA etc. as well as they provide their own cloud data warehouses. As we expect a big amount of structured data to be imported in, the relational database will be one of the key components in the Solutions+ data storage repository.

There exists many other options for storing and querying structured data. Another widely used option is to store data in graph databases [6] that organize data in the form of a graph. A graph can be considered as a collection of nodes and edges, where nodes typically represent entities, and edges represent the relationships between those entities. One of the distinguishing aspects of graph databases is that they are inherently weakly schemed, or schema-less. It means that as time passes, and the graph is needed to be extended to hold new types of entities and data, one can simply add new nodes into the graph that describe them without breaking what is already there, or having to make complex schema changes to the database. The biggest enterprise clouds service providers Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft Azure support the most widely used graph database Neo4J [7]. There exist also other widely used solutions provided by cloud enterprises like Neptune owned by AWS and CosmosDB owned by Microsoft Azure. Although we are not planning to integrate a graph database into the initial version of the Solutions+ data storage repository, we plan to take it into consideration in the later phases if a need for a graph database emerges in the further discussions with the project partners.

As the necessity of processing big data increases in the last decades, the popularity of non-relational NoSQL (not only SQL) databases [8] grows equally. NoSQL databases do not store data in tables. Instead there are multiple ways to store data in NoSQL databases such as key-value, column-based, and document-based. Key-value databases are the most simple of all the NoSQL databases, where the basic data structure is a dictionary or map. A value can be stored as an integer, string, JSON, or an array with a key used to reference that value. Column-based databases store data in grouped columns rather than in rows. Similar to the schema in a relational model, column-based databases use keyspaces that contain multiple column families. Column families are similar to tables in a relational model. Column-based databases provide many advantages such as efficient data compression, high performance with aggregate functions and scalability across a ton of machines. In document-based databases data is structured in the form of documents and collections. A document can be in PDF, Word document, XML or JSON format. Each document is associated with a key and does not have to be in the same structure as other documents. This gives the flexibility to add new documents without having to change the structure of the entire database. Documents are grouped into collections, which serve a similar purpose to a relational table. Document databases enable querying collections of documents with particular attributes. Document databases provide flexible data modeling and eliminate the need to force fit relational schemas, as they can handle unstructured, structured as well as semi-structured data. There are a lot of options on storing NoSQL data in the cloud. Most widely used ones are MongoDB, Cassandra, Redis, CouchBase etc. We expect to receive a significant amount of unstructured data. Depending on in which form they are preferred to be kept and for which purposes they are going to be used, in the further phases we see the possibility to integrate a NoSQL database into the repository.

Time-series databases [9] are relatively new compared to RDBMS or NoSQL databases, however, they are trending up with the growth of interest in system monitoring and the internet-of-things. A time-series database is a database that is dedicated to storing and querying time-series data. While it's possible to store and query time-series in a relational or NoSQL database, a time-series database has specialized time-related functions that would be more complicated and less efficient to implement in ad-hoc functions executed in other types of databases. Time-series databases enable efficiently conducting the operations where time is the first class citizen. Some of the often used essential operations are as follows: *(i)* Time-weighted aggregates that could be operated on irregular intervals not having a fixed periodicity, *(ii)* Time downsampling while time-series data is recorded with very high frequency but it is often needed to be analyzed only with a lower frequency. Time downsampling functions return data at the request interval and they can interpolate the values between the timestamp before and after, in case the lack of an exact match to the timestamps' of the requested interval. *(iii)* Efficient implementations of comparison operators dedicated to time related comparisons, *(iv)* Tailored indexing and optimization on insert/append operations and effective data compression benefiting from the a priori knowledge on timestamps being key values. There exist many time-series databases some of which are built for time-series from scratch such as widely used InfluxDB, and some of which use existing SQL or NoSQL databases as the backend such as KairosDB using NoSQL database Cassandra at the backend, and TimescaleDB using SQL database PostgreSQL at the backend. We expect the time-series to cover the major part of the data that will be imported into the Solutions+ data storage repository. We plan to involve a time-series database into our repository in case we receive requests for efficient execution of time-dedicated operations mentioned above.

