A Holistic Approach to Sustainable, Digital EU Agriculture, Forestry, Livestock and Rural Development based on Reconfigurable Aerial Enablers and Edge Artificial Intelligence-on-Demand Systems

CHAMELEON D5.1 CHAMELEON Innovation Platform and Data Governance Plan v1

Work package	WP5: CHAMELEON innovation platform
Task	Task 5.1: DIP technical solution reference architecture Task 5.2: Data governance plan
Authors	**Gonçalo Rolo – Unparallel Innovation**
Dissemination level	Public (PU)
Status	Final
Due Date	30/11/2023
Document date	30/11/2023
Version number	1.0
Funded by the European Union	Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Executive Agency. Neither the European Union nor the European Research Executive Agency can be held responsible for them.

Revision and History Chart

Version	Date	Main author	Summary of changes
0.1	16/11/2023	UNP	Draft outline
0.2	20/11/2023	UNP	DIP Architecture
0.3	21/11/2023	UNP	DIP Design
0.4	22/11/2023	UNP	DIP Implementation
0.5	23/11/2023	UNP	Data Platform
0.6	24/11/2023	UNP	Integration with the DIP
0.7	27/11/2023	UNP	First version for internal revision
0.8	29/11/2013	ACCELI, AiDEAS	Internal revision
1.0	30/11/2013	UNP	Final submitted version

Table of Contents

Index of Figures

Index of Tables

LIST OF ABBREVIATIONS AND ACRONYMS

Abbreviation	Meaning
AI	Artificial Intelligence
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
DIP	Drone Innovation Platform
IoT	Internet of Things
JSON	JavaScript Object Notation
OIDC	OpenID Connect
REST	Representational State Transfer
UAV	Unmanned Aerial Vehicle
WP	Work Package

1 EXECUTIVE SUMMARY

This document reports the outcome of the first months of development of the Drone Innovation Platform (DIP).

A brief context and the motivation for the work done are presented in the introductory chapter. The next chapter focuses on the DIP itself covering topics that include the high-level architecture, the shopping cart management and the graphical design of the platform. Additionally, more technical aspects are dealt with, such as the setup of the database, the implementation of the core pages and the guidelines for integrating the Datasets Catalogue, the CHAMELEON Store and the Bundle Execution Mechanism. The last technical chapter introduces the UNPARALLEL Web Framework as the main candidate to be the source of information for the Datasets Catalogue. Finally, the conclusions are drawn and an analysis is conducted on what will be addressed as future work.

2 INTRODUCTION

Since its first breakthroughs, technology has been proving how efficient it can be in aiding human beings with their everyday tasks. With that in mind, the CHAMELEON project aims at developing the Drone Innovation Platform (DIP), a centralised web interface where the stakeholders have access to a variety of tools and functionalities whose main purpose is to provide them with guidance on how to tackle a specific problem.

The technological assets involved are Unmanned Aerial Vehicles (UAVs), responsible for collecting images of targetted geographical areas, and Artificial Intelligence (AI) models that process and interpret the images in order to generate the guidelines for the users, commonly referred to as Bundles in the context of the project.

Given that the main domains that fall under the scope of the project are Agriculture, Forestry and Livestock, the problems covered by the CHAMELEON solution include “Crop growth and development monitoring”, “Vineyard water stress due to drought”, “Monitoring flora at high-altitude grazing areas for seasonal animal feeding”, “Collecting parameters related to the health and stress of livestock”, “Lameness detection in cows”, “Herd management and individual monitoring”, “Vegetation monitoring and census”, “Monitoring humidity of soil and plants for assessing risk of fire”, “Hotspot identification at the beginning of wildfire”, “[Identifying] Large woody debris on rivers”, “Health status of vegetation (mainly bark beetle), game browsing, ground cover, and fungal growth”, amongst many others.

This document focuses on the developments that led to the first version of the DIP, and is organised as follows. The first chapter delves into the several details of the platform, starting with the layers of the system’s architecture. Afterwards, the graphical design of the solution is presented, followed by the actual implementation aspects. This chapter is closed with a section dedicated to the integration of the main components. In its turn, the second chapter introduces the UNPARALLEL Web Framework as the source of information for the Datasets Catalogue. The document ends with the conclusions derived from the work that was carried out, along with the steps to be taken in the short-term future.

3 DRONE INNOVATION PLATFORM

The Drone Innovation Platform, hereinafter referred to as DIP, is a webplatform that intends to provide a set of Bundles, i.e., Artificial Intelligence (AI) models fine-tuned to tackle a specific problem within the Agriculture, Forestry or Livestock domains. Since the bundles are fed with images, a catalogue of Datasets must be available in the platform as well. Furthermore, the execution of Bundles must be managed so that the user can monitor the whole process within a single graphical interface.

The main purpose of the current chapter is to report the work developed regarding the DIP. To that end, the architecture is first described, from a top-down perspective, starting with the high-level architecture before delving into the shopping cart specific flow, responsible for managing pending orders. Afterwards, the design of the pages is explained through a set of mock-ups, along with the technical aspects behind enforcing the different developers to comply with the established graphical rules. The following section is dedicated to the implementation, where both the used technologies and the project structure are introduced. Finally, the integration strategies for the different components of the platform are described.

3.1 ARCHITECTURE

3.1.1 HIGH-LEVEL ARCHITECTURE

With the aim of providing the platform with all the expected functionalities, its components were identified, as well as the interactions between them.

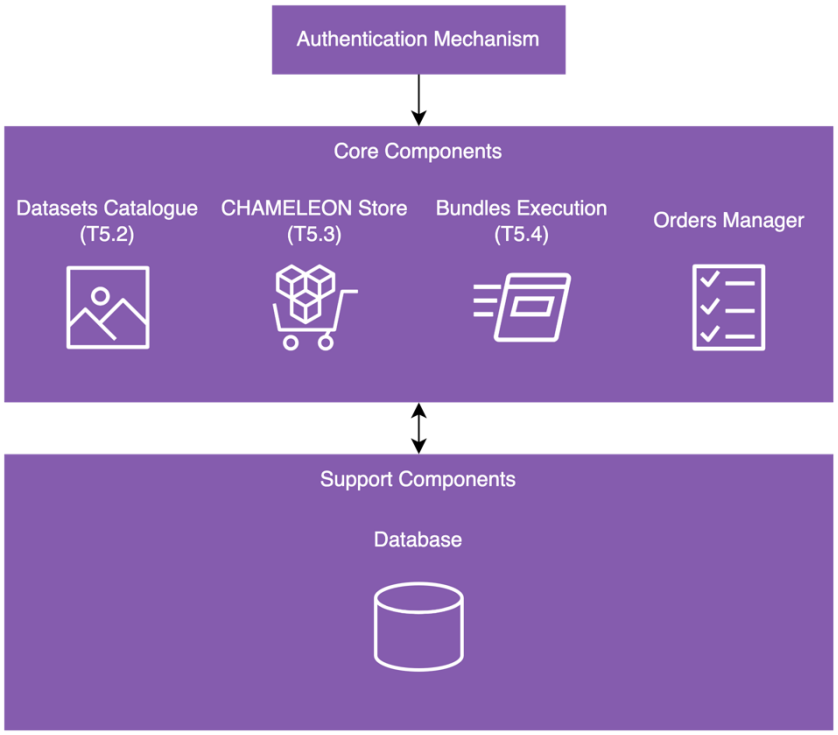

Figure 1: Architecture of the DIP.

According to Figure 1, the DIP includes three types of components.

Firstly, there is the authentication mechanism, whose goal is to prevent unauthorised users from accessing the content of the platform.

The second group – dubbed "Core Components" – contains the main components, each of which will be integrated with DIP, providing a vital feature:

- The **Datasets Catalogue** is where the user can search for images and other types of input to feed their bundles with. Since this component relates to task T5.2, further details are explored in section 4 of this document.
- The **CHAMELEON Store** displays the list of bundles developed in Work Package 4 of the project, and enriched on Open Calls, allowing the user to find the desired bundle in the most user-friendly possible manner, either through filters, sorting and matchmaking algorithms. This component is developed under the scope of task T5.3, so please refer to deliverable D5.2 for more information.
- The **Bundle Execution** module is responsible for managing the execution of a bundle from the moment it is triggered by the user until it finishes successfully, through a queueing system. This is part of the integration work developed in task T5.4, therefore more details can be expected in D5.3.
- The concept of *Order* was created so that users can register their request to execute a given bundle with the selected images as input, and then assess the outcome of that run. The **Orders Management** module thus provides an intuitive interface for users to manage add, track and remove their orders.

Finally, a third group includes the so-called "Support Components" which, as of now, consist of the database where the orders and related information are stored.

3.1.2 SHOPPING CART

From all the possible interactions in the DIP, the flow that leads to an order being registered in the platform is the most complex one.

As briefly described above, to add an order the user must:

- Select a bundle, i.e., an AI model that processes a given input and whose output helps the user deciding how to tackle a certain problem.
- Configure the parameters exposed by the selected bundle.
- Choose a set of images in the appropriate format to be processed by the selected bundle.

