Ref. Ares(2021)5947826 - 30/09/2021

D2.5

LEAD DDDAS System v1

Deliverable number: D2.5
Author(s): Dimitra Politaki, Antonis Mygiakis, Ioanna Fegardiotou
Author’(s’) affiliation (Partner short name): INLE

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 861598. LEAD is a project under the CIVITAS Initiative. Read more - civitas.eu

Deliverable No.	**2.5**		
Work Package No.	**2**	Work Package Title	**Digital Twin Model and Simulation Environment**
Task No.	**2.4**	Task Title	**DDDAS System**
Date of preparation of this version:	**26/03/2021**		
Authors:	**Dimitra Politaki (INLE)** **Ibad Kureshi (INLE)** **Ioanna Fergadiotou (INLE)** **Antonis Mygiakis (INLE)** **Abdelhadi Belfadel (IRTX)** **Sebastian Hörl (IRTX)** **Rodrigo Tapia (TUDELF)** **Ioanna Kourounioti (TUDELF)** **Angel Batalla (LMT)** **Bråthen Svein (MOLDE)** **Nagy Vivien (BKK)**		
Status (F: final; D: draft; RD: revised draft):	**F**		
File Name:	**DDDAS_D2.5_v0.8.doc**		
Version:	**0.8**		
Task start date and duration	**01/02/2021 – 31/08/2022**		

This document is issued within the frame and for the purpose of the LEAD project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 861598.

The views represented in this document only reflect the views of the authors and not the views of the European Commission. The dissemination of this document reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Revision History

Version No.	Date	Details
0.1	26/03/2021	1st draft version
0.2	20/7/2021	LL3 - Lyon
0.3	25/7/2021	LL2 – The Hague
0.4	24/8/2021	LL1 - Madrid
0.5	27/8/2021	LL5 - Budapest
0.6	01/09/2021	LL4 - Oslo
0.7	02/09/2021	First version for initiating the review process
0.8	22/9/2021	Update after reviews

Reviewers List/

Name	Company	Date	Signature
Jakob Puchinger	ISX	20/9/2021	
Rodrigo Tapia Ioanna Kourounioti	TUDELFT	20/9/2021	

Executive Summary

This document describes the LEAD Data-Driven Dynamic Application System (DDDAS), which refers to the system responsible for the data ingestion and the execution of the simulations of the Digital Twin (DT) platform. The current document reflects the system's design decisions by month fifteen (M15) of the project. The scope of this document is to present the design decisions and an initial implementation of the DDDAS towards building a scalable, robust, flexible, and reusable platform for urban logistics operations.

With reference to the deliverable D2.1 "Technical Requirements – Solution Architecture", LEAD develops a scalable and robust DT platform for urban logistics operation. The platform is reliable, and interoperable and based on an open architecture with open software standards, ensuring that privacy and ethical issues are respected by design. The platform includes ready-to-go integration connectors for the seamless ingestion of multiple, voluminous, and heterogeneous data-in-motion and data-at-rest sources. Moreover, the platform supports analytics and efficient complex data driven simulations, reproducing operational conditions and by managing diverse data sources and the connectors' provision.

The design of the DDDAS, and by extension the platform, was developed through iterative discussions and virtual meetings with the Living Lab (LL) stakeholders and the teams designing closely related systems such as the Model Library (ML). This report includes an in-depth description of the LEAD platform architecture providing context for the design of the DDDAS. The platform's high-level description and design choices are provided to the reader, together with its respective components. Finally, an initial deployment of the platform is presented, displaying the solutions and the technologies used.

A very important and critical aspect, for the platform in general but also specifically for the DDDAS, is the ingestion of data in a scalable and flexible manner that can accommodate the various needs of the LLs. To that end, a presentation of the data ingested by each LL has been provided, offering the information needed towards a unification of the data entities that would enable simulation models' interoperability.

Last, this report focuses on the DDDAS communications with other system components and its role in defining key performance indicators (KPIs), selecting appropriate models, configuring input data and parameters, and executing a model's sequence.

This deliverable consolidates the challenges, specifications, design, and data that have led to the initial implementation presented. The DDDAS is a dynamic component that evolves together with the needs of the LLs and the provided simulation models.

1 Content

List of Figures

List of Tables

Acronyms

Acronyms	**Descriptions**
ABM	Agent Based Model
API	Application Programming Interface
BEV	Battery Electric Vehicles
CNG	Compressed Natural Gas
CI/CD	Continuous Integration and Continuous Deployment
CSV	Comma Separated Values
DDDAS	Data-Driven Dynamic Application System
DSS	Decision Support System
DT	Digital Twin
EDV	Electrical Delivery Vehicle
EV	Electric Vehicles
FLP	Future Location Prediction
GA	Grant Agreement
GDPR	General Data Protection Regulation
GTFS	General Transit Feed Specification
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LEZ	Low Emissions Zone

LL	Living Lab
MASS-GT	Multi-Agent Simulation System for Goods Transport
MATSim	Multi-Agent Transport Simulation
NDA	Non-disclosure agreement
RDBMS	Relational Database Management System
SLA	Service-Level Agreement
UI	User Interface
WSZL	Waberer's-Szemerey Logisztika Kft

2 Introduction

In LEAD project, WP2 focuses on the implementation of a digital twin platform that supports decision making in last-mile logistics operations. To tackle such complex problems, the digital twin can perform simulations, guide logistics decision making and evaluate its impact.

The objective of the LEAD DDDAS Deliverable is to design and implement the Dynamic Data-Driven Application System (DDDAS). The DDDAS has a special role for the LEAD platform. It is orchestrating the data ingestion, the digital models, the simulation, and optimisation environment as well as the connection between the Digital Twin and the physical world. It is a core component that contains a large portion of the features that a Digital Twin platform encompasses.

2.1 Mapping LEAD Outputs

This deliverable is based on the commitments made on LEAD's Grant Agreement. In the following, a mapping between these commitments, as described in the Deliverable and the Tasks, and the project's respective outputs and work performed is presented. A list of the deliverable's components and task description together with the corresponding chapter of this report that discusses the approach that was followed and work that has been carried out can be seen on Table 1.

Table 1 Adherence to LEAD's Deliverable & Tasks Descriptions

LEAD GA Component	Document chapters & Justification
DELIVERABLE	
D2.4 DDDAS Initial prototype of the DDDAS system with (i) the data handling pipeline to enrich and validate streaming data, (ii) unification of datasets from different LLs, (iii) orchestration of models and data.	The deliverable focuses on the design and implementation of the DDDAS component of the LEAD platform. Initially, the platform architecture is presented to provide a foundation for the system's description. Then, the data management and storage principles of the DDDAS system are discussed along with separate cases for each LL and their connections to the meta-model. Finally, the system's specifications along with the initial deployment and its orchestration are provided to the reader.
TASKS	
ST2.4.1: Data Handling Pipeline	In Chapter 3 the overall LEAD architecture is presented, overviewing the general data handling approach of the platform. Furthermore, in Section

Using the API's designed by the previous task this sub-task ingests the data from external systems, performs data clean up, validation, and where possible data augmentation and fusion. In a dynamic Big Data environment data validation is critical and complex task. This pipeline then either stores data for future use (batch or micro-batch), uses it to validate the scenarios where ground truth is missing (i.e., based on the following set of conditions what was the outcome?), or pass it on to the DDDAS sub-system for immediate use.	0, information regarding the data ingestion pipeline for each LL is presented. It covers the data pre-processing, augmentation, and thresholding validation. Moreover, sections 4.4, 4.5, and 4.6 further analyse the management and validation of the input data, thresholds while also presents the management of the results from the execution of the simulation models.
ST2.4.2 Dynamic Data-Driven Application Systems Once the scenarios have been programmed using the visual interfaces, the DDDAS system manages the simulation process by provisioning computational hardware, initializing the models and managing outputs. The key complexity in this task is temporal and change management. Temporal complexity comes from either different simulation horizon requirements (i.e., real-time, short, medium- or long-term predictions) or simulation execution timescales (i.e., the combination of models for each possible outcome may take different times to execute). The DDDAS system also needs to continue to monitor the Data Pipeline and make decisions on whether simulation outcomes will be relevant or not. A drastic change in ground realities may make a simulation run moot and may require re-initialisation, alternatively the DDDAS can choose to spool up parallel simulations and then using future real-time data validate whether the outcomes were significantly affected using the validation data.	Chapter 5 covers aspects of the DDDAS design and implementation. First, methodology for defining the specifications, the key actors of the system and the functional and non-functional requirements are set (5.1). These sections cover the functional, non-functional, design, and component analysis of the DDDAS. Since the models from the LLs are not yet ready for execution in the platform, temporal challenges are going to be addressed in Period 2 of the project, as the complexity of the LL representation will have evolved and further models will have been added to the Model Library. Finally, an early overview of the threshold management for the LEAD MVP is reported in this deliverable. Further work will be carried out once the LLs are operational and capable to refine this functionality. This will be reported in D2.6 (M27).