1.2 Data Versioning & Archiving

Archives are used to store data that remains valuable to an organization but is no longer in active use. Data archiving [2] differs from data replication in that it is not used in the purpose of disaster recovery or to create backup copies of data. Replication provides protection for the current state of a data volume, while data archiving is more about protecting a volume's historical versions. Data archives are typically created for auditing purposes, to save the management and maintaining costs, or to meet compliance requirements. When frequently-requested, hot data often stays on fast and expensive storage formats to ensure its ready availability, while cold data such as archive data is usually stored on inexpensive, capacity storage. This is one of the best practices for archiving data to reduce operational costs. However, it can often be the case that organizations find it difficult to effectively archive unstructured data, which intermittently require fast, on-demand access to it. They end up with high performance storage for storíng any data that could increase the cost drastically.

Data archiving methods should make use of the most cost-effective solutions for data storage, ideally without the need to sacrifice easy access to the data when required. In order to take advantage of the durability and cost effectiveness of cloud object storage, one can choose from a range of storage options such as that offered by Amazon S3 or Azure Blob. Together with the data owner and consumer partners in the Solutions+ consortium we are going to discuss the requirements on archiving the Solutions+ data and we are going to investigate on the most efficient option that fits in the project needs.

1.3 Data Providing/Access Interfaces

Depending on the provider, there exists many different ways of loading data into a data storage repository. If one has a collection of data in batches, the simplest way is to directly import CSV, XLS, JSON or SQL files into a data repository. A more convenient and widely used method is to use an API such as REST [10] or GraphQL API [11] as the intermediary application that ensures a standard communication between the data repository and the data provider, which could be a manual importer or more likely a Solutions+ toolbox application, or Mobility as a Service application. One can also utilize JDBC/ODBC application programming interfaces as the API in between.

API's can be directly utilized for data access or can serve as a middleware between the repository and a frontend application that provides a web user interface to navigate over the data as well. A more detailed discussion on data providing and access interfaces will be outlined in Section 3.a.

1.4. Security & Privacy

1.4.a Authorization and Authentication

Database security [12] is an important component of any organization's information security plan. It involves protecting the data repository from any unauthorized access, modification and destruction attempt. As well as preserving and protecting the data for the orderly functioning of the organization, the database managers have also the responsibility of protecting the privacy of the individuals whom data is stored. The design of the repository should reflect the commitment of the organization to the protection of the individuals privacy rights by storing only those data that the owner of the repository has right to know and also by keeping them secure. Developing a secure data access mechanism involves different aspects such as authorization and authentication.

Authorization [12] means to explicitly grant permissions in different levels for the database objects to make them accessible to users. DB administrators specify permissions for users by means of authorization rules stated usually in SQL. Authorization rules are statements that specify which users have access right to what data objects, and also what operations they are permitted to execute on what data. Authorization can be applied in different ways. The most simplistic way of authorization is to use a least-privileged user account (LUA) approach ensuring that users follow the principle of least privilege and always log on with limited user accounts. However, least privileged users may require elevated permissions to run an application over the database in order to function orderly. Granting excessive permissions to users in order to regain functionality can leave the database vulnerable to attacks. Another authorization way is to grant role-based permissions. Permission sets that are assigned to roles are inherited by all members of the role. It is more efficient to work with a role-based authorization approach compared to recreating separate permission sets for each individual user.

Another important aspect of database security is authentication [12], which can be defined as the verification process of the identity of a user. The most basic and widely used authentication method is to use an ID and password. However, this method is not very secure as the users may not be very careful with protecting their password. Strong authentication can also be ensured with a multi-factor approach: the combination of things the user knows (e.g. password), and things the user has (e.g. token card). Multi-factor authentication has several advantages. It provides more choices of authentication methods such as tokens, smart cards, operating system authentication, or network authentication services (e.g. Kerberos). Such authentication mechanisms may also reduce the administrative overhead.