However, it was decided not to force the user to make these selections and configurations in a particular order, but to follow a shopping cart logic instead. This means that the only dependency between these steps is that the parameters can only be configured after selecting a bundle, as different bundles may require different parameters to be fine-tuned. It is therefore possible to select the bundle and the image(s) at any given time and, regardless of the order in which these actions are executed, as soon as both elements are added to the shopping cart and the bundle parameters have been fine-tuned, the user can sign the order up for execution.

The diagram below describes the flow that leads to having the images added to the shopping cart.

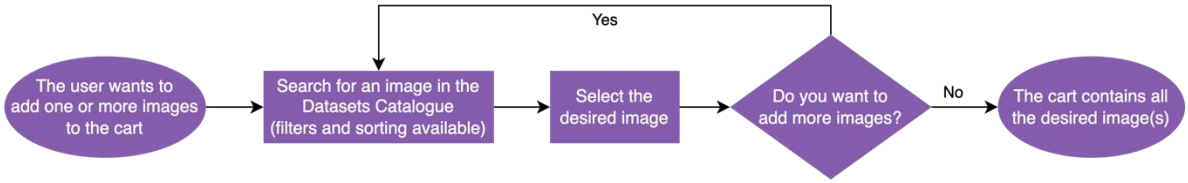

Figure 2: Flow to add one or more images to the cart.

As depicted in Figure 2, whenever the user wants to add one or more images to the cart, they only need to go to the Datasets Catalogue and search for images making use of the available filters and sorting. As soon as they find suitable images, the user selects them, by which time the images are added to the cart. The process can be repeated until the cart contains all the needed images.

The diagram below portrays the flow to select a bundle and add it to the cart.

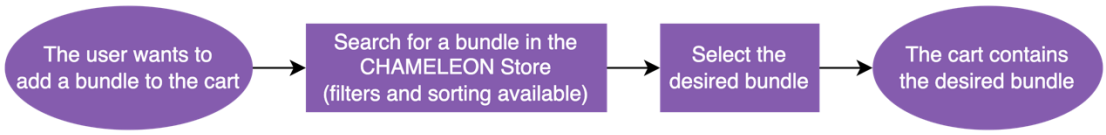

Figure 3: Flow to add a bundle to the cart.

According to Figure 3, in order to add a bundle to the cart, the user must visit the CHAMELEON Store page in the DIP and search for a suitable bundle (filters and sorting may be used as well). Once they find it, the user adds the bundle to the cart by selecting it.

At this point, the user may register the order, i.e., request the execution of the bundle.

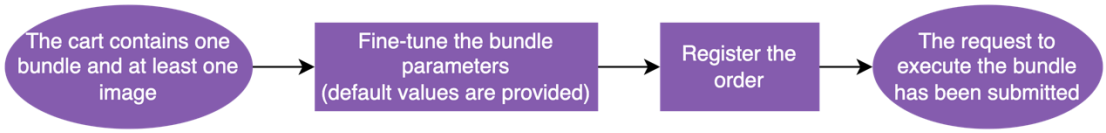

Figure 4: Flow to register the order.

Figure 4 shows that, after the selection of a bundle and a set a set of images, the remaining step is to fine-tune the parameters that the bundle allows users to configure. Afterwards, the user may finally submit their request to execute the bundle.

3.2 DESIGN

Having finalised the specification of the DIP, the design of its frontend, which comprised two main topics, was carried out. This section thus contains details about the mock-ups that were drawed and the development of the corresponding Bootstrap [1] theme, both vital for the development of the DIP and to ensure coherent and visually pleasant integration of all modules and flows.

The first mock-up to be designed was that of the first page that the user comes across when visiting the DIP, the login page.

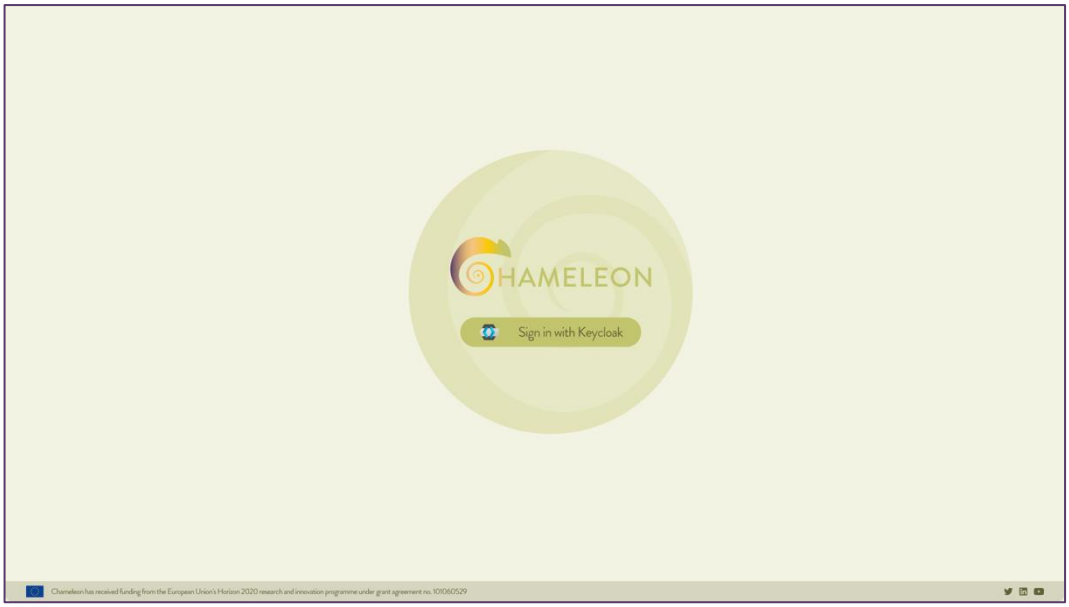

Figure 5: Login page.

As depicted in Figure 5, this page contains the button that triggers the authentication process before the user can access the functionalities of the platform. Following the insertion of valid credentials, the homepage is displayed.

[1] https://getbootstrap.com

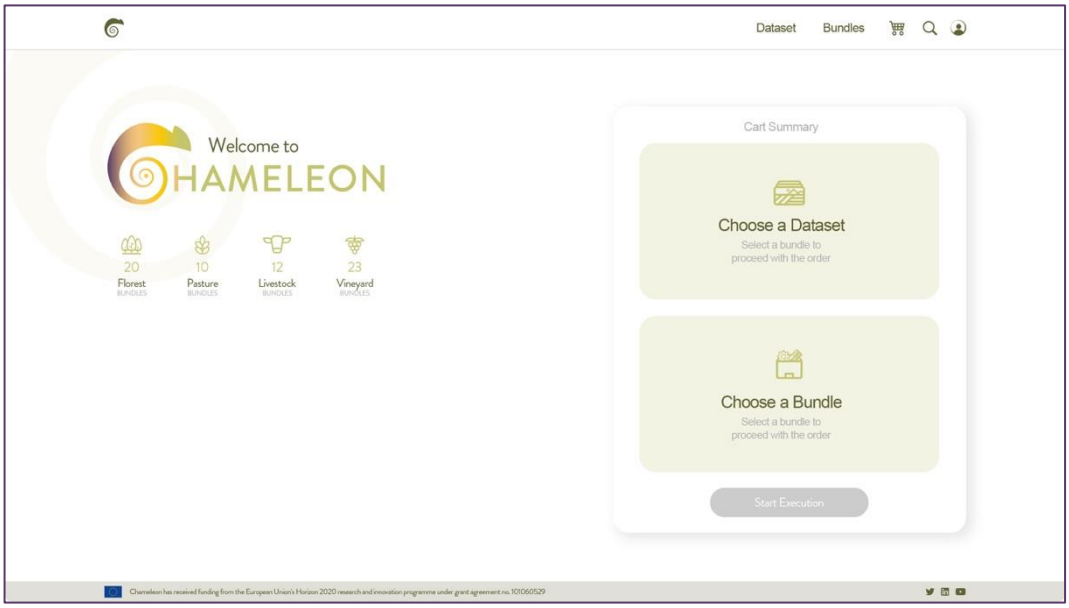

Figure 6: Homepage.

The homepage, represented in Figure 6, establishes the layout of the platform with three main components:

- The header contains: the CHAMELEON logo that can be clicked at any time to go back to the homepage; links to the Datasets Catalogue ("Dataset") and the CHAMELEON Store ("Bundles"); an icon to preview the cart; a magnifying glass to access the search feature across the entire platform; a clickable user icon that displays the name of the logged in user and the option to log out from the DIP.
- The main content section, in the middle of the page, shows the number of bundles available in the CHAMELEON Store by category (Forest, Pasture, Livestock and Vineyard) and quick access to both the Datasets Catalogue and the CHAMELEON Store in order to interact with the shopping cart. Notice how the "Start Execution" button is greyed out whenever no bundle or images have been added to the cart.
- The footer provides the project information.

Clicking the "Choose a Dataset" button, in the Cart Summary bar on the right-hand side of the main content section, takes the user to the Datasets Catalogue. The result is shown in the figure below.