2.2 Deliverable Overview and Report Structure

This report presents an in-depth description of the LEAD's Dynamic Data-Driven Application System (DDDAS), its design and architecture, how it interfaces with the physical world and within the LEAD platform as well as the dashboards it offers to the user.

In **Chapter 3** the overall LEAD architecture is presented. It offers to the reader a crucial knowledge foundation for better understanding of the DDDAS. Then, in **Chapter 4** the data ingestion pipelines are analysed separately for each LL's case. The DDDAS is then discussed (**Chapter 5**); the specifications of the system, its actors and requirements are initially presented while the architecture, components and implementation follow giving a thorough description of the system to the reader.

Finally, a conclusion (**Chapter 6**) summarizes the information provided in the deliverable while also presenting an outlook of the DDDAS in the context of the overall project.

3 LEAD Platform

In the following sections, an overview of the LEAD platform architecture is going to be presented. The DDDAS central to the overall platform's design since it is the starting point where simulations are configured, and the data inputs and model parameters are set.

3.1 High-Level Architecture

The LEAD platform's purpose is to provide the infrastructure where the models can be executed towards building a complete digital twin, serving the needs and use cases of the LLs. As displayed in **¡Error! No se encuentra el origen de la referencia.**, data should be ingested from the physical world into the platform through APIs, established IoT sensors' protocols or in a different fashion as defined by the availability and accessibility of the operational data.

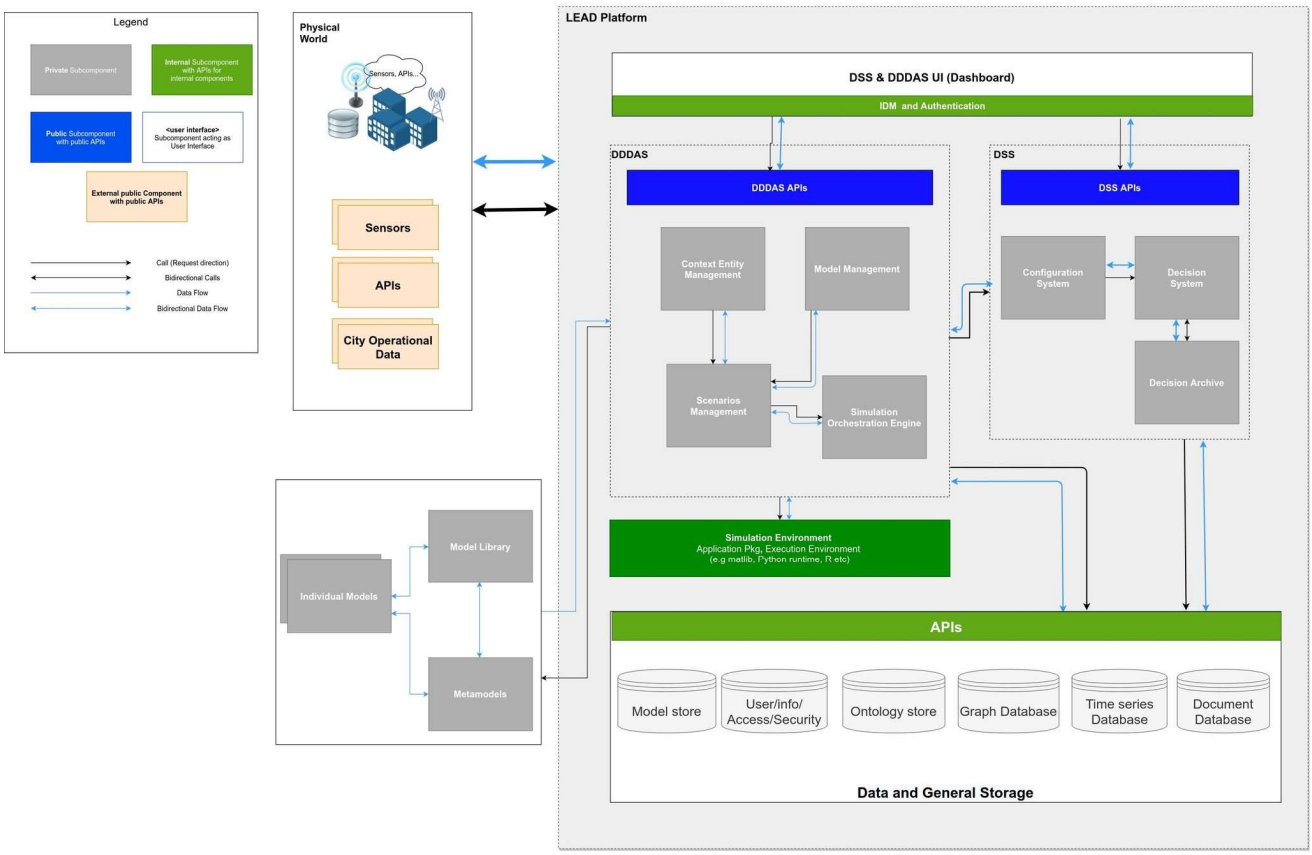

Figure 1 LEAD platform high-level architecture

The DDDAS is responsible to build the context entities describing the physical world but also communicate with the Model Library to connect those entities with available models. The DDDAS also manages the scenario creation and the simulation orchestration, making any outputs available to the DSS. The communication of the DDDAS and DSS concerns the results of a simulation. The DSS is designed to identify the best-case scenarios through cognitive methods and present this output to the

user. Both the DDDAS and DSS communicate with a dashboard that accepts all user inputs and provides access to the storage infrastructure of the platform.

3.2 Components

The LEAD platform has been designed in a modular fashion so that it accommodates a wide range of requirements that emerge from the various use cases of the LLs. Such use cases involve investigations regarding a single definitive business decision while others concern repetitive simulations with changing parameters on real time and historical data.

Thus, the DDDAS has been designed to use multiple systems for data ingestion to accommodate different data scenarios, such as message brokers for streaming data, a distributed file system for large static or nearly static data, big data stores such as Hadoop, etc. Additionally, the system is expected to run a variety of models developed by different partners and using different technologies.

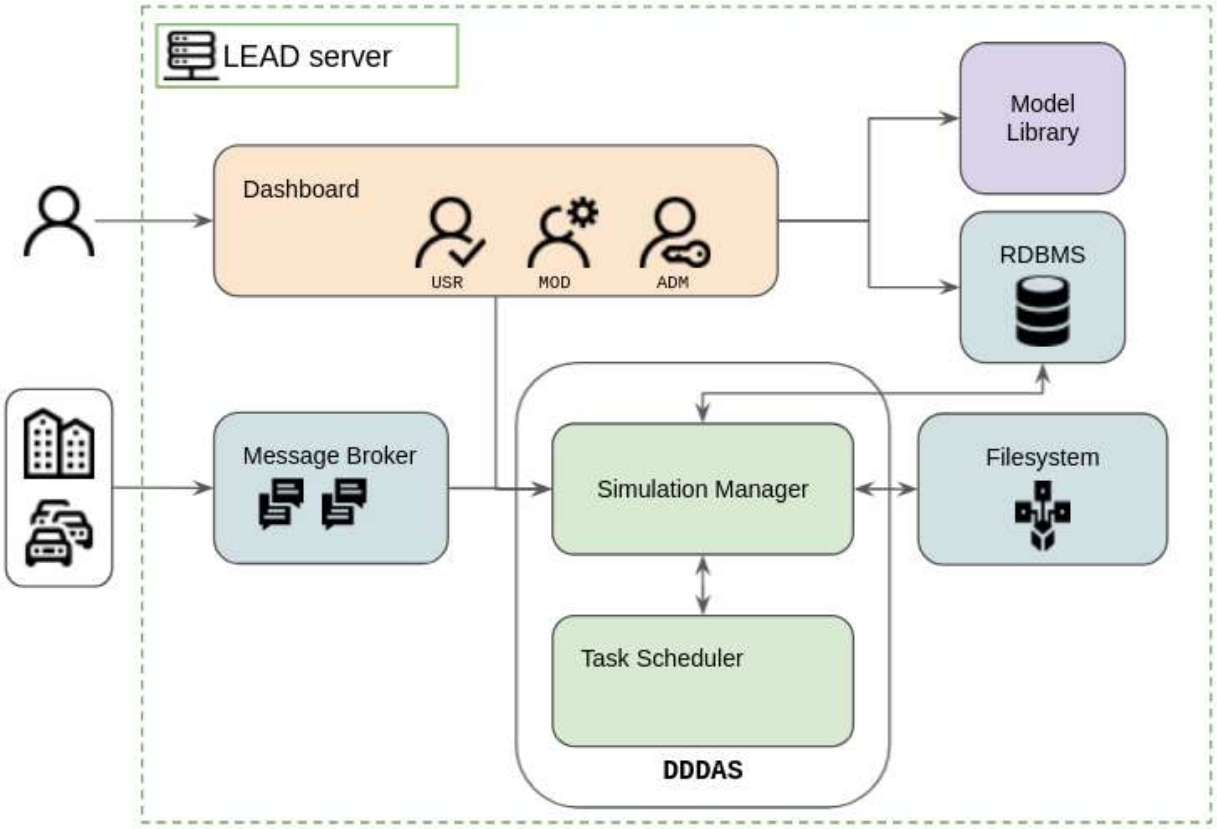

Figure 2 An abstract overview of the DDDAS-related components.