1.4.b Hosting Locations

The cloud has drastically changed the shape of the data storage landscape. It enables greater cost efficiency, more flexible business practices and reduced resource needs. However, the adoption of cloud raises many challenges in the face of new, and sometimes competing, privacy regulations across various jurisdictions. In that sense, location of the data becomes important for several reasons. One reason is that data protection laws and regulations can greatly differ between countries, in respect to both access of data and who can access it. For example, organizations relying on multiple cloud service providers may have little or no control over the transfer of their data from one data center to another around the world. For this reason, organizations often want to ensure that their data is stored in a cloud host within their jurisdiction so that it is covered by their legal system. This eases maintaining control of data and it is very critical for organizations to have full control of the data when they store important documentation in the cloud. So, it is not surprising that organizations enjoying the advantages of outsourced cloud services are also concerned about their data privacy and protection in the cloud as it can be problematic to have their data remain in a territory in a conflicting jurisdiction.

Cloud Service Provider	**Number of Regions**	**Located in**
Amazon Web Services	~24	U.S., Canada, France, Germany, Spain, Italy, Sweden, Ireland, the U.K., Australia, India, Japan, Singapore, South Korea, Sweden, Brazil, Bahrain, South Africa, and China.
Microsoft Azure	~60	U.S., Canada, Germany, France, Norway, Spain, Poland, Italy, Australia, New Zealand, India, Norway, UAE, Switzerland, Japan, Korea, Ireland, the Netherlands, Hong Kong, Singapore, South Africa, Israel, Qatar, and Brazil.
Google Cloud	~23	U.S., Belgium, Poland, Finland, Belgium, Switzerland, Germany, the Netherlands, the U.K., India, Qatar, Australia, Brazil, Canada, Hong Kong, Indonesia, Korea, Japan, Singapore, and Taiwan.

The pioneering cloud service providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud offer many options on where to locate the data across the world (See the table above). The service users' may need to deploy their application workloads across the globe, or they may want to deploy specific applications closer to their end-users. The cloud service providers tailor their cloud infrastructure offers with respect to the service user's location needs and privacy concerns. In this respect, we recommend to see the report titled *"Magic Quadrant for Cloud Infrastructure as a Service, Worldwide"*[1] from Gartner for a detailed service comparison of the leading cloud providers.

It is very important to us to be aware of privacy concerns and security needs of our project partners in the aspects detailed above, to review their concerns and needs carefully, and to decide accordingly on which cloud storage option we take.

1.5 Backup & Restore

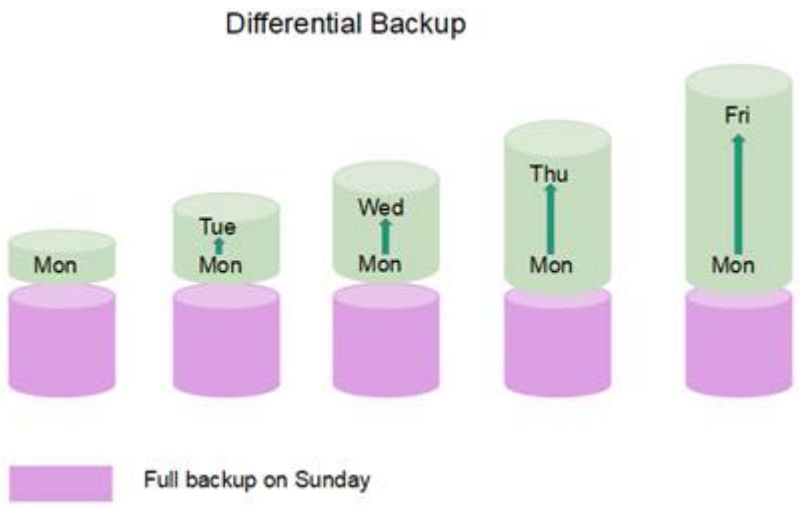
The main purpose of a data repository is to store and provide data to different stakeholders. Since a lot of time and effort is spent on collecting these data, the data loss should be limited to a minimum in case of a disaster. Components that should be backed up often reside in the persistence layer of the system. The main backup strategies [13] can be divided into full, differential, and incremental backups. Although these techniques differ on how the backups are organized, they all have in common that they represent a state of the data at a specific point in time (snapshot). An overview of the characteristics is shown in the table below.