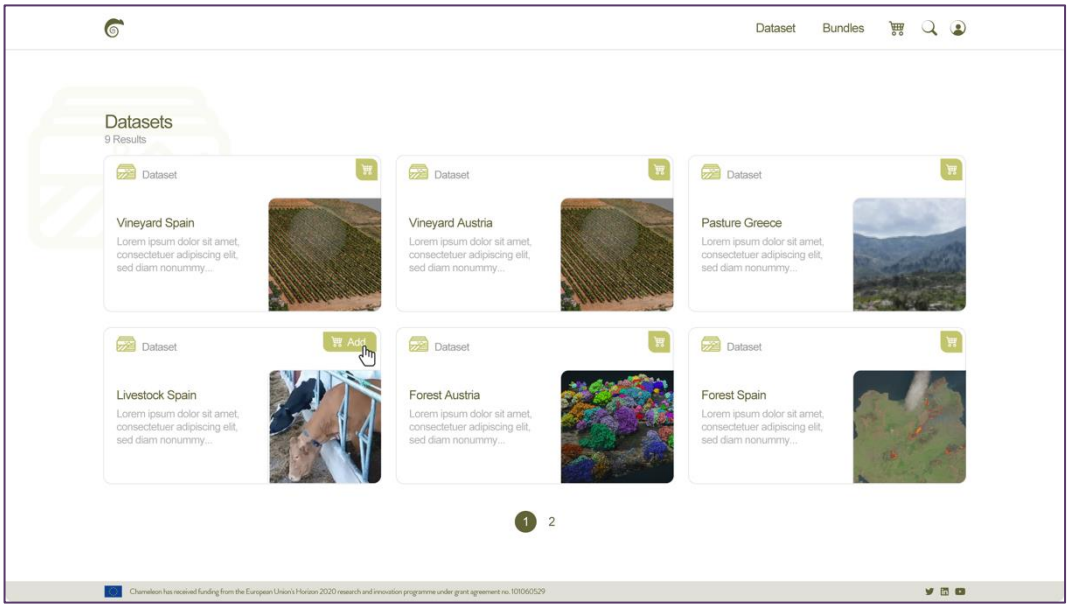

Figure 7: Datasets Catalogue page.

As represented in Figure 7, this page allows users to search datasets, see related-details and add them to the cart.

In its turn, the "Choose a Bundle" button leads to the CHAMELEON Store. As depicted in Figure 8, it allows users to navigate throughout the available bundles, read their descriptions and select the most suitable one for the problem they wish to obtain a solution for.

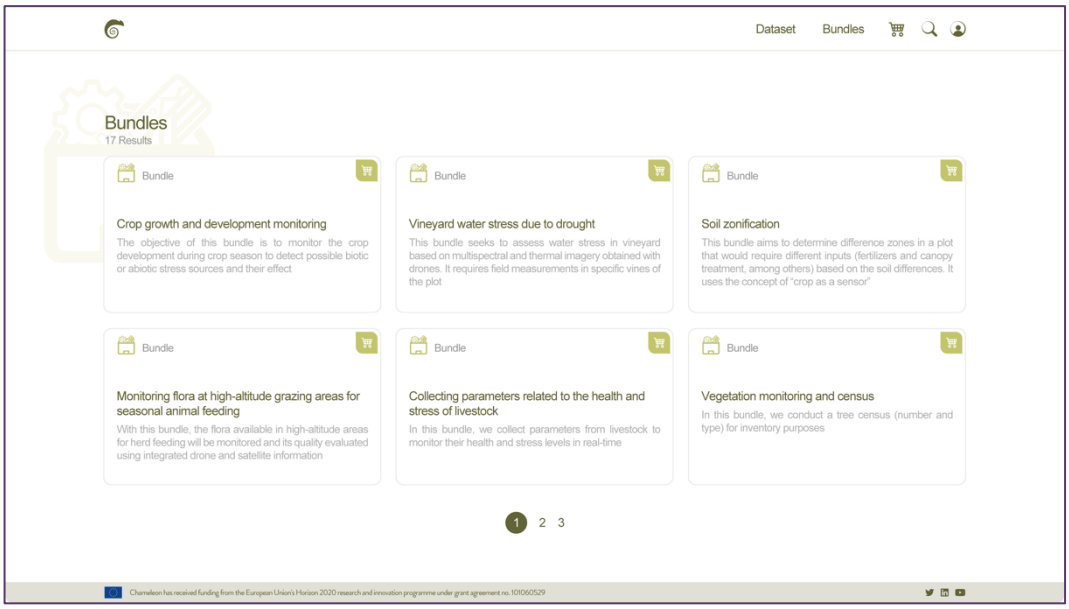

Figure 8: CHAMELEON Store page.

Once the cart has at least one dataset and one bundle, the "Start Execution" button becomes available. Figure 9 depicts how the Cart Summary accurately represents the content of the cart so that the user can confirm whether the correct images and bundle have been selected.

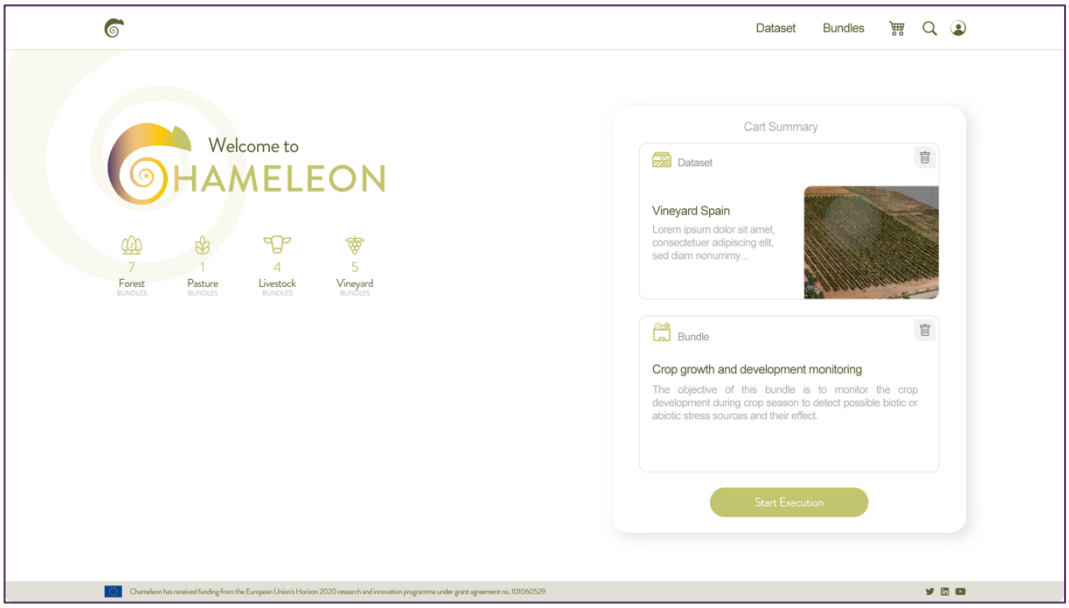

Figure 9: Cart Summary.

Provided that that is the case, when the user clicks the "Start Execution" button, the "Cart Details" panel opens to the left of the "Cart Summary", unravelling further information about the selected items, as portrayed by Figure 10.

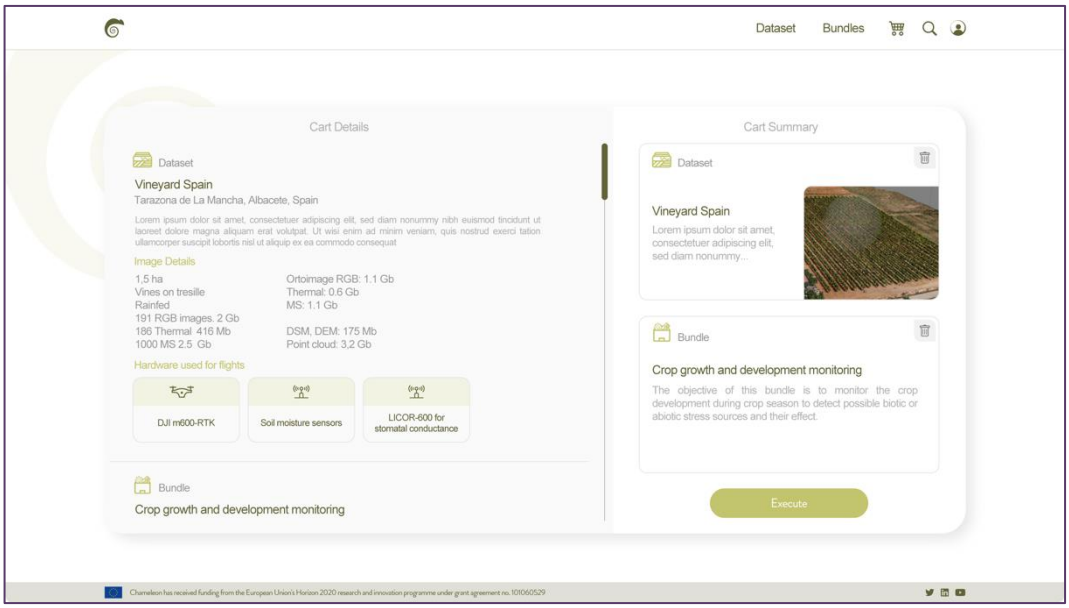

Figure 10: Cart Details panel.

In case the details exceed the height of the page, scrolling is available as well. In this case, by doing so the user comes across the "Configure Parameters" button under the Bundle section (see Figure 11).