An abstract overview of the DDDAS-related components is presented in Figure 2. The user interface component is the main interaction point of the platform with the user. The dashboard exposes functionality that is integrated into the DDDAS and DSS. From there, the user can browse through the available KPIs and configure a scenario; that includes providing a model sequence along with input

parameters and associated datasets. Therefore, the simulation management component of the DDDAS along with the Model Library guide the user through the scenario creation together with appropriate and user-friendly frontend elements. Additionally, the simulation manager accesses and retrieves any information necessary from the storage infrastructure consisting of the filesystem, the databases, and a message broker. After the submission of a scenario the simulation manager communicates with the task scheduler to orchestrate the simulation's execution. After a scenario simulation execution, the simulation manager is notified and handles the storing and indexing of the results to the appropriate storage resources.

3.3 Deployment

With references to the criteria and solution analysis presented in the deliverable D2.1 "Technical Requirements – Solution Architecture" [1], we choose for the LEAD platform the following technologies. As an initial step, the platform is hosted on a single server and can be scaled as needed. It is also deployed with CI/CD procedures and all microservices are containerized. The dashboard's frontend is implemented using the popular JavaScript framework ReactJS[2] while the backend is based on Python Flask[3]. A Relational Database Management System (RDBMS) in the form of PostgreSQL[4] is chosen. A document-based database along with an in-memory data structure is also possible to be provided depending on the use cases and the performance achieved.

Apache Kafka[5] is a well-established open-source event streaming platform that is often preferred for applications with real time data. Kafka topics are therefore used for data storage, and they can be distributed to other platforms through Kafka Connect in a scalable and reliable manner. Kafka Connect imports data from any external source – called Source connector – and exports data to any external system – called Sink connector- offering the flexibility of changing data source/export system at any time in the future without changing the stream processing code. The Kafka cluster is administered by a Zookeeper[6]; this service keeps track of the cluster's metadata, such as the nodes, topics, partitions and so on.

Another platform under the Apache Foundation has been selected as a task scheduler. Apache Airflow[7] is a platform to programmatically author, schedule and monitor workflows. Comes pre-loaded with features that provide dynamicity and extensibility to the platform's model execution. It also offers an elegant UI that will be used for monitoring the execution pipelines. Finally, the machine's resources along with metrics from the Kafka subsystem and the storage infrastructure are monitored through Prometheus exporters. The collected data can then be visualized with dedicated dashboards (Grafana, Prometheus) in a variety of different graphs and with alerting functionality built in.

[1] *D2.1 "Technical Requirements – Solution Architecture", LEAD project*
[2] *https://reactjs.org/*
[3] *https://flask.palletsprojects.com/en/2.0.x/*
[4] *https://www.askpython.com/python-modules/flask/flask-postgresql*
[5] *https://kafka.apache.org/*
[6] *https://zookeeper.apache.org/*
[7] *https://airflow.apache.org/*

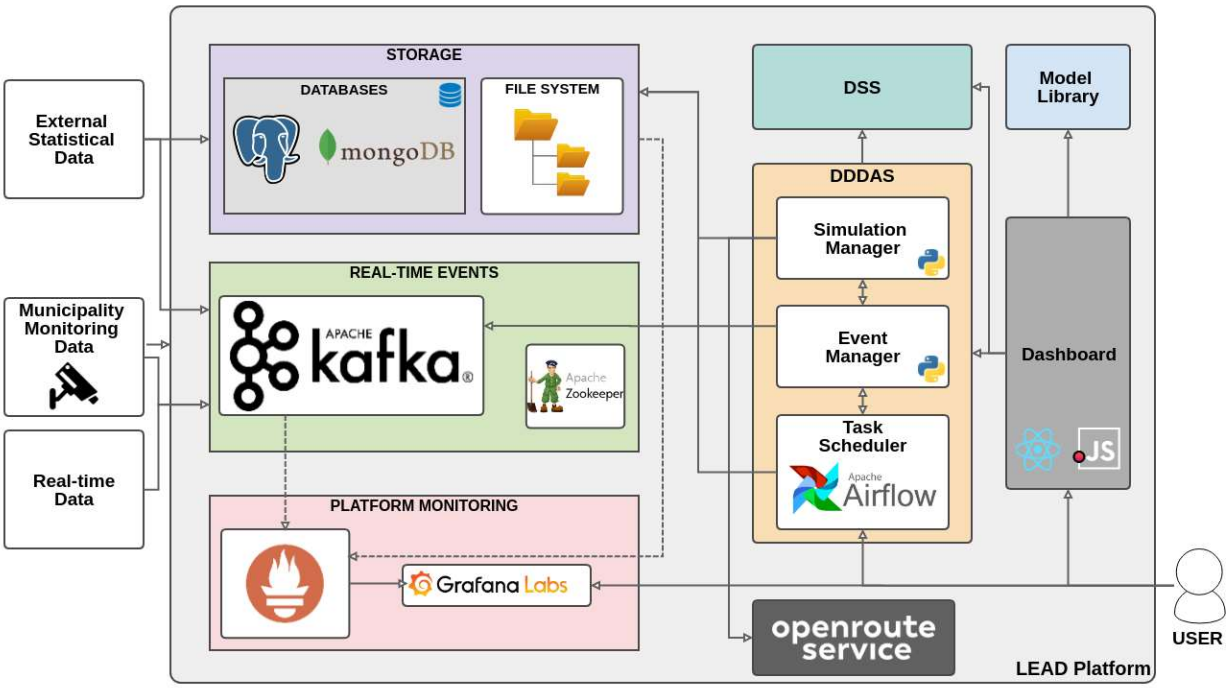

Figure 3 LEAD platform deployment

4 Data Pipeline

4.1 Pipeline Principles

To ensure reproducibility of the platform, all component installations have been automated and scripted. These scripts are managed by a version control system and are part of a Continuous Integration and Continuous Deployment pipeline (CI/CD). This pipeline will monitor the code repositories of the various components and any changes will trigger re-testing and operator instructed redeployment of the platform. In terms of system functionality, the next phase of the platform development will focus on ingesting more data sources and integrating them with the higher layers of the architecture, e.g., Dashboard. As more data are integrated into the system and more analysis is performed, the requirements and expectations of the underlying system will evolve. This task will adopt an agile approach in dealing with change requests.

Kafka topics are used for a variety of tasks in the context of the DDDAS system. Firstly, they are the main point of reference for the physical-world real-time data. The data ingestion to the Kafka topics will be visualized with a platform (e.g., Grafana[8]) so that the data rate, volumetrics and availability for each LL can be precisely defined and monitored. Moreover, the incoming real-world data can be used in conjunction with historical data for the simulations but also as a validation measure on the output of a simulation, provided that the models allow for such input. Furthermore, specific topics can pass information to the dashboard together with the scheduler API so that the user is updated on the status of the simulations that are being executed. In addition, information regarding the simulation execution history and in cases output data will be stored in Kafka topics.

4.2 Data Storage

As previously mentioned, the Apache Kafka distributed event streaming platform is mainly used for the storage of the incoming streaming data. It is used because of its reliability and scalability, but also for the flexibility it provides to share the data with other data storage solutions through Kafka-Connect. Since the DDDAS platform is made to support the execution of models that are developed using a wide variety of technologies and for different cases, the potential to seamlessly provide the data to a variety of storage solutions is of uttermost value and importance. As a result, the Kafka topics, a RDBMS and a time series database are provided as out-of-the-box storage solutions that can be expanded based on the needs of the LLs and their models.