Backup strategy	**Backup Base**	**Backup Speed**	**Backups required for recovery**
Full	Full Backup	Slow	Last full backup only
Differential	Full Backup	Medium	Last full backup + last differential backup
Incremental	Full or incremental backup	Fast	Last full backup + all incremental backups

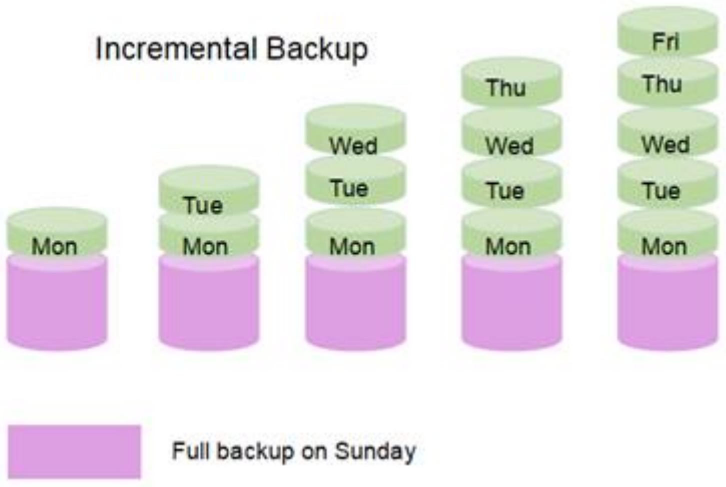
Full backup: The simplest backup strategy is creating full backups where the entirety off the data is backed up every time. An obvious downside of this approach is the amount of space that is needed to store all these backups. Especially as we will end up backing up the same data again and again regardless of if the data has changed at all. Also, the amount of time spent performing the backup is the highest compared to other methods.

Differential backup: To tackle this issue a differential backup strategy can be used. Here only data that has changed since the last full backup is included in the differential backup. This approach still leads to increasing backup sizes because all changes after the last full backup will be included in every differential backup until a new full backup is performed.

[1] https://www.gartner.com/en/documents/3947472/magic-quadrant-for-cloud-infrastructure-as-a-service-wor



Incremental backup: Incremental backups behave like differential backups in the sense that they also only include changed data in the backup file. But instead of using the last full backup as the baseline only data that has changed since the last incremental backup is included. This backup strategy produces the smallest space requirements, but it is also the most vulnerable as every single incremental backup, following the full backup, is required to restore data.



Often all the effort will be put on the creation of the backups, but the truth is that no backup will be sufficient, in case of a disaster, if the recovery / restore process is not thought through and tested sufficiently. Although having some backup is better than nothing it is beneficial to have the recovery process laid out **before** the worst-case scenario. There are several types of failover and recovery strategies that differ in downtime and the number of manual steps needed to get a backup site up and running.

Cold backup system: A cold backup system consists only of backup files stored in a save location. The term cold is used because a new instance must be created with the most recent backup to get the backup system up and running. This leads to significant downtime of the system and a huge workload on the administration side.

Warm backup system: The term warm backup is used if a second instance, besides the production, is set up that continuously receives updates from the productive system. Therefore it is capable of taking over the major responsibilities of the productive system in case of failure. Benefits in comparison to the cold backup are the less to zero amount of data loss and the reduced response time until a fallback system is up and running. Some warm backup solutions limit the access to the backup system to read-only.