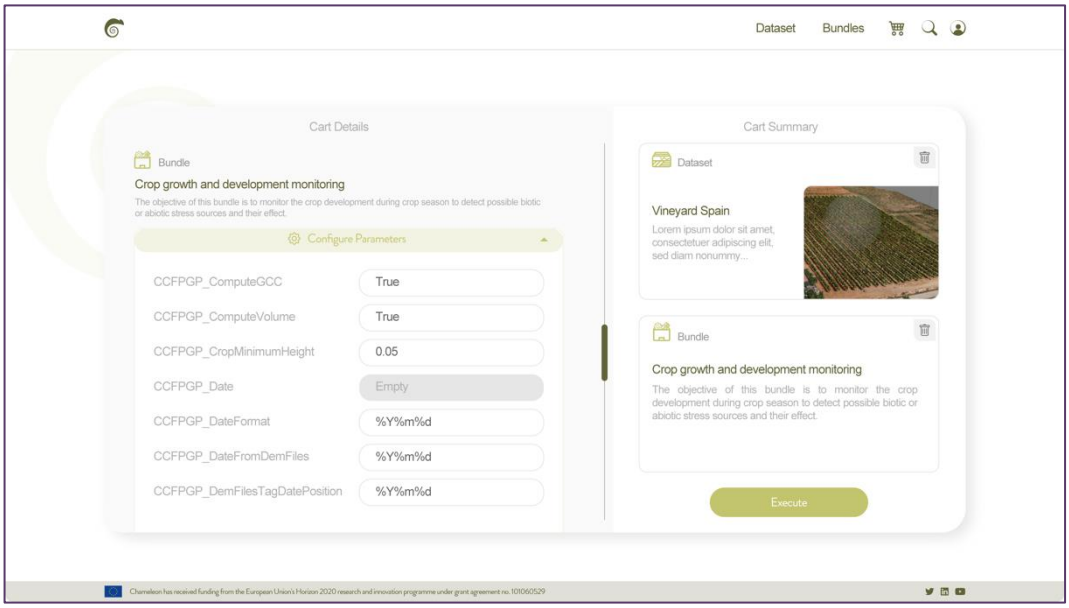

Figure 11: Bundle configuration section.

These parameters have default values but the user is free to fine-tune them before signing the order up for execution.

Finally, it is expected that, once the execution finalises, the user receives a notification, as shown in Figure 12.

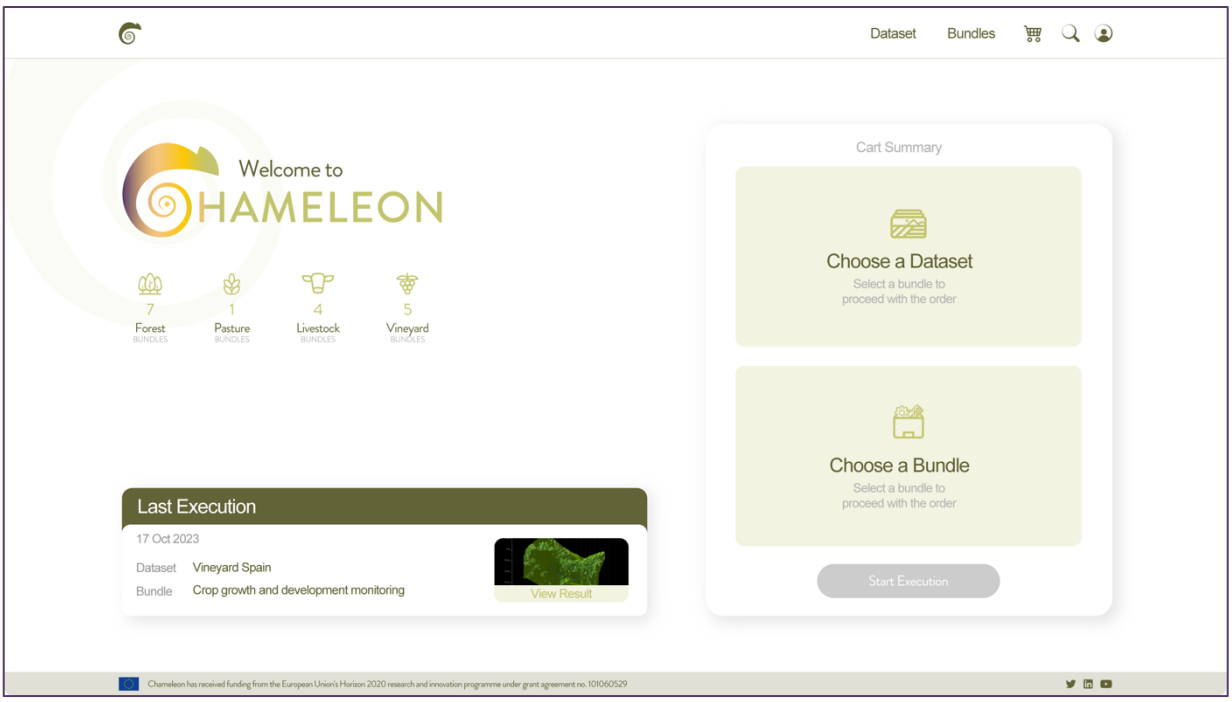

Figure 12: "Bundle execution finished" notification.

With the aim of making sure that all contributors of the project have access to the same colour palette and other theme-related variables, a Bootstrap theme was developed and shared with the technical partners.

3.3 IMPLEMENTATION

The present section focuses on the technical aspects of the development of the DIP from both the backend and the frontend perspectives.

3.3.1 DATABASE

With the goal of storing the information related with the orders, a MongoDB instance was set up. Taking the aforementioned behaviours and functionalities into account, multiple iterations of the database diagram were designed, with the latest version being represented in Figure 13.

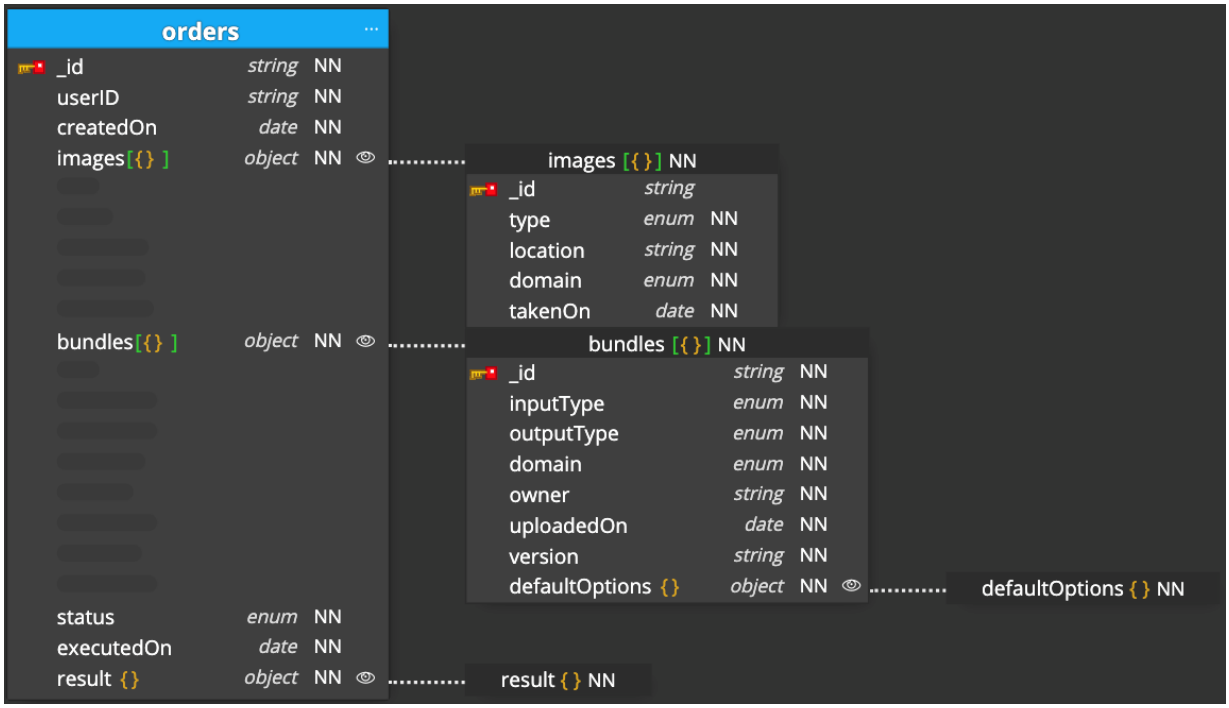

Figure 13: DIP database diagram.

The central entity of the platform and, consequently, the database is the *orders*. It includes:

- *_id* – the ID of the order in the database.
- *userID* – the ID of the user who registered the order.
- *createdOn* – the date when the order was created.
- *images* – an array with the IDs of the images that were added to the order.
- *bundles* – an array of objects containing the IDs of the bundles that were included in the order and corresponding parameters. Even though only one bundle per order is expected at the moment, this field supports an array so it is possible to run several bundles sequentially, in the future.
- *status* – the current status of the order as per Table 1.
- *executedOn* – the date when the order execution finished.
- *result* – the output of the bundle execution.