4.3 Data Ingestion

4.3.1 LL1 - Madrid

4.3.1.1 Data Connectors

In Madrid LL, the three different data types, as defined in deliverable D2.3, are 1) Vehicle information, 2) Service information, and 3) Settings information. As the LL scenario is a totally new initiative, there

[8] *https://grafana.com/*

is no historical data currently available. Data will be provided by CLOGIN and PAN, in Excel files (xlsx, csv). These files will be uploaded to the server using File Transfer Protocol (FTP) and stored in appropriate data structures such as document and time-series databases. Before the platform extracts data from it, pre-processing and augmentation of the data is required, as described in Section 4.3.1.2. To simulate Madrid use cases and scenarios, data are received through the Kafka platform and stored to dedicated (Kafka) topics, each topic corresponds to each data type - 3 in total for Madrid LL - while an extra topic is used to contain with all the data types merged information as described in Section 4.3.1.2. Moreover, the Kafka producers ingesting the data are responsible for any data conversion needed by the data consumers of each model. Kafka producers that can be manually executed on historical data are also implemented to support different use cases. During the simulation execution, model consumers use the necessary data from the available sources.

4.3.1.2 Data Pre-Processing & Augmentation

In Madrid LL, data pre-processing and augmentation are crucial to developing specific models for routing optimization on current and the new scenarios. To extract general data characteristics such as how many clients and answer important questions for developing LL models such as how many deliveries in a given time window on each of the five postal codes of Madrid LEZ, delivery time period, number of deliveries per day of week/month/year and more, data should be enriched with some extra information. The initial data are provided in Excel files as mentioned in Section 4.3.1.1 and the extra information is stored initially as extra columns to them. For data pre-processing and augmentation, Python and specifically the pandas library is used and therefore data are stored in DataFrame objects, that is 2-dimensional labelled data structures with columns of potentially different features/indicators offering a great level of functionality for further data analysis.

A fundamental source of information to be fed into the Madrid route optimization models is the delivery addresses dataset that must be expressed in latitude and longitude coordinates (stored in 2 new columns of the delivery data frame with labels delivery latitude, and delivery longitude respectively). Moreover, aiming at computing statistical characteristics at the pre-processing stage is needed to split delivery date to day, month, and year respectively and save them to separate columns of the order data frame and convert date to day of the year using among others the datetime python package. All the information regarding orders, deliveries, fleet, and settings are then merged to one data frame and potentially to one Excel/csv file based on a common field. The common field between order and fleet data frames is the waybill number.

4.3.1.3 Data Stream Meta Extraction

As previously mentioned, the Madrid LL data are provided as Excel files and after the data pre-processing and augmentation stage, they are stored in 3 different Kafka topics. Each Excel file or Kafka topic has its own features/characteristics that can be considered as metadata for data extraction. During the execution of a simulation, these files are retrieved based on conditions, and this procedure can be documented and stored in the document-oriented database. Metadata from data streams can also be meta extracted if needed based on scenarios and user requirements that are defined through the user dashboard.

4.3.2 LL2 - The Hague

The Hague LL aims to explore the interactions of different delivery methods for parcel distribution. The LL will model crowd shipping solution as well as the integration of the various delivery methods into one platform. The platform includes the interactions with other logistical and technological small and medium enterprises. Using MATSim, MASS-GT and network models, it is expected to investigate their impact to the parcel delivery efficiency.

4.3.2.1 Data Connectors

In its current version, the DT modelling of the LL has 6 static datasets, as described in deliverable 2.3. The details on the mechanisms, protocols and storage of this data are still being defined and they will be available in the next months. The zonal data and parcel nodes data are delivered in shape files. The parcel demand, delivery time of parcels, activity-based data and vehicle data are tables that will be provided in an Excel file or CSV format. More data sources are expected to be included in future versions of the DT; it is yet to be confirmed by the LL's partners.

4.3.2.2 Data Pre-Processing & Augmentation

The data processing and augmentation is done within the models and data format is presented in D2.3. Once new data streams will be identified, the pre-processing and augmentation processes will be enhanced accordingly.

4.3.2.3 Data Stream Meta Extraction

As mentioned in section 4.3.2.1, The Hague LL data are provided as Excel or CSV files and after the data pre-processing and augmentation stage, they are stored in 3 different Kafka topics . Each Excel/CSV file or Kafka topic has its own features/characteristics that can be considered as metadata for data extraction. For the shapefiles, the metadata can be obtained in a XML format. During the execution of a simulation, these files are retrieved based on conditions, and this procedure can be documented and stored in the document-oriented database. Metadata from data streams can also be meta extracted if needed based on scenarios and user requirements that are defined through the user dashboard.

4.3.3 LL3 - Lyon

In the Lyon LL a consolidation hub will support the local last mile operations within the district whilst the LEAD platform will be used for exploring the impact of alternative delivery modes on various socio-economic KPIs.

4.3.3.1 Data Connectors

The data to be collected include traffic data from Grand Lyon Métropole (open dataset) together with historical socioeconomic and logistics data. Another set of data is to be provided by the operators involved in the experimentations of the LL, concerning existing operation indicators on other sites and local indicators once the LL is implemented (parcel's types and volumes, vehicles, origins, and destination). All these data will be provided under .csv format and stored in IRT SystemX servers to be executed by the simulation models to populate the scenarios.

Data will be generated by local video cameras to supervise the local logistics traffic. Data will be stored in a dedicated IRT SystemX server to ensure GDPR compliance before adequate processing (anonymization).

4.3.3.2 Data Pre-Processing & Augmentation

Rough data will be pre-processed to generate the actual characteristics of logistics operation at LL level and beyond (customers, volumes, requests, operations). Historical socioeconomic, logistics and traffic data will be extracted to generate a synthetic population and elaborate the distribution scenarios (as-is and to-be) relevant for the LL.

Video data will be processed through image recognition algorithms to determine the share of logistics vehicles in the global traffic flow. These treatments will support the generation of a range of new KPI illustrating the contribution of logistics operations in the global traffic and its timely tendencies.

4.3.3.3 Data Stream Meta Extraction

Different datasets will be established: logistics and socioeconomics data will be stored in different containers than video data, due to GDPR requirements. Processed information extracted from video data could be stored into the main dataset. Each specific folder could be considered as metadata for extraction as requested for simulations. Data stream can be extracted if needed using specific metadata based on scenarios/user requirements elaborated in the simulation.

4.3.4 LL4 - Budapest

Budapest LL aims to investigate the effects of WSZL deliveries to public transportation with use of micro hubs. The first goal is to define the possible locations of the micro hub in connection with the zones, which need developments. Secondly, we focus on using dedicated time windows for delivering to local shops. At last, we are investigating the effects of different vehicle types.

4.3.4.1 Data Connectors

In Budapest, the different data types are defined in Deliverable 2.3 like 1) store information, 2) mini-hub information, 3) fleet information, 4) origin destination information and 5) order information. All currently available data are historical and provided by WSZL in Excel files. These files are uploaded to a server using File Transfer Protocol (FTP) and stored in appropriate data structures such as document and time-series databases. Before the platform extracts data from them, data pre-processing and augmentation needed as described in Section 4.3.4.2. To simulate Budapest use cases and scenarios, data are received through the Kafka platform and stored to dedicated (Kafka) topics, each topic corresponds to each data type - 5 in total for Budapest LL and an extra topic with all the data types merged as described in Section 4.3.4.2. Moreover, the Kafka producers ingesting the data are responsible for any data conversion that is needed by the data consumers of each model. Kafka producers that can be manually executed on historical data are also implemented to support different use cases. During the simulation execution, model consumers use the necessary data from the available sources.

4.3.4.2 Data Pre-Processing & Augmentation

In Budapest LL, data pre-processing and augmentation are crucial to develop specific models both for routing optimization, rescheduling, and managing the delivery time period. To extract data general characteristics such as how many orders per client in a given time window, those zip codes of the

stores and mini-hub in Budapest, that will be delivered by WSZL. The WSZL will provide Excel files with information such as the type and capacity of the vehicles, the vehicle quantity of their fleet and the working time of the vehicles' drivers. Moreover, WSZL will give information about the orders such as delivery date, time-window, unloading time and order quantity as mentioned in Section 4.3.4.1. BKK will provide the Macroscopic Transport Model of Budapest for the DT with the static traffic data (the origin-destination matrices will be fixed except the freight layer) and territorial data. Thus, in the Budapest case, we have 5 different data frames that correspond to 5 different data types that are mentioned in Section 4.3.4.1.

Very important information for the Budapest model are the stores' and mini-hub's addresses, which will be expressed in geolocation format (latitude, longitude), using the database provided by WSZL. We then merge order, fleet, mini-hub and store information to one data frame and potentially to one Excel/csv file based on a common field.