Hot backup system: An exact copy of the production system is set up that is working in parallel or in the background. In such a setup there might even not exist a designated production instance and all nodes are equal. This means that if one node fails or is destroyed another node will take over without the end user even noticing. Nodes organized in such a cluster can also highly improve the availability and performance of the system, although of course it takes more time to setup and maintain such a system.

2. Solutions+ Data Storage Repository Survey

We prepared a data repository survey in order to get familiar with the characteristics of the Solutions+ data prior to its collection as well as with the needs and concerns of the project partners who could be either the provider or the user of the data that will be stored in the data storage repository. The survey helps us to get some insights about the data that will be stored, and also to further detail the feasible options for the data storage repository. It contains two main sections targeting the data providers and the data consumers. The data providers section involves the questions about data quality, quantity and versioning as well as the preferences on data providing interface, and data security/privacy. The data consumers section involves the questions about the preferences on data consuming interface and data export formats. We group the answers to the survey questions as follows:

Data Quality & Quantity: The data has a big diversity ranging from structured time-series sensors data, aggregated impact assessment results, energy consumption measurements, user need assessments, CAN measurements and GIS files and maps to semi or unstructured surveys, interview transcripts, pictures and videos. Data provider partners plan to provide the various amounts of data in many different formats such as SQL, CSV, XLS, XML, GPX, DOCX, KML etc.

Data Versioning & Archiving: The data provider respondents have different preferences on how long it is needed to archive the data after the project lifetime. The answers range from 1 month to 3-5 years.

Data Providing Interface: The majority of the respondents prefer basic file imports and web forms for data loading. In addition, one respondent specified their need for data import via continuous data streams using web sockets.

Security & Privacy: The majority of the respondents stated that they do not have any concerns on storing the data in the cloud. 25% of the respondents remarked that their data should get aggregated and anonymized if it will stay in the cloud. 25% of the respondents have concerns on where on earth the data servers will be physically located. The majority of the respondents state that they need different level authorization groups.

Data Access: The respondents have diverse preferences on the ways of accessing the data. The majority prefer to utilize web user interfaces, probably having a data tagging mechanism to navigate over the data while the rest prefer to use API's or basic JDBC/ODBC interfaces. The majority of the respondents also prefer to export data in CSV or JSON format.

The data repository survey gave us some insights on what kind of data we are going to store in the repository, in what structure the data is going to be, what additional storage, indexing, querying possibilities we should take into consideration, in what extension we might need to utilize additional plugins or other types of databases dedicated to specific types of data (e.g. time-series, geospatial), as well as the preferences of the project partners on data providing/access and their concerns on data privacy and security. Based on the review of the responses to the survey, in the next section we outline the architecture of the data repository we plan to construct.

3. Architecture

In this section we want to outline a possible architecture approach for the data repository. This outline is not meant to be final and can be changed as new requirements arise and areas of operation become more specific. The basic architecture follows a 3-tier structure which splits the system into components that interface with other systems or users (Frontend), the business logic and request handling on the server (Backend) as well as a persistence layer which involves all major storage technologies like databases and file directories.