Table 1: Order statuses.

Status	Description
PENDING	The order has not been submitted by the user. This allows the user to resume the last unfinished order, regardless of the device they are on.
TO_BE_EXECUTED	The order has been submitted by the user and is waiting in the queue to be executed.
EXECUTING	The bundle execution is in progress.
EXECUTED	The bundle execution has finished and the results can be accessed.

Another relevant entity in the context of the DIP is the *images*, described by:

- *_id* – the ID of the image in the database.
- *type* – the type of image, useful for filtering purposes in the Datasets Catalogue and to assess the compatibility with the bundles (e.g., *TIFF*).
- *location* – the location where the image was taken, also useful for filtering the images.
- *domain* – the domain within the CHAMELEON project to which the image can be applied (e.g., *FORESTRY*, *PASTURE*, *LIVESTOCK* and *VINEYARD*). Not only can this be used to filter the images but also to assess the compatibility with corresponding bundles.
- *takenOn* – the date when the images were collected.

Finally, there is the *bundles* entity comprising:

- *_id* – the ID of the bundle in the database.
- *inputType* – the type of images that the bundle can process (e.g., *TIFF*).
- *outputType* – the format that the bundle outputs its results in (e.g., *SHP*, *JSON*, *CSV* and *PDF*).
- *domain* – the domain within the CHAMELEON project to which the bundle can be applied (e.g., *FORESTRY*, *PASTURE*, *LIVESTOCK* and *VINEYARD*). Not only can this be used to filter the bundles but also to assess the compatibility with corresponding images.
- *owner* – the name of the bundle's owner.
- *uploadedOn* – the date when the bundle became available at the CHAMELEON Store.
- *version* – the version of the bundle.
- *defaultOptions* – an object containing key-value pairs, identifying the configurable bundle parameters and corresponding default values. Since the user is allowed to tweak these values, this object must be stored in the array of bundles of an order together with the bundle ID.

Notice how, in this diagram, the *images* and *bundles* are not at the same level as the *orders*. This represents the fact that it is not mandatory to store their data in the same instance of the database as the *orders*. In any case, it is of utmost importance that the IDs stored in the *images* and *bundles* fields of an order match the *_id* value stored in whichever database they are on.

3.3.2 FRONTEND

Given that the DIP has several partners contributing to its frontend, a collaborative frontend development framework was prepared under the scope of task T5.1, with the aim of providing a common baseline for the developers to implement their part of the graphical interface.

The DIP is based on Next.js, a React framework that smoothens the process of creating full-stack Web applications. On top of that, the NextAuth.js library is used to enforce the authentication and authorisation mechanisms, namely using Keycloak as the OpenID Connect (OIDC) provider. The repository containing the previously described infrastructure can be consulted at https://gitlab.com/unparallel-open-source/chameleon-dip.

HELPER

From the source code standpoint, it is relevant to report the creation of the so-called Helper. This library consists o a wrapper for all the methods that can be used by the developers to interact with both the authentication tool (e.g., getting session data such as the name of the logged in user) and the database (e.g., CRUD operations applied to the *orders* entity). Using the Helper is as simple as importing the *lib/helper* folder into the desired React component. The methods made available by the Helper are summarised in Table 2.

Table 2: Helper methods.

Category	Method	Description
Keycloak	*getKeycloakSession*	Returns the Keycloak session object.
	getKeycloakToken	Returns the Keycloak session token.
	getKeycloakUserID	Returns the Keycloak ID of the logged in user.
MongoDB (Bundles)	*getBundles*	Returns the bundles present in the database, according to the provided filters.
	getBundleDefaultOptions	Returns the default options of a bundle, given its ID.
MongoDB (Images)	*getImages*	Returns the images present in the database, according to the provided filters.
MongoDB (Orders)	*addOrder*	Adds the given order to the database, associated to the logged in user.
	getOrders	Returns the orders of the logged in user present in the database, according to the provided filters.
	getCurrentOrder	Returns the order of the logged in user whose status is *PENDING*.

Category	Method	Description
MongoDB (Orders)	*updateOrder*	Updates the order with the provided information (e.g., bundle, images).
	addBundlesToOrder	Adds the bundle(s) with the provided ID(s) to the order.
	updateBundleOptions	Updates the bundle options selected for the current order, given the bundle ID.
	removeBundleFromOrder	Removes a bundle from the current order, given its ID.
	addImagesToOrder	Adds the image(s) with the provided ID(s) to the order.
	removeImageFromOrder	Removes an image from the current order, given its ID.

SHOPPING CART MANAGEMENT

The other main value added by the first version of the platform is the implementation of the shopping cart behaviour. Similarly to what is expected from the other components of the DIP, the shopping cart management flow was developed making use of the Helper methods that were previously introduced.

The first method to be used as soon as the user logs into the platform is the *getCurrentOrder*. This returns the content of the order whose status is *PENDING*, allowing the user to resume the selection of the bundle and/or images regardless of the device they are on. If the user doesn't have any order with the *PENDING* status, an new order is created.

Whenever the user visits the Datasets Catalogue, the *getImages* method is invoked to obtain the list of available images. Likewise, the *getBundles* method is used every time the CHAMELEON Store is accessed, so that the updated list of Bundles is communicated to the frontend and displayed to the user.

Once the user has entered either of these pages, selecting an item there triggers, in the first place, the *addOrder* method (in case no pending order exists) and then the *addImagesToOrder* or the *addBundlesToOrder* methods, respectively. It goes without saying that unselecting an image invokes *removeImageFromOrder*, where as unselecting a bundle calls the *removeBundleFromOrder* method.

After finalising the selection of items for the order, the user must configure the bundle parameters. The interface that enables this configuration is populated based on the results of the *getBundleDefaultOptions* method, i.e., the key-value pairs with the name of each parameter and corresponding default value. When the bundle is added to the order, its default options are stored along with the ID of the bundle and, as the user fine-tunes the parameters, their values are updated in the database through the *updateBundleOptions* method.

There are two other methods also worth mentioning, despite not being directly involved in the shopping cart flow:

- *getOrders* returns all the orders, taking into consideration the provided filters. A possible use case for this method is to obtain data to populate a page where the orders of the logged in user are summarised.
- *updateOrder* updates the order with the content that is sent in the body of the request. This method is used in the implementation of *updateBundleOptions*.

3.4 INTEGRATION

In order to prepare the common baseline for the other developers to contribute, some placeholders had to be used (e.g., a dummy page for the CHAMELEON Store). This does not mean that there is no implementation of such components, but that it is still not integrated with the DIP.

Since there are several developers contributing to the platform, it is of utmost importance to have clear guidelines as to how the information exchanging takes place, where does one developer's responsibility ends and where is the other developer supposed to take charge. That is precisely what this section focuses on.

3.4.1 DATASETS CATALOGUE

Starting with the integration of the Datasets Catalogue, the guidelines below must be followed:

- Reimplement the *getImages* method from the Helper, taking into account that:
  - Two parameters must be available: *filters* (object containing MongoDB query-like filters) and *projection* (object containing MongoDB query-like projection).
  - The method must return an array of objects, where each one is an image according to the database defined in 3.3.1.
- Invoke the *addImagesToOrder* method from the Helper, considering that:
  - *imageIDs* (array of strings containing the IDs of the images to be added to the order) parameter must be received.
  - The method must return the result of the operation (the result of the MongoDB *updateOne* method is currently being returned).
  - The *updateOrder* method from the Helper may be used.
- Invoke the *removeImageFromOrder* method from the Helper, considering that:
  - *imageID* (string containing the ID of the image to be removed from the current order) parameter must be handled.
  - The method must return the result of the operation (the result of the MongoDB *updateOne* method is currently being returned).
  - The *updateOrder* method from the Helper may be used.

Replacing these methods will establish a communication channel between the shopping cart of the DIP and the Datasets Catalogue, which feeds the platform with datasets.

3.4.2 CHAMELEON STORE

In order to integrate the CHAMELEON Store with the DIP, the following steps must be completed:

- Reimplement the *getBundles* method from the Helper, considering that:
  - Two parameters must be available: *filters* (object containing MongoDB query-like filters) and *projection* (object containing MongoDB query-like projection).
  - The method must return an array of objects, where each one is a bundle according to the database defined in 3.3.1.
- Reimplement the *getBundleDefaultOptions* method from the Helper, taking into account that:
  - *bundleID* (string containing the ID of the bundle whose default options we want to retrieve) parameter must be handled.
  - The method must return an object with the default options of the given bundle.
- Invoke the *addBundlesToOrder* method from the Helper, knowing that:
  - *bundleIDs* (array of strings containing the IDs of the bundles to be added to the order) parameter must be received.
  - The method must return the result of the operation (the result of the MongoDB *updateOne* method is currently being returned).
  - The *updateOrder* method from the Helper may be used.
- Invoke the *removeBundleFromOrder* method from the Helper, considering that:
  - *bundleID* (string containing the ID of the bundle to be removed from the current order) parameter must be handled.
  - The method must return the result of the operation (the result of the MongoDB *updateOne* method is currently being returned).
  - The *updateOrder* method from the Helper may be used.