4.3.4.3 Data Stream Meta Extraction

In Budapest LL, as mentioned in Section 4.3.4.1 the different types of data are in Excel files and after a data pre-processing and augmentation, they are stored in 5 different WSZL Kafka topics. Each Excel/file or Kafka topic has its own features/characteristics that can be considered as metadata for data extraction. Each time during the simulation, these files are retrieved based on requirements, and this procedure can be documented and stored in the document-oriented database. Data stream can be meta extracted if needed using specific metadata based on scenarios/user requirements that are defined via the user dashboard and/or API.

4.3.5 LL5 - Oslo

Oslo LL aims at testing four scenarios for NIMBER's B2C home deliveries of larger products from IKEA furniture superstore in a stepwise refinement from dedicated vans and direct deliveries to deliveries via a micro hub through crowd-shipping and, finally, with additional suppliers, adjacent to IKEA, from NIMBER's business customers database. These scenarios will, wholly/partly, use e-vehicles. At this stage, the descriptions that follow may be subjected to changes.

4.3.5.1 Data Connectors

In Oslo LL, the 3 different data types as defined in Deliverable 2.3 are 1) store information, 2) trip information and 3) order information. All currently available data are historical and provided by NIMBER in Excel files or other appropriate formats. These files are uploaded to a server using File Transfer Protocol (FTP) and stored in appropriate data structures such as document and time-series databases. Preliminary to platform extracting data from them, one must pre-process and augment them according to the procedures illustrated in Section 4.3.5.2. To simulate Oslo use cases and scenarios, data are received through the Kafka platform and stored to dedicated Kafka topics, each topic corresponds to each data type - 3 in total for Oslo LL and an extra topic with all the data types merged as described in Section 4.3.5.2. Moreover, the Kafka producers ingesting the data are responsible for any data conversion that is needed by the data consumers of each model. Kafka producers that can be manually executed on historical data are also implemented to support different use cases. During the simulation execution, model consumers use the necessary data from the available sources.

4.3.5.2 Data Pre-Processing & Augmentation

In Oslo LL, data pre-processing and augmentation are crucial to developing specific models to test the viability of each scenario with respect to the given KPIs. Extracting data/information with respect to general characteristics such as the number and features of orders/customers, the routing of the deliveries with/without the micro hub, the number of bringers and their vehicle types, fuel types, kms driven etc., implies referring to the data provided in Excel files as mentioned in Section 4.3.5.1. For data pre-processing and augmentation, adequate software like e.g., Python is applied, and data are stored in convenient data structures with columns/records containing the relevant features/indicators. Hence, in the Oslo case, we have 3 different data structures that correspond to 3 different data types that are mentioned in Section 4.3.5.1.

One of the most important information that should be extracted for the Oslo models is the origin and destination addresses to be reported in geolocation format (latitude, longitude), using Google Maps Platform and specifically Google Key APIs. The order and delivery latitude and longitude coordinates are stored with labels order latitude, order longitude, delivery latitude, and delivery longitude respectively. Moreover, aiming at computing statistical characteristics at the pre-processing stage one needs to split both order and delivery date to day, month, and year granularity respectively and save them to separate columns of the order data frame and convert date to day of the year using, among others, the datetime python package. We then merge store, trip, and order information to one data frame and potentially to one Excel/csv file based on a common field. The common field between store, trip and order data frames is the order id.

4.3.5.3 Data Stream Meta Extraction

In Oslo LL, as mentioned in Section 4.3.5.1 the different types of data are in Excel files or similar and after a data pre-processing and augmentation, they are stored in 3 different NIMBER Kafka topics. Each Excel/file or Kafka topic has its own features/characteristics that can be considered as metadata for data extraction. Each time during the simulation, these files are retrieved based on requirements, and this procedure can be documented and stored in the document-oriented database. Data stream can be meta extracted if needed using specific metadata based on scenarios/user requirements that are defined via the user dashboard and/or API.

4.3.6 LL6 - Porto

Porto LL aims at transforming SONAE's deliveries towards electric mobility and at optimizing its fleet operations. The first goal is to position the EDV charging stations to SONAE's stores by considering some parameters such as vehicle type, delivery lead time, distance travelled, traffic, frequency, and duration of EV stop. Secondly, we focus on routing optimization and rescheduling orders finding alternative shortest paths.

4.3.6.1 Data Connectors

In Porto LL, the 4 different data types as defined in Deliverable 2.3 are 1) store information, 2) charging station information, 3) fleet information and 4) order information. All currently available data are historical and provided by SONAE in Excel files. These files are uploaded to a server using File Transfer Protocol (FTP) and stored in appropriate data structures such as document and time-series databases. Before the platform extracts data from them, data pre-processing and augmentation needed as described in Section 4.3.6.2. To simulate Porto use cases and scenarios, data are received

through the Kafka platform and stored to dedicated (Kafka) topics, each topic corresponds to each data type - 4 in total for Porto LL and an extra topic with all the data types merged as described in Section 4.3.6.2. Moreover, the Kafka producers ingesting the data are responsible for any data conversion that is needed by the data consumers of each model. Kafka producers that can be manually executed on historical data are also implemented to support different use cases. During the simulation execution, model consumers use the necessary data from the available sources.

4.3.6.2 Data Pre-Processing & Augmentation

In Porto LL, data pre-processing and augmentation are crucial to developing specific models both for routing optimization, rescheduling, and location charging stations. To extract data general characteristics such as how many clients, how many clients and reply important questions for developing Porto models potentially such as how many orders per client in a given time window, which are the code postal in Porto that SONAE is delivering, order time period, delivery time period, number of delivery per day of week/month/year, some extra information should be extracted from data provided in Excel files as mentioned in Section 4.3.6.1 and stored initially as extra columns to them. For data pre-processing and augmentation, Python and specifically the Pandas library is used, and data are stored in data frames, 2-dimensional labelled data structures with columns of potentially different features/indicators. Hence, in the Porto case, we have 4 different data frames that correspond to 4 different data types that are mentioned in Section 4.3.6.1.

One of the most important information that should be extracted for the Porto models is the store and delivery addresses to be expressed in geolocation format (latitude, longitude), using Google Maps Platform and specifically Google Key APIs. The order and delivery latitude and longitude coordinates are stored in 4 new columns of the order data frame with labels order latitude, order latitude, delivery latitude, and delivery longitude respectively. Moreover, aiming at computing statistical characteristics at the pre-processing stage is needed to split both order and delivery date to day, month, and year respectively and save them to separate columns of the order data frame and convert date to day of the year using among others the datetime python package. We then merge order, fleet, and store information to one data frame and potentially to one Excel/csv file based on a common field. The common field between order and fleet data frames is the vehicle license of the delivery car and the common field between store and order data frame is the store id.

4.3.6.3 Data Stream Meta Extraction

In Porto LL, as mentioned in Section 4.3.6.1 the different types of data are in Excel files and after a data pre-processing and augmentation, they are stored in 4 different SONAE Kafka topics. Each Excel/file or Kafka topic has its own features/characteristics that can be considered as metadata for data extraction. Each time during the simulation, these files are retrieved based on requirements, and this procedure can be documented and stored in the document-oriented database. Data stream can be meta extracted if needed using specific metadata based on scenarios/user requirements that are defined via the user dashboard and/or API.

4.4 Data Input Validation

Providing a set of valid data as inputs to the models is a challenging process for the LEAD platform, since multiple actors have an active role in the development of the models to be executed. Therefore, a well-defined set of metadata that characterizes all data that are ingested to the platform is needed.

To that end, when the user is presented with the options to provide input data to a model, based on its specification as provided by the Model Library, data types are queried across the file system, Kafka topics, and the available databases. A dedicated DataTypes table in the RDBMS of the platform handles the management of the available data types and entries are added in cases such as:

- The execution of a scenario that is generating data
- The addition of a data source providing data to a Kafka topic
- An update to the model library

Firstly, static data stored in databases, or the file system have their RDBMS table entries generated or entered manually. Similarly, every topic that is created for a new data source is accompanied with its corresponding entry in the DataTypes table. Finally, the description of the data generated by each model should be provided by the Model Library so that the output data are discovered by the scenario execution finalization scripts and are properly tagged as new rows to the table. Therefore, a complete view of the types of the available data can be extracted and data can be provided to the models based on such properties.

However, the platform only provides such a higher-level approach and several points of caution need to be addressed by the model owners and the platform's administrators. The data are provided to the models with the validation covering aspects of their metadata. This does not involve their internal structure or content. As such, proper validation, and error handling at runtime by the models is still essential.

4.5 Input Threshold Monitoring

Following the selection of the KPI and the corresponding model(s), rules regarding a successful simulation scenario are applied in the form of thresholds. These thresholds are the means to evaluate the model's potentials and guide the users towards valuable business insights. Given that a scenario can be executed multiple times based on the range and step of its input parameters, a proper monitoring of the KPIs evaluation with regard to the input thresholds can provide useful feedback to the users.