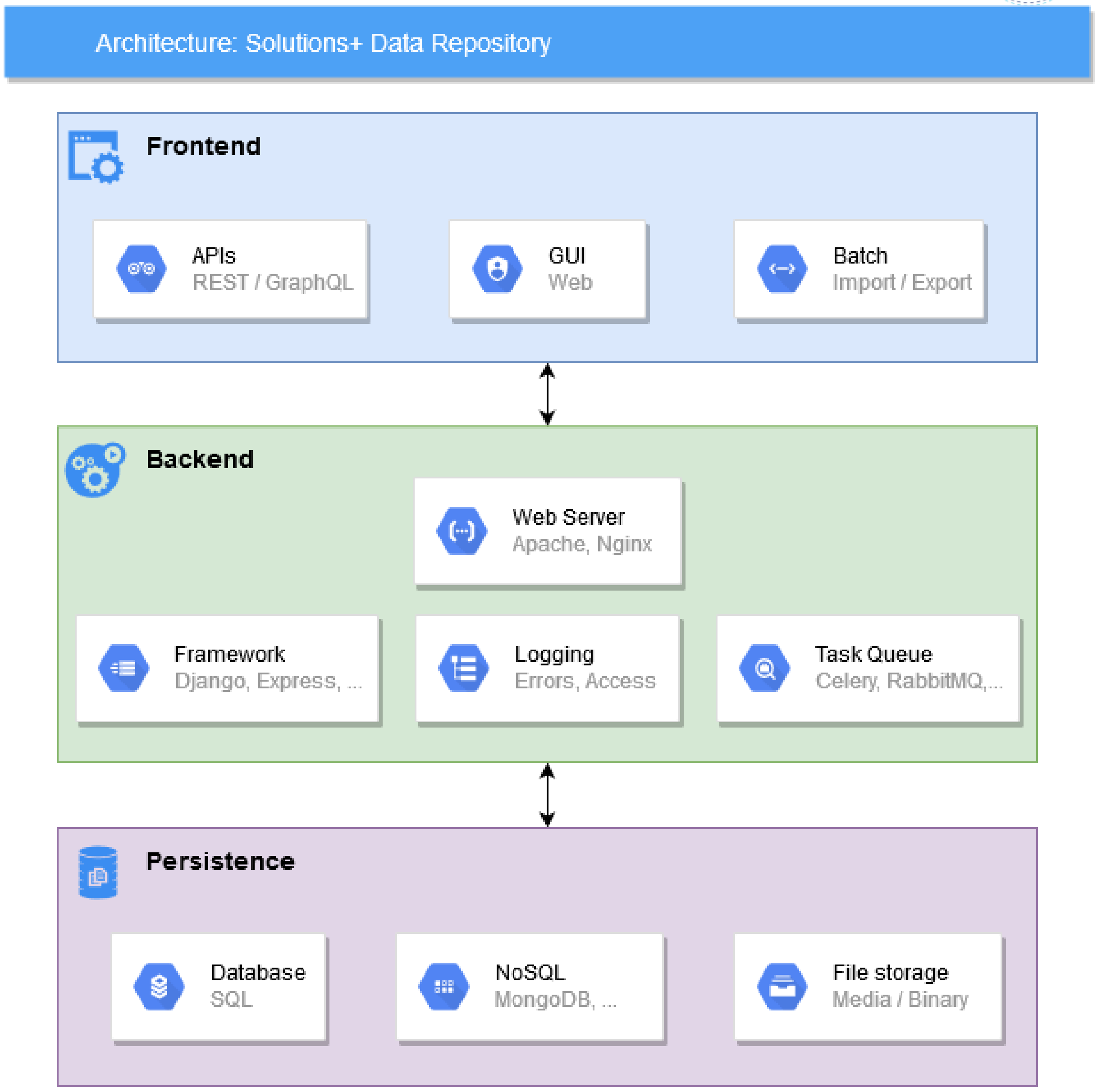


Figure 2: 3-Tier-Architecture of the Solutions+ Data Repository

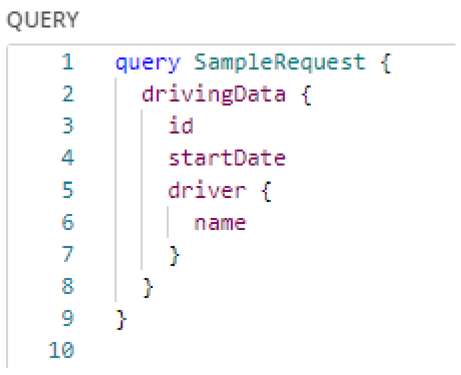
3.1 Frontend

The frontend tier contains all components that are meant to be used to interact and access the repository. It is a good approach that every request and interaction between the repository and other systems and users must go through a frontend component to ensure proper security and logging capabilities are met. In general, we distinguish between three different use cases in which the system can be accessed. The first possibility is by using an "Application-Programming-Interface" (API) to retrieve and modify data. An API can be used by external systems to work with the data provided by the repository. As for all interfaces a proper authentication must be provided before accessing the repository. Right now, two major API types are considered standard and are widely used. The most widespread and mature type is REST. REST uses the basic HTTP methods to indicate which operation should be executed (see table below).

HTTP method	Description
GET	Retrieve a record identified by the accessed URL. GET requests should not provide a payload.
POST	Main purpose is to add new data records, but it is often also used to update records instead of using PUT or PATCH.
PUT	Overwrites the data record with the content of the payload.
PATCH	Partially modifies attributes of the data record specified in the payload.
DELETE	Deletes a record.

Another API approach that gains popularity, is GraphQL. GraphQL is a query language for APIs developed and maintained by Facebook. One of the biggest benefits, compared to REST, is that, once the schema is fully defined in the backend, the frontend application decides what and how to query data from the API. That can lead to a strong decoupling between front- and backend, while still the backend in the background can decide how data is queried and processed. In a nutshell, the frontend tells the backend via a query what it needs, and the backend responds with the data accordingly. Therefore network traffic can be highly reduced since unwanted data does not need to get delivered to the client. Also, GraphQL eases querying of relations between data by just defining them in the schema and using these relations in the query accordingly (see example below). This is a huge benefit compared to REST APIs which provide a static set of endpoints with a fixed amount of data that gets retrieved. Also querying relational dependencies can be tedious in REST if no specific endpoints are provided. Lastly while REST endpoints provide many endpoints, a GraphQL API only needs one endpoint to receive queries and respond with the according data.