These guidelines will ensure that the CHAMELEON Store fits into the current implementation of the DIP and both elements are able to communicate successfully.

3.4.3 BUNDLE EXECUTION

The Bundle Execution will be conducted by a Job Orchestration and Deployment mechanism responsible for managing the requests of all users, as portrayed in Figure 14.

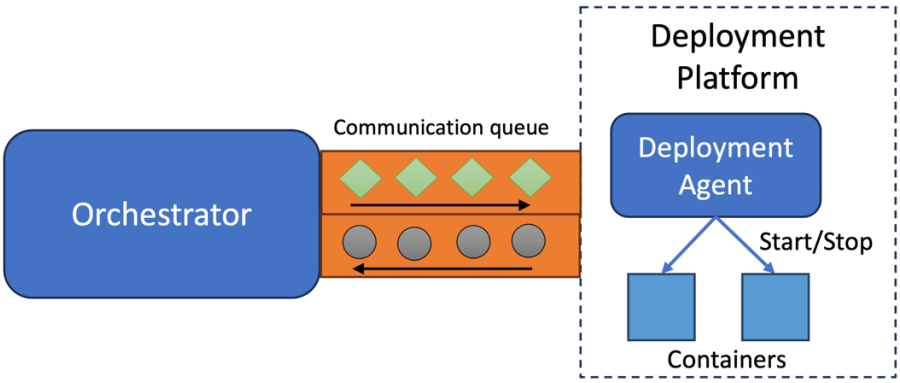

Figure 14: Mechanism for Job Orchestration and Deployment.

The operation of this mechanism can be summarised by the steps below:

1. The orchestrator receives all job requests and adds them to the communication queue using a well-defined protocol.
2. The deployment agent analyses the jobs on the queue and launches the services required to process them or stops services when they are no longer needed.
3. When launched, the services consume the corresponding jobs and add the corresponding results to the queue.

It is worth mentioning that this communication queue protocol is based on the one used by the IoT-Catalogue.com[2] to dispatch work for external services.

The Bundle Execution Mechanism will ensure a smooth flow of requests and responses between the DIP and the CHAMELEON infrastructure, which will host all the project services.

[2] https://www.iot-catalogue.com

4 DATA PLATFORM

This chapter aims at introducing the UNPARALLEL Web Framework, which will be used as the source of information for the Datasets Catalogue. The UNPARALLEL Web Framework is a well-established outcome of several UNPARALLEL Innovation's developments during the past years.

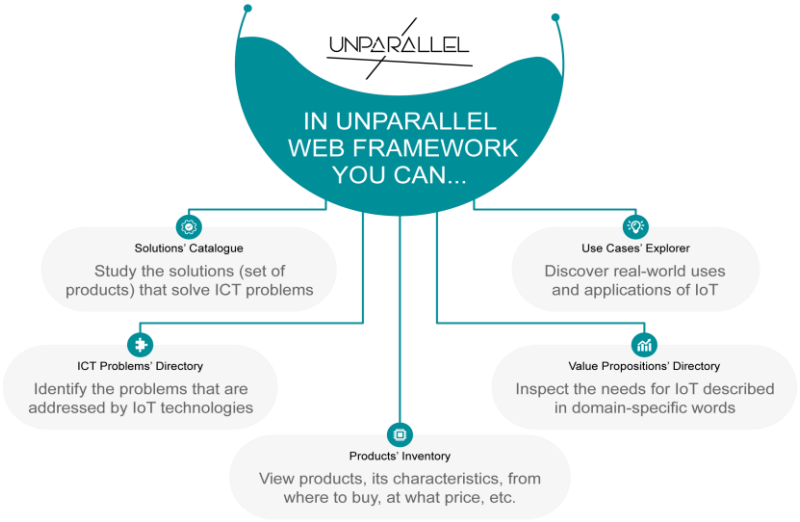

Figure 15: UNPARALLEL Web Framework.

The UNPARALLEL Web Framework, with its content structuring capabilities and already being used in company products such as the IoT-Catalogue.com, is perfectly suitable for leveraging the content of the Datasets Catalogue. The UNPARALLEL Web Framework serves as a backend base of information, providing the Datasets Catalogue with data on components and datasets gathered under the scope of this project, through the following functionalities:

i) **Components**

Provides information about several components from simple sensors with common functions to more complex instrumentation delivering out-of-the-box technological solutions, with the possibility to apply filters when listing them by type, location, and so on.

When opening a component, detailed information is displayed, including its characteristics, purchase options, price, name, description, etc. Several interactive bars are also provided, contributing for an intuitive navigation within the component page and through the other elements related to it.

Within the CHAMELEON context, this functionality is useful to model the Unmanned Aerial Vehicles (UAVs) that were used to collect the datasets, i.e., images.

ii) **Datasets**

Displays information about large amounts of data in an organised and intuitive manner, including the Measurable Quantities represented in the dataset, the location in a map view, a chart representation of the values for numeric datasets, amongst many others.

iii) **Usage in IoT-Catalogue.com**

Figure 16: IoT Catalogue logo.

UNPARALLEL's IoT-Catalogue.com application makes extensive use of the UNPARALLEL Web Framework to provide information related with the Internet of Things (IoT), including several technological products and its applicability on use cases which are part of several EU Research projects. This application is used by partners working on EU Research projects, as it provides an organised way to consult the information about a project where they are working on and to find other projects sharing the same solutions and products, enabling the linking between partners of different projects.

So far, IoT Catalogue was used or is still being used in the following projects: AURORAL, Boost 4.0, EUR3KA, FACTLOG, FAR-EDGE, ICP4Life, i3-MARKET, INFINITECH, IoF2020, KYKLOS 4.0, MONICA, PROPHESY, Qu4lity, SmartAgriHubs, Thomas, VICINITY, and WAZIUP. The latter, already finished, was the first project to be included in IoT-Catalogue.com.

4.1 DATASETS

4.1.1 DATASETS EXPLORER

DATASET MAP VIEW

The map view (Figure 21) provides an overview of all Dataset locations, with the datasets listed changing according to the user interaction with the map. The user may also select Measurable Quantities to filter the results.

DATASETS

List view of datasets containing information about all data collections related with the selected data concept. If nothing is selected all datasets currently visible on the map are displayed by default. The list is updated as the user navigates and changes the map view.

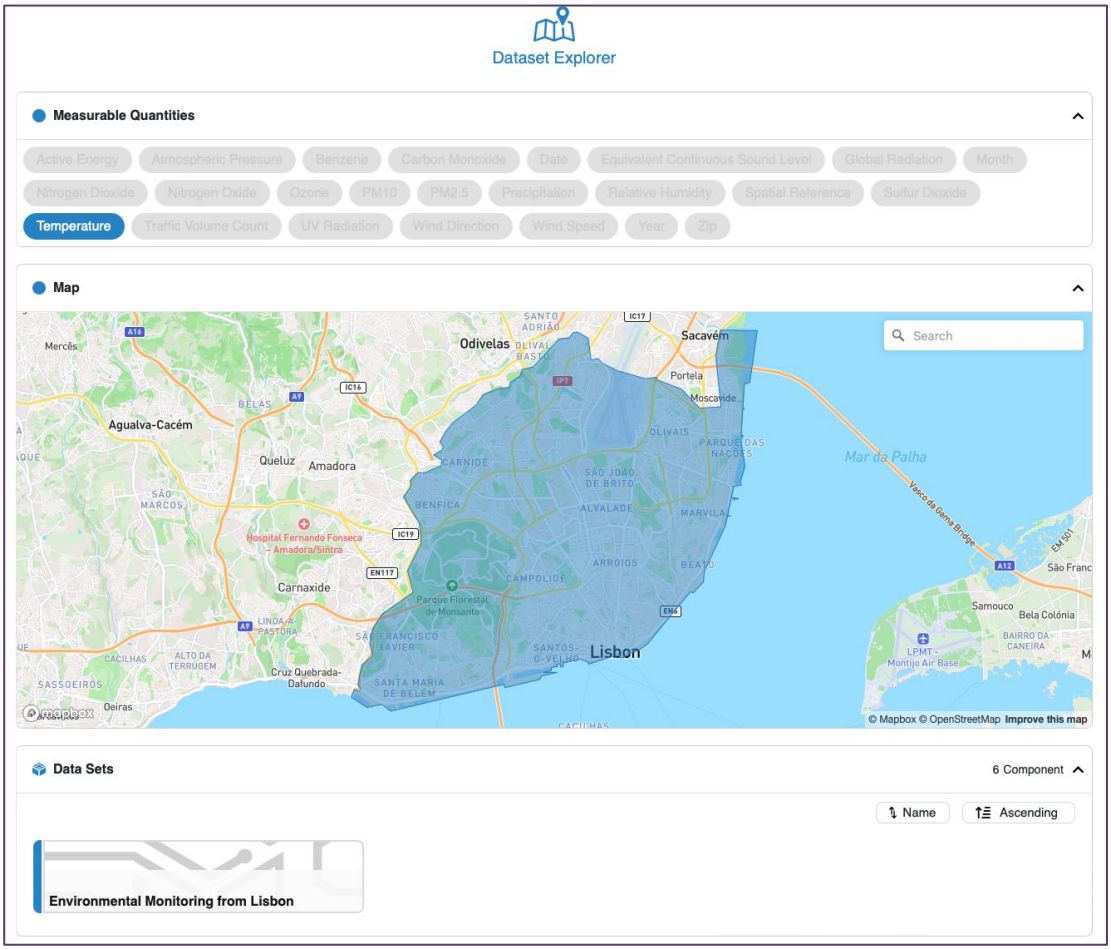

Figure 17: Interactive map view.