Such functionality is provided through a dedicated topic of the message broker that publishes JSON messages concerning the KPI outputs of running scenarios. Such output can also be linked to a visualization platform such as Grafana for better monitoring experience. Finally, based on the monitoring output and through the web interface of the Apache Airflow, the user has the ability to act on the scenario execution pipelines by cancelling one or more steps to save resources or to further investigate the model execution process through its logs.

4.6 Model Result Management

A final significant part of the scenario execution process is the storage of the results of an executed scenario in a consistent but flexible manner that would facilitate data sharing across models. The platform receives a definition of the output that it expects from every model through the specification provided by the Models Library. At the end of the execution of a model, a set of metadata is stored in the DataTypes table of the RDBMS of the LEAD platform. Each row contains the metadata of the output with information such as the output data type, the date of scenario execution, the output path,

URL or topic, the model and its execution parameters, the user that initiated the scenario and the execution duration while more can be added as needed.

Flexibility is offered to the model owners to feed the results of their models to several options, such as the message broker, a database, the file system, or any other requested data storage solution. Currently, most models are dependent on files as the main input and output method, therefore a strategy regarding the naming of the directories in a generated standardized and unique manner has been followed. Further options are investigated considering the future gradual incorporation of real-time data.

5 Dynamic Data Driven Application System

5.1 Technical Specifications

5.1.1 Methodology

An overview of the methodology that is used to define the technical requirements for DDDAS is shown in Figure 4. Four main components have been identified:

1. the usage (business) scenarios,
2. the key actors,
3. the assets,
4. the functional and
5. non-functional requirements.

A key actor is a human that specifies a role played by a user that interacts with the LEAD platform. An asset is an entity that communicates with/or are used by the platform during the execution or realization of a use case. Usage (business) scenarios specify a series of actions or events between an actor and a system to achieve a goal. These goals are based on functional and non-functional requirements.

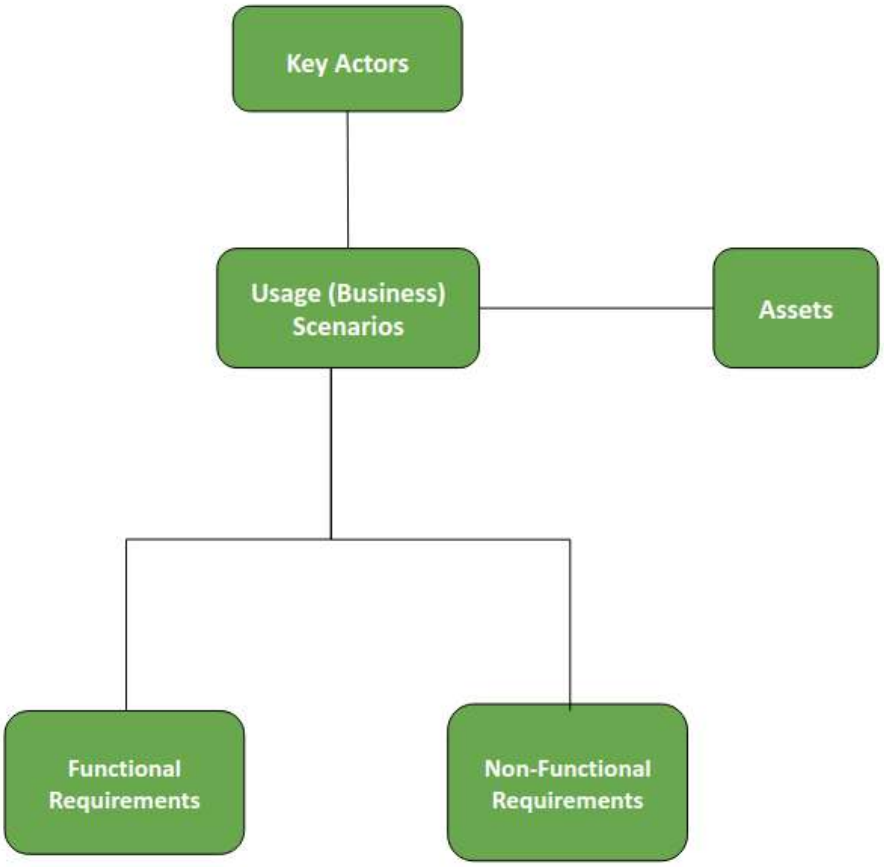

Figure 4 Methodology schema for the definition of the technical requirements

The requirements can be categorised based on their priority as displayed in Table 2. The category key is then going to be used throughout the tables that define the requirements. The categories describe the status of the requirements that have been discussed and agreed on by the stakeholders and will be formed through iterative discussions.

Table 2 Key requirement categories

Category	Key	Description
MUST	**M**	A requirement that must be satisfied for the platform to succeed
SHOULD	**S**	A high-priority item that should be included in the platform. Often a critical requirement which can be satisfied differently if strictly necessary
COULD	**C**	A requirement which is considered desirable but not necessary. It will be included if time and resources permit.

WON'T	**W**	The stakeholders have agreed that the requirement will not be implemented in the current project but may be considered for the future.

5.1.2 Usage Scenarios

Usage scenarios describe the overall user interactions with the platform. A few usage scenarios describing functionality to be incorporated in the DDDAS are listed in Table 3.

Table 3 DDDAS usage scenarios

Usage Scenario	ID	Description
Add/Remove data from the platform	A	Refers to any activities related to adding or removing data sources from the platform.
Pre-processing data ready for analysis	B	Refers to any preparation needed for the analysis of the datasets (cleansing, normalising, extracting, merging, transforming, loading)
Building Digital Twin Scenarios	C	The activities involved in visualisation and analysis using statistical, modelling, and AI-based analytics tools and code
Orchestrating Models	D	Refers to creating algorithms and workflows through applications or programming to transform raw data into statistical results
Curating & archiving data	E	The activities involved in the lifecycle of a trial from the initial brief or experimental design to the release of data from the trial as complete and final

5.1.3 Actors & Assets

A definition of the key actors and assets of the platform have been presented in Section 5.1.1. It concerns any person or system respectively that has a role interacting with the LEAD platform. The actors and assets as identified in the current state of the project is listed in Table 4 and Table 5 respectively.

Table 4 DDDAS actors

Actor	ID	Description
Data Owner	**DO**	The person responsible for data curation including classification according to GDPR and Commercial Sensitivity
Modeller	**MO**	The person responsible for constructing or programming analytical models
Analyst	**AN**	The person responsible for designing and running DT scenarios
System Administrator	**SA**	The person responsible for operational availability of the platform
User Administrator	**UA**	The person responsible for user registration and onboarding

Table 5 DDDAS assets

Asset	ID	Description
WAN	**WA**	The wide area network the platform is connected to
LAN	**LA**	The local area network associated with the platform
Data pipeline	**DP**	Data transport, transformation and storage mechanisms
Packaged Application	**PA**	Application software used by the platform (e.g., MATSim)
Custom Software	**CS**	Scripts and applications of the platform
External System	**ES**	Systems external to the platform (e.g., weather, traffic APIs)

5.1.4 Functional Requirements

Functional requirements generally describe what a system or product must do, providing in detail what is being requested by the users. The functional requirements are presented in Table 6,Table 7, Table 8, Table 9, Table 10 based on the component they concern along with the key actors, assets, and their priority category.

Table 6 Functional Requirements: Add/Remove data from the platform

Add/Remove data from the platform				
Functional Requirement	**ID**	**Key Actors Involved**	**Assets Involved**	**Category**
Add data in any agreed format to the platform	A.1	DO, MO, AN	DP	M
Provide metadata for every file (e.g., sensitivity, keyword and tags, short description, owner and uploader)	A.2	DO	N/A	S
Provide volumetric information for validation purposes	A.3	DO	N/A	S
Securely add data following the security and data encryption requirements documented in D2.3	A.4	DP	N/A	M

Table 7 Functional Requirements: Pre-processing data ready for analysis

Pre-processing data ready for analysis				
Functional Requirement	**ID**	**Key Actors Involved**	**Assets Involved**	**Category**
Ability to clean data using data cleaning techniques, tools, and programming languages (e.g., Excel, Python, R)	C.1	MO, AN	DP, PA	M
Run classical statistic analytics and models normalising, extracting features (feature engineering), transforming data	C.2	MO, AN	DP, PA	M

Export reports in a range of formats (e.g., pdfs, PPT, Excel) from unattended techniques and models	C.3	AN	WA, LA, DP, ES	M