Example: GraphQL query



```
query SampleRequest {
  drivingData {
    id
    startDate
    driver {
      name
    }
  }
}
```

While APIs are already a great way to interact with the data repository their main application interfacing with other systems. Therefore the usage for human-users can be tedious and is not comfortable at all. That is where graphical user interfaces (GUIs) come into play. They provide an easy and intuitive way for users to manage and lookup data. In the case of Web-GUIs the user interacts with the system by browsing through HTML pages provided by a web server.

These pages can either be provided entirely by the backend as pure HTML sites or take use of the client's browser capability to execute JavaScript code, which leads to a more responsive look and feel. In the last case a JavaScript Framework such as Angular, React or Vue can be used to transfer some logic to the client itself.

What we covered so far are the connections to external systems through APIs as well as the graphical user interface that can be used by human-users. One thing that is still missing is a way to upload and download multiple data records at once. Operations that target multiple data records are referred to as batch operations. These are most of the time performed asynchronously in the background due to the longer response time that is needed for fetching and processing multiple records. The batch interface should provide the possibility to upload data coming from a structured file format like .csv, .xls, etc. to the data repository as well as download data in a structured file format. For the upload to work properly the input file must follow a predefined structure that can be interpreted correctly by the backend.

3.2 Backend

Web applications highly depend on the "Hypertext Transfer Protocol" (HTTP) to communicate with web browsers or handle requests coming from other applications through an API. The service that handles all these different requests is referred to as a Web Server or HTTP Server. Two of the most popular web servers include Apache and Nginx which are also very extendable by including various plugins and modules. By the usage of such modules attack vectors such as "Denial-of-service" and "Man-in-the-middle" should be countered by the Web Server before even reaching other parts of the system. To prevent "man-in-the-middle" and "phishing" attacks an SSL certificate should be obtained and used to encrypt the connection between the Web Server and the clients.