The view provides a clickable dataset listing using cards and geographical points, selecting a dataset will open the corresponding page with more details and the latest measurements, the concepts it captures as well as the Measurable Quantities of the values it contains.

4.1.2 DATASETS VISUALISER

Currently two types of data can be visualised: time-series data (Figure 18) and georeferenced data (Figure 19). Time-series data is shown on a graph with a map on the left-hand side displaying the location of data collection (if such information is provided). Selecting a Measurable Quantity on the top bar changes the locations that are displayed on the map to those that have values for the selected Measurable Quantity. Then, selecting a dot updates the graph on the right-hand side to display the time-series data for that location.

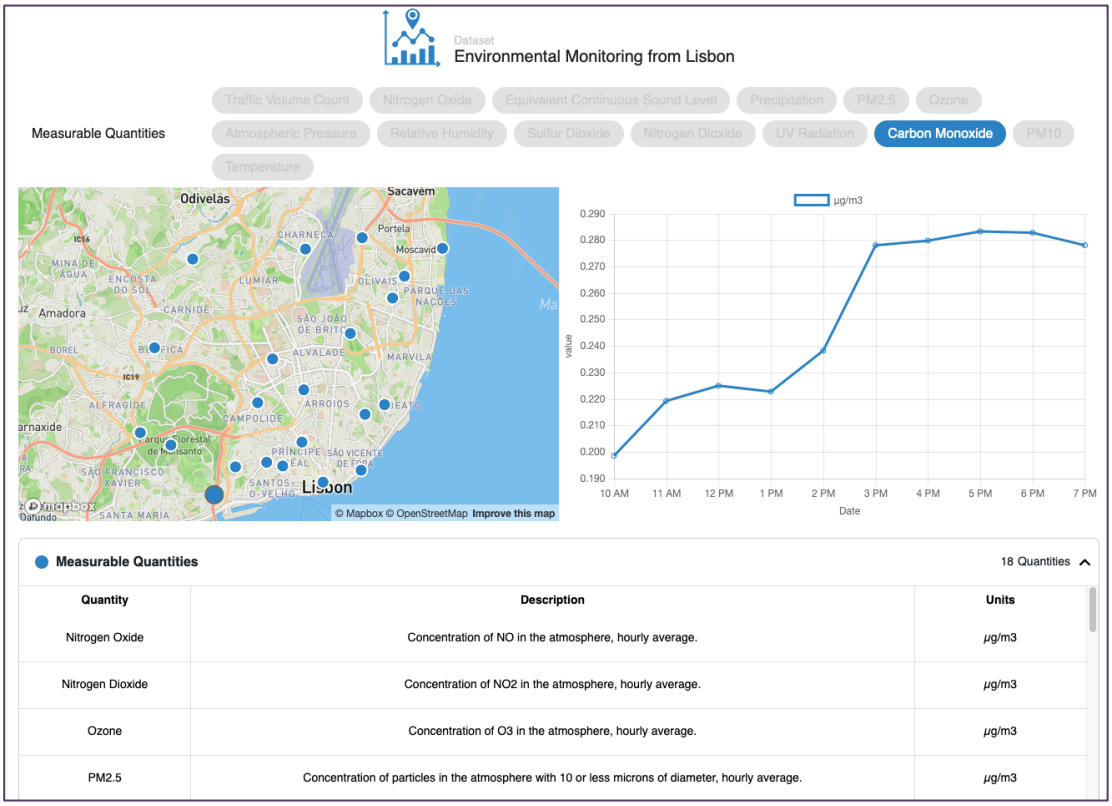

Figure 18: Dataset time-series visualiser.

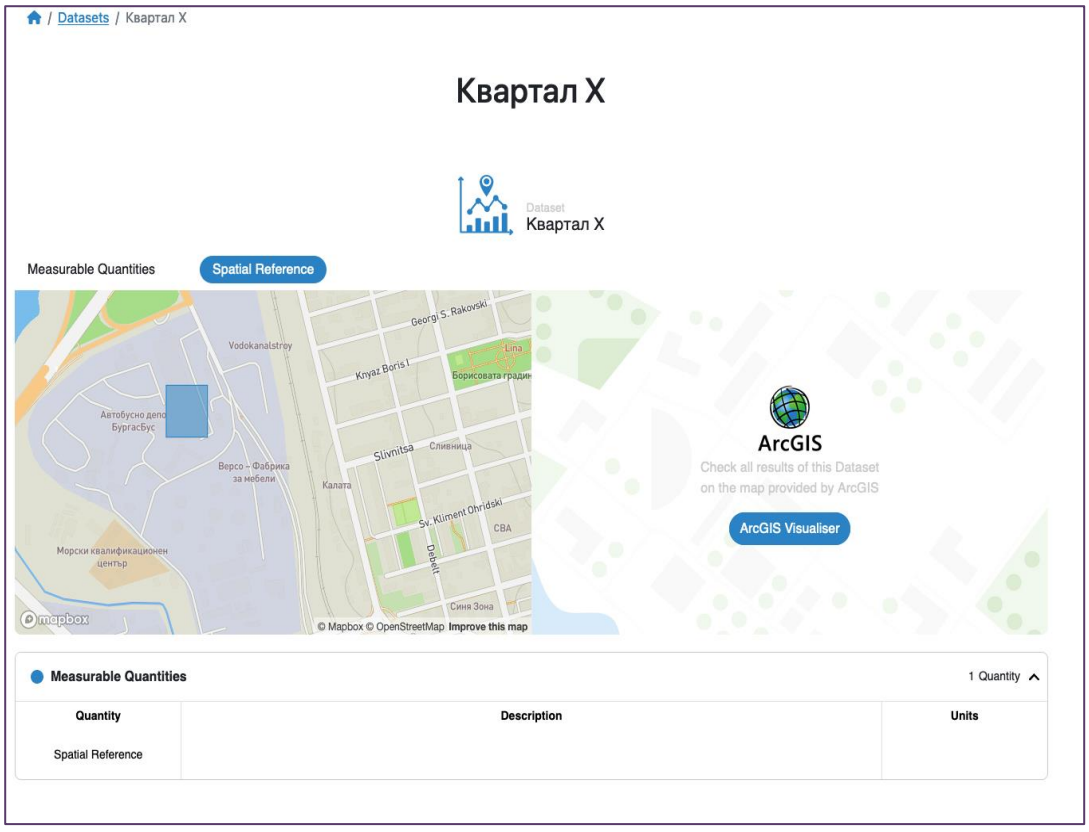

Figure 19: Dataset ArcGIS.

Georeferenced data can be visualised with the help of their online visualiser (Figure 20). The area shown in the image is also displayed on the map on the left.