Table 8 Functional Requirements: DT Scenario Building

DT Scenario Building				
Functional Requirement	**ID**	**Key Actors Involved**	**Assets Involved**	**Category**
Enable the linking of data sources to models	C.1	DO, MO, SA, AN	DP	M
Enable discovery of suitable models	C.2	MO	ES	M
Enable parameterization of models	C.3	MO, AN	N/A	M
Enable the linking of models into a pipeline	C.4	AN, MO	ES	M
Automation of model pipeline	C.5	MO, ES, AN	ES	S
Enable support for input thresholding	C.6	AN	DP	M
Interrupt pipeline based on thresholding	C.7	N/A	N/A	C

Table 9 Functional Requirements: Orchestration Models

Orchestrating Models				
Functional Requirement	**ID**	**Key Actors Involved**	**Assets Involved**	**Category**
Ability to create algorithms through applications	D.1	MO, AN	DP, CS, ES, PA	M
Ability to create workflows through applications	D.2	MO, AN	DP, CS, ES, PA	M

Ability to select concerned models	D.3	MO, AN	DP, CS, ES, PA	M
Ability to set configuration for simulation	D.4	MO, AN	ES	M
Ability to execute simulation	D.5	SA, UA	N/A	M

Table 10 Functional Requirements: Curating & archiving data

Curating & archiving data				
Functional Requirement	**ID**	**Key Actors Involved**	**Assets involved**	**Category**
Ability to collect data from diverse sources	E.1	DO, AN, SA	WA, LA, ES, DP, ES, CS	M
Ability to integrate data into repositories	E.2	DO, AN, ES	WA, LA, ES, DP, ES, CS	M
Ability to capture records and documents and handle them	E.3	DO, AN	WA, LA, DP, ES	M
Ability to retrieve contextual data (logistic profile)	E.4	DO, AN, SA	WA, LA, ES, DP, ES, CS	M
Export reports in a range of formats (e.g., pdfs, Excel, CSV) from archiving data	E.5	AN	WA, LA, DP, ES	M

5.1.5 Non-Functional Requirements

Non-functional requirements concern aspects of usability, security, performance, reliability, supportability, testability, and maintainability of the DDDAS. The following tables provide a description of the non-functional requirements based on such aspects along with their priority category.

Table 11 Non-Functional Requirements: Usability

Usability		
ID	**Category**	**Description**
US.1	S	Custom information screens (e.g., for adding metadata) should be designed to make user tasks and workflows intuitive and efficient
US.1	M	Save analysis and visualization outputs to local drives
US.2	M	Record and retain metadata and volumetrics information associated with a dataset
US.3	M	It must be possible to access the platform remotely
US.4	M	The platform must be accessible to all project team members

Table 12 Non-Functional Requirements: Security

Security		
ID	**Category**	**Description**
SE.1	M	Physical security to Tier 1 minimum
SE.2	M	Secure access control to the platform
SE.3	S	Unified user management for the platform with IP restrictions, and permission delegation options
SE.4	W	Single sign-on
SE.5	M	Access to platform must be supported in writing from institutional PIs (and all other PIs will be informed)
SE.6	M	Connection to LEAD intranet will use SSH
SE.7	S	Hold all data in an encrypted vault (AES128 min.)

SE.8	**S**	Offer an encrypted mail drop for the most sensitive information

Table 13 Non-Functional Requirements: Information Management

Information Management		
ID	**Category**	**Description**
IM.1	**M**	All analysis outputs should be checked back into the platform (even if saved locally)
IM.2	**S**	***Outputs are automatically classified based on*** Table 14. Status may be changed based on Data Governance meeting
IM.3	**C**	Support for distributed code repositories
IM.4	**M**	Data should be anonymised and followed the General Data Protection Regulation (GDPR)

Table 14 Output Combinations

	Public	**Private**	**Restricted**
Public	Public	Private	Restricted
Private	Private	Restricted	Restricted
Restricted	Restricted	Restricted	Restricted

Table 15 Non-Functional Requirements: Infrastructure

Infrastructure		
ID	**Category**	**Description**
IN.1	M	Platform must be protected against unauthorised intrusion and viruses

IN.2	M	Use gigabit or greater networking between platform hardware components
IN.3	S	Aggregate distributed physical resources into one, shared compute and data resource platform
IN.4	S	Support low latency, parallel processing
IN.5	C	Support long-running services
IN.6	S	The platform should support multiple Infrastructure models - bare metal, private cloud, public / hybrid cloud

Table 16 Non-Functional Requirements: Integration

Integration		
ID	**Category**	**Description**
IT.1	M	Platform is a single platform as far as user is concerned
IT.2	W	Cloud and non-cloud components need to be fully integrated
IT.3	M	An air gap is permissible between the cloud and non-cloud components during the project
IT.4	C	The platform should provide integrated application support with rich API layer

Table 17 Non-Functional Requirements: Compliance

Compliance		
ID	**Category**	**Description**
CO.1	S	The platform must comply with accessibility standards such as W3C Content Accessibility Guidelines as described in D2.3.
CO.2	M	Personal Information storage must comply with GDPR.
CO.3	M	During the project duration the freedom of Information

Table 18 Non-Functional Requirements: Operations

Operations		
ID	**Category**	**Description**
OP.1	M	SLA (99.99% availability)
OP.2	M	Service levels

5.2 System Implementation

5.2.1 Components

The entry point of every user is the dashboard which consists of a frontend and a backend. Depending on the role of the authenticated user the dashboard provides different levels of functionality. The backend takes care of the user authentication and communication with any data services (DB, message broker or filesystem) as well as the information retrieval from the model library.

The authenticated user is presented with the platform dashboard, an early mock-up of which is displayed in Figure 5 that is presented in detail in deliverable D2.1 "Technical Requirements – Solution Architecture"[9]. The dashboard communicates with the model library so that the user's selections are presented properly, the parameters set, and the prerequisites satisfied. As the user initiates a simulation (either a single execution or repetitive one) the simulation manager gathers the necessary information, sets the environment, and passes the execution to the task scheduler that is going to start the execution based on resources availability. To support such functionality, the DDDAS includes an API that is developed based on the OpenAPI 3.0 standard (**¡Error! No se encuentra el origen de la referencia.**). In this current form, the API is designed to support the definition of a context entity, the setup of a model sequence, the scenario creation, and the interaction with the simulation environment. Naturally, it is expected to evolve based on the evolving platform's needs.

The real-time data part of the application is based around a distributed event streaming platform. Data sources (such as city traffic data, product orders and CO_2 levels) are retrieved through a producer-consumer schema. Such data can be used as input to simulations together with historical data or as validation data for simulations that have been already executed. The message broker can also be used for the exchange of messages between the platform's microservices for notifications and more.

[9] *D2.1 "Technical Requirements – Solution Architecture", LEAD project.*

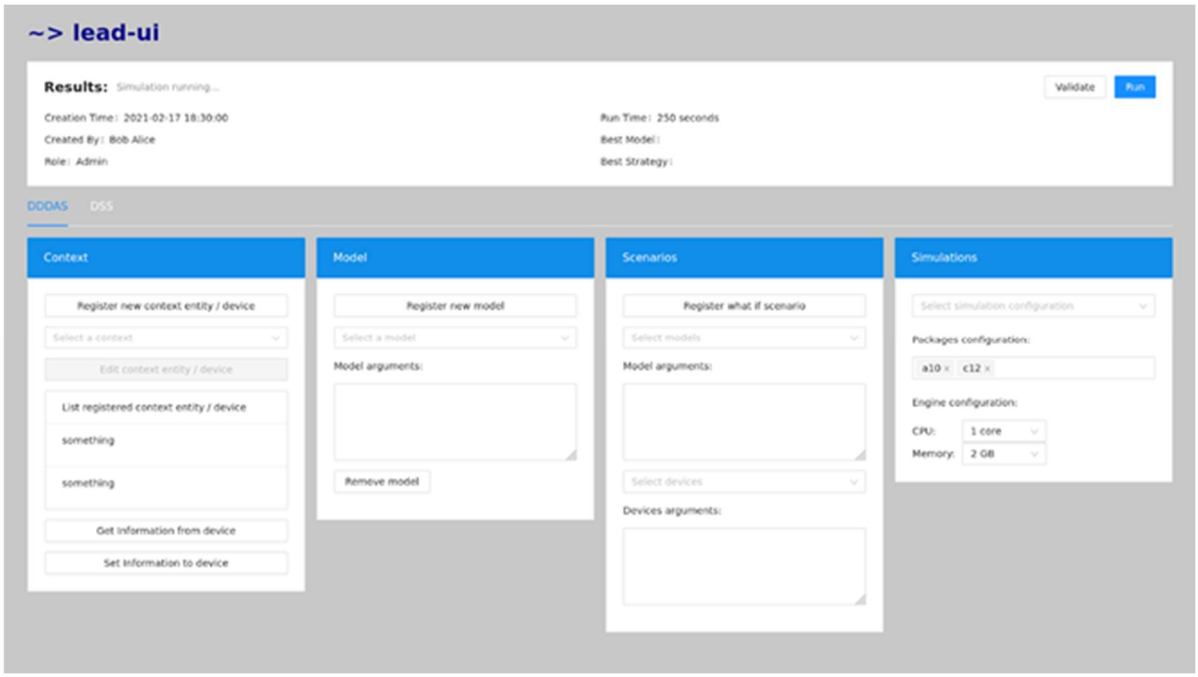

Figure 5 Early mock-up of the DDDAS user interface

The role of the simulation manager is to orchestrate all the data from the model library, the user parameters and the input data and define an execution pipeline that will be then sent to the task scheduler. The pipeline stages can vary between different models, but as a basis they consist of:

- the environment preparation (containerization is encouraged)
- the cloning of the source code (ideally from a publicly available repository)
- the retrieval of input data (and storing for later use if necessary)
- the execution of the model

Other models though are accessed through an API and not executed in the platform. In that case, the simulation manager will package the user input accordingly into a request towards the API.