After the requests are received and processed by the Web Server, they are passed down to the server application for domain-specific processing. Since a lot of steps and requirements are the same for any application, Web Frameworks are often used to speed up the development. Also, developers can then focus on the key points of the application. The decision which framework to use highly depends on the requirements as well as preferences regarding programming languages.

One major functionality of a data repository that can be easily overlooked is the logging mechanism. It is worthwhile to consider the requirements and constraints of centralized access and error logging right in the beginning of the development process to avoid time consuming reworking and integration of the logging functionality afterwards. Proper logging is not only necessary for debugging and fixing bugs but also for access control and documentation.

Task queues enable the backend to run scheduled and long-running tasks asynchronously in the background. One application for a long-running task would be a batch import of a file into the database. Whereas a scheduled task might be calculating aggregational values or housekeeping operations like archive & purge.

3.3 Persistence

The persistence layer consists of components and services that deal with non-volatile data. These components must not be directly accessed by any client and should be regularly backed up and secured properly. A relational database is in most cases the preferred choice to store structured data. Besides a SQL database a dedicated place to store media/binary files is most often required by web applications.

In the case that requirements ask for storing loose structured data a NoSQL database can be introduced. These databases highly differ in their overall structure and functionality and should be chosen on demand. Possible options might include document-based databases like MongoDB or Graph databases like Neo4j. To transfer data between a NoSQL and a relational database, an ETL (Extract-Transform-Load) process with different adapters can be introduced.

Conclusions and outlook

In this deliverable we present the characteristics of the Solution+ data that we expect to collect during the project lifetime, and the basic options of the data storage repository from the data structure, versioning, interfacing, backup, security and privacy perspectives. We also outlined the architecture of the Solutions+ data storage repository with regards to the up-to-date cloud service technologies as well as the data repository survey that gives insights about the preferences and concerns of the data owners and users.

As future work, we plan to establish the concrete cloud storage for the Solutions+ data repository and to ensure reliable and secure access to it. Moreover, we further investigate to better understand the needs of the project partners on accessing and querying specific types of data such as time-series, NoSQL, or geospatial, and also to enrich our repository with additional abilities on managing and efficiently accessing and querying these specific type data. Another topic we plan to investigate is to present an easy-to-use data access interface that could be utilized by any project partner.

References

[1] Project Proposal (2019), Solutions+: Integrated Urban Electric Mobility Solutions in the Context of the Paris Agreement, the Sustainable Development Goals and the New Urban Agenda.

[2] Elmasri, R., & Navathe, S. (2007). *Fundamentals of database systems*. Boston: Pearson/Addison Wesley.

[3] Date, C.J. *An Introduction to Database Systems 8th edition,* Pearson.

[4] Chatfield, C., & Xing, H. (2019) *The analysis of time series: An introduction with R 7th edition*, CRC Press.

[5] Lloyd, C. D. (2010), *Spatial data analysis: an introduction for GIS users*, Oxford University Press.

[6] Robinson, I., Webber, J. & Eifrem, E. (2015), *Graph Databases 2nd edition*, O'Reilly.

[7] Baton, J., & Van bruggen, R. (2017), *Learning Neo4J 3x 2nd edition*, Packt publishing.

[8] Vaish, G. (2013), *Getting started with NoSQL*, Packt publishing.

[9] Dunning, T., & Friedman, E. (2014), *Time series Databases: New ways to store and access data*, O'Reilly.

[10] Masse, M. (2012), *REST API: Design rulebook*, O'Reilly.

[11] Porcello, E., & Banks, A. (2018), *Learning GraphQL: Declarative data fetching for modern web apps*, O'Reilly.

[12]Ricardo C. M., & Orban S. (2017), Introduction to database security, *in Databases Illuminated 3rd Edition*, Jones & Bartlett Learning.

[13] Walters, R., & Fritchey, G. (2012), Database backup strategies, *in Beginning SQL Server 2012 Administration,* APress.