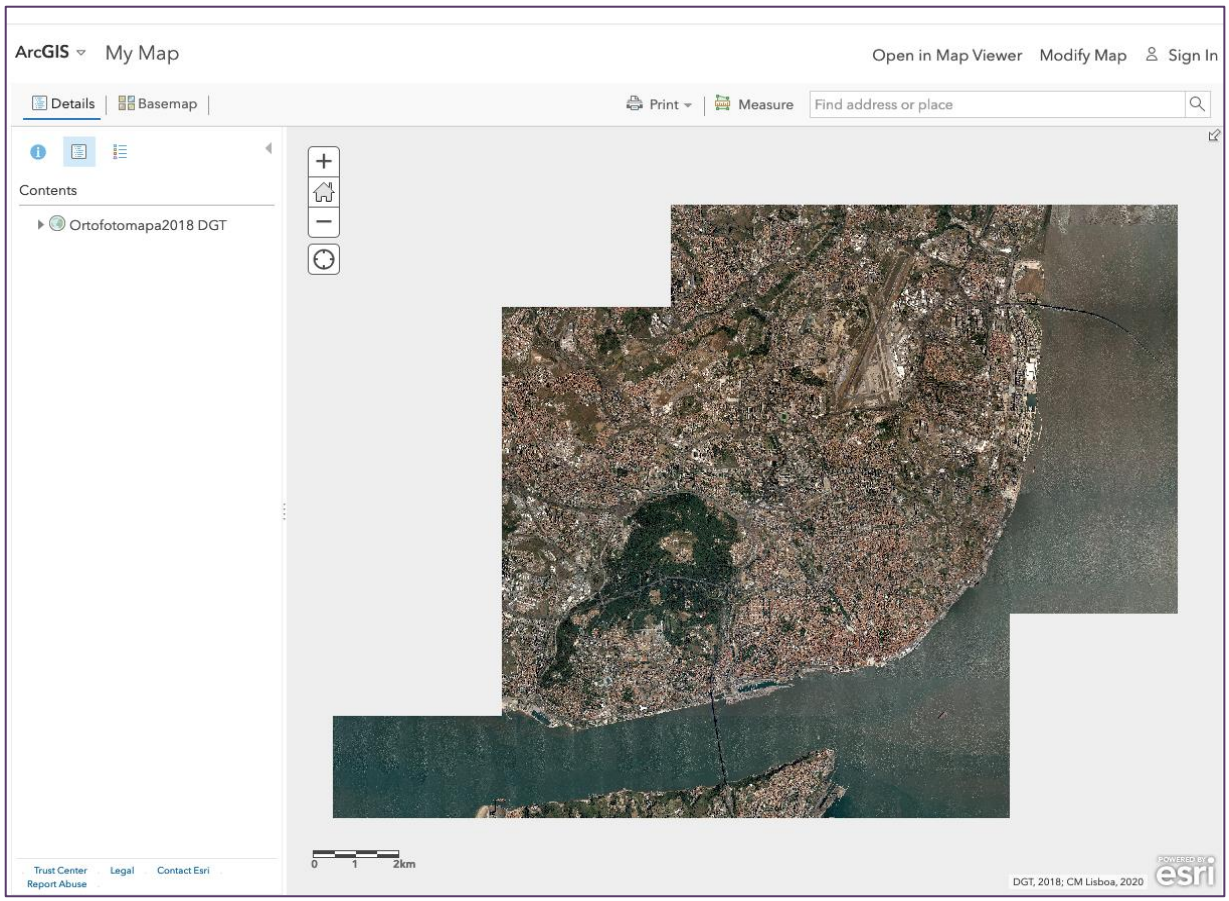

Figure 20: Dataset ArcGIS visualiser.

4.1.3 DATASETS EXPLORER

DATASET MAP VIEW

The map view (Figure 21) provides an overview of all Dataset locations, with the datasets listed changing according to the user interaction with the map. The user may also select Measurable Quantities to filter the results.

DATASETS

List view of datasets containing information about all data collections related with the selected data concept. If nothing is selected all datasets currently visible on the map are displayed by default. The list is updated as the user navigates and changes the map view.

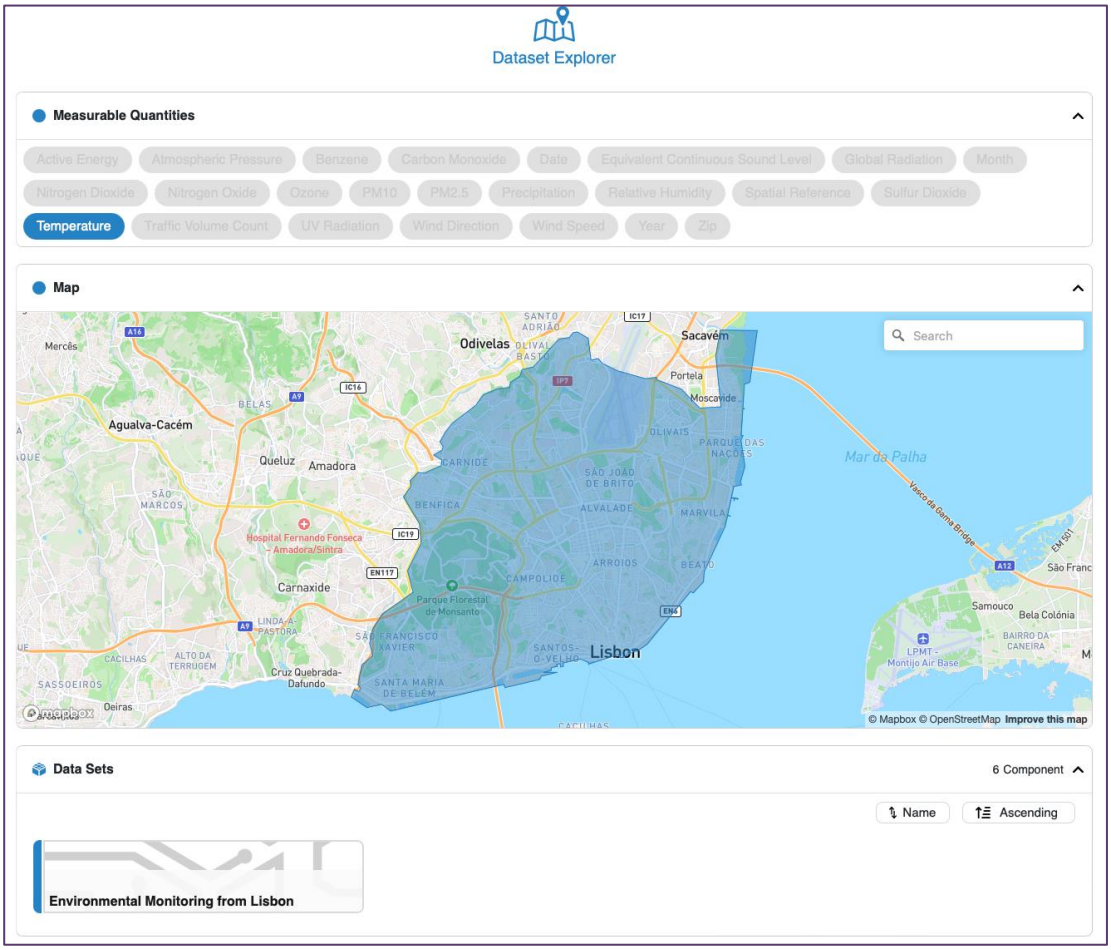

Figure 21: Interactive map view.

The view provides a clickable dataset listing using cards and geographical points, selecting a dataset will open the corresponding page with more details and the latest measurements, the concepts it captures as well as the Measurable Quantities of the values it contains.

4.2 CONTENT MANAGEMENT

The data described above will be available at the Datasets Catalogue of the DIP and can be accessed through a web interface or an external application using a REST API. Both access methods provide authentication for private data access.

The endpoints listed in Figure 22 are used to insert new datasets as well as to update any related information, such as the latest values to display in the time-series visualiser.

Figure 22: Swagger REST API.

The REST API enables the communication between the catalogue and external services. A token may be required when accessing restricted information. The REST API uses JSON for the communication between an external service and the catalogue, providing the following features:

- **Token authentication** – for private accesses, an authorised user may be authenticated using a unique token, which may be revoked to remove the user permission.
- **Data fetching** – allows an external application to fetch both private (when a token is provided) and public data.
- **Data search** – returns search results based on a query. Some wild card like "*" may be used.
- **Data update** – used to update data available on IoT Catalogue. This feature may be useful in scenarios where the data must be updated on a regular basis. A token is required to use this feature.
- **Adding new data** – used to insert new data on Catalogue. It is particularly useful when adding large sets of data. A token is required to use this feature.

The Dataset endpoint (*/api/data/dataSets/*) provides access to methods that manipulate information, as shown in Table 3. All the API responses are in JSON format.

Table 3: Dataset endpoint methods.

Operation	Input	Output
GET	*id*: ID of the dataset to retrieve. *access_token*: for fetching private data.	Dataset matching the provided ID.
PUT	*id*: ID of the dataset to edit. *access_token*: for fetching private data.	JSON object with fail or success message.
DELETE	*id*: ID of the dataset to remove. *access_token*: for fetching private data.	JSON object with fail or success message.
GET	*itemsPerPage*: number of items per page (default is 10). *page*: limits results to the selected page. *access_token*: for fetching private data.	Array containing the retrieved datasets.
POST	The request body must contain the dataset to be added.	JSON object with fail or success message.

5 CONCLUSIONS AND IMPLICATIONS

This deliverable aimed at summarising all the work carried out under the scope of tasks *T5.1 DIP technical solution and reference* architecture and *T5.2 Data governance* plan, up until month 17 of the CHAMELEON project.

The high-level architecture and the shopping cart approach were introduced and, even though some small changes may take place due to further development works, the layout of the core components of the system is not expected to suffer significant modifications.

Regarding the graphical appearance of the pages, the colour pallete in use should remain the same as it is in accordance with the project's. The main characteristics of the pages layout is fixed as well, notwithstanding some adaptations that may take place to accommodate any emerging feature.

On the other hand, the implementation stage will likely demand significant amount of effort in the near future (e.g., developing the Orders Management page, using the *getOrders* method from the Helper). Likewise, integrating the Datasets Catalogue and supporting the integration of the CHAMELEON Store and the Bundle Execution Mechanism, which implies interacting with partners from T5.3, T5.4 and WP4, will also be top priorities as part of the WP5 coordination.

CHAMELEON

A Holistic Approach to Sustainable, Digital EU Agriculture, Forestry, Livestock and Rural Development based on Reconfigurable Aerial Enablers and Edge Artificial Intelligence-on-Demand Systems

The Members of the CHAMELEON Consortium:

ACCELIGENCE

adrestia

AIDEAS

DIPUTACIÓN DE ÁVILA

DELAIR

DIE JOHANNITER

LAMMC LITHUANIAN RESEARCH CENTRE FOR AGRICULTURE AND FORESTRY

CIHEAM MAI CHANIA

SQUAREDEV

UCLM Universidad de Castilla-La Mancha

UNPARALLEL

VNiVERSiDAD D SALAMANCA

Contact:

Project Coordinator: **Pantelis Velanas** Acceligence Ltd.	pvelanas@accelligence.tech

Disclaimer