The task scheduler provides several potential features to the platform. Tasks can be set to run indefinitely in a periodic fashion, the stages, and logs of a currently running task can be monitored and a history of previously executed simulations can be viewed.

Lastly, the platform's model output data are stored in a distributed file system with naming conventions to suit the models' execution and avoid data duplication. The distributed nature of the file storage provides the users with the option to further optimize their models through MapReduce and accelerate their results.

5.3 Simulation Orchestration

The simulation manager handles the retrieval of the model execution parameters and creates an Airflow DAG[10] (Directed Acyclic Graph). A DAG is a collection of the tasks that the pipeline needs to execute organized in a way that reflects their relationships and dependencies. It is written as Python

[10] *https://airflow.apache.org/docs/apache-airflow/1.10.12/.*

code and must be placed in a specific folder of Airflow. A DAG describes how a workflow is run and an Airflow Operator is the entity that defines what is executed. A variety of Operators exist such as BashOperator, PythonOperator, PostgresOperator and more covering a wide range of the models' requirements. For each model in the platform a DAG is generated based on its execution parameters. Users' parameters and input outputs are provided to the DAG dynamically at every execution.

As an example, a use case from LL3 - Lyon is going to be discussed. It concerns a MATSim[11] model that consists of a pipeline with multiple stages. First, a large amount of data needs to be available in specific locations and in a structured manner. The pipeline then starts with a BashOperator that checks the availability of the data in the file system and requests any missing or outdated data from the defined sources. Another BashOperator (and therefore a next stage of the pipeline) clones a publicly available repository, and a Python environment is created based on a Conda environment YAML[12] file. With the proper Python activated the first model execution starts. The output is stored in a specific location. Finally, a last BashOperator clones another repository, configures a Java environment, and builds the application with Maven. With the Java environment set the execution then starts using an output configuration file from the previous stage.

Finally, it needs to be stressed that such workflows are dynamic. In the example above, one or more steps for future models can be added in the pipeline following the patterns of the DAG.

[11] *https://www.matsim.org/*
[12] *https://yaml.org/*

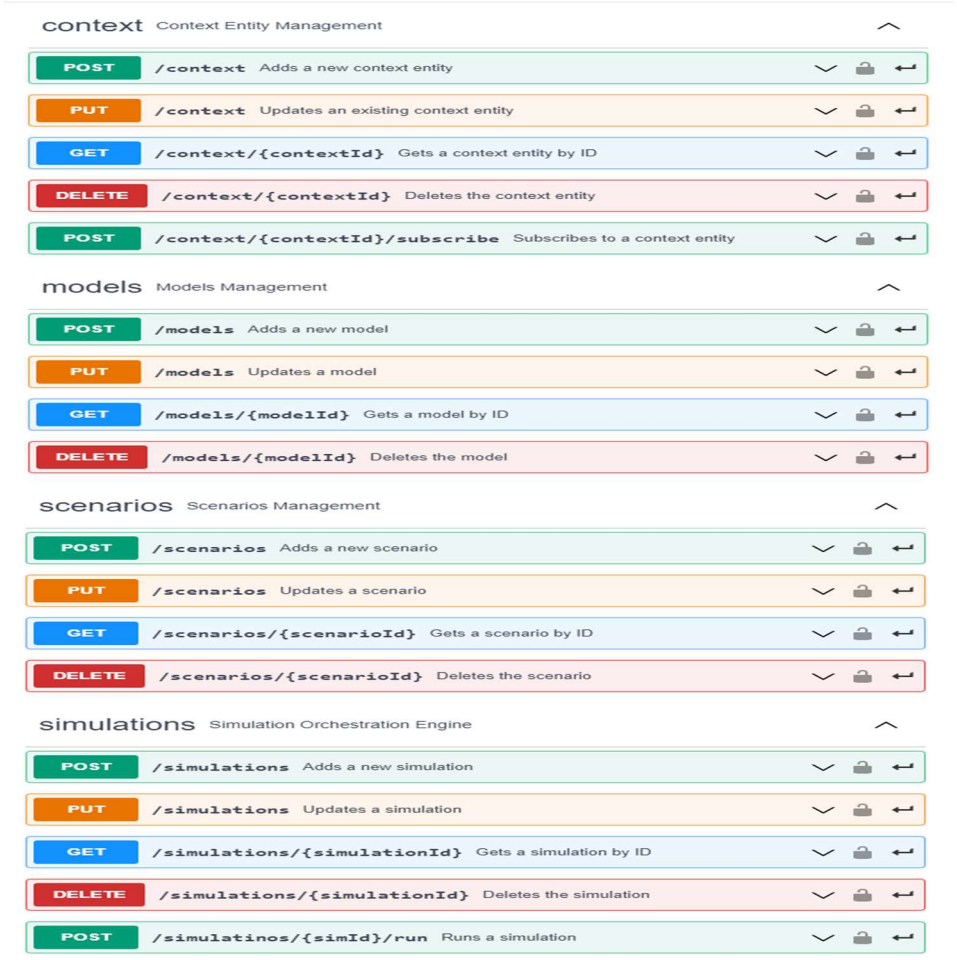

Figure 6 DDDAS API

6 Conclusion

LEAD aims to develop a digital twin platform for supporting the optimisation of last mile green logistics operations. The deliverable D2.5 describes the design and development of the technological core of the LEAD Digital Twin Platform and focuses on the Dynamic Data-Driven Application System that orchestrates the data ingestion, the digital models, the simulation and connects the Digital Twin to the physical world.

The LEAD platform is scalable, flexible, and reliable thanks to its building blocks and it is due to containerized deployment that is as presented in Section 3.3. It consists of the dashboard that is implemented using JavaScript Framework ReactJS for the frontend and a Python Flask for the backend. The storage system consists of the file system and a RDBMS while a document and timeseries databases are also foreseen. The DDDAS also queries the Model Library (ML) to access simulation models' properties and metadata. The real-time component is based on Apache Kafka distributed event streaming platform ingesting the incoming streaming data and supporting a wide variety of simulation models and LL use cases through the inherent flexibility provided by the Kafka topics as a storage solution. Furthermore, LL data connectors acting on those topics can perform any data pre-processing and augmentation needed. The DDDAS also communicates with the task scheduler, based on the Apache Airflow platform, for the orchestration of the simulations' execution.

Furthermore, to achieve a successful scenario in DDDAS after selecting the appropriate KPIs and models, input rules are important in the form of thresholds to monitor the KPIs evaluation and provide useful feedback to the users by publishing JSON messages and potentially visualising the results using Grafana. The metadata information of the output of successful scenarios are stored in the DataTypes tables of the RDBMS of the platform.

The technical specifications for the DDDAS are also presented focusing on functional and non-functional requirements that are crucial to provide a dynamic data driven system that conforms to the needs of the project's partners. The DDDAS outlook regarding the development of its components, its connections and technologies is also discussed. To meet the objectives above, we will use technologies such as Kafka, SLURM, Apache Airflow etc. Moreover, a topology of architecture and DDDAS user interfaces are shown in Section 5.2.1. The role of the simulation manager is pointed out that essentially defines an execution pipeline that will be then sent to the task scheduler. DDDAS is also connected with simulation orchestration component that handles the retrieval of the model execution parameters and creates an Airflow DAG (Directed Acyclic Graph) providing the users' parameters and input outputs dynamically at every execution.

The LEAD platform is still in an early development phase, therefore the design presented here will be enhanced and expanded based on iterative discussions with the project's partners and incremental development of features. Future developments, platform features and design modifications will be discussed in the future version D2.6 of this deliverable in M27.