Deliverable D3.2: Technical Specifications

WP	3	Climate Service and Mobile Application	
Task	3.2	Technical Requirement and Architectural Design	
Dissemination level[1]	PU	**Due delivery date**	31-12-2017
Nature[2]	R	**Actual delivery date**	18-06-2018

Lead beneficiary	MET
Contributing beneficiaries	ISMB, NAP, IRTA, BSC

Version	1.0
Total number of pages	58

[1] Dissemination level: **PU** = Public, **PP** = Restricted to other programme participants (including the JU), **RE** = Restricted to a group specified by the consortium, **CO** = Confidential, only for members of the consortium

[2] Nature of the deliverable: **R** = Report, **P** = Prototype, **D** = Demonstrator, **O** = Other

Version history

Document Version	Date	Author	Comments[3]
0.1	15/10/2018	María Navarro (MET)	
0.2	15/11/2018	Claudio Rossi (ISMB)	Architecture
0.5	10/12/2018	Claudio Rossi (ISMB)	Initial data format
0.6	02/02/2018	Boris Basile (NAP), Omar Garcia Tejera (IRTA), Ignasi Porras (MET), Raül Marcos (BSC)	Details on forecast modules
0.7	03/04/2018	Claudio Rossi (ISMB)	Revision of data formats
0.8	03/04/2018	María Navarro (MET)	Quality check
0.9	28/05/2018	Claudio Rossi (ISMB)	Revision of data formats
1.0	18/06/2018	Claudio Rossi (ISMB), Alessandro Farasin (ISMB)	Finalization of overall system architecture, data formats and structure

Deliverable abstract

The main goal of this document is to finalize the overall technical activity of WP3, namely the definition of the system architecture, the specification of the main processes, data formats and data structures, and the list the key technical requirements of the platform.

This document contains the output of both task 3.2 “Technical requirement and architectural design” and of task 3.3 “Data integration and processing”. While T3.2 was smoothly executed, T3.3 required a major effort in order to finalize due to the cross definition coming from WP2. Therefore, the submission of this deliverable has been delayed so as to include the finalization of the data structures and of Application Program Interface, which were possible only after the final definitions in WP2.

This deliverable is structured as follows: after the document objectives are stated in Section 1, the general overview of the VISCA system is reported in Section 2. Then, the system architecture that implements such overview is explained in Section 3 together with more details on the key modules. Section 4 and Section 5 include the detailed specification of the data formats and of the key technical requirements, respectively.

[3] Creation, modification, final version for evaluation, revised version following evaluation, final.

Contents

List of Figures

List of Tables

1. Document objectives

The main goal of this document is to finalize the overall technical activity of WP3. Specifically, this deliverable aim to define:

- the system architecture together with the detail of its key components and processes
- the data formats and structures;
- the key technical requirements of the platform.

2. General System Overview

The initial VISCA overall architecture scheme (Fig. 6 of the project proposal) has been updated in order to reflect the outcomes of task T3.1 “End-user requirement definition” as well as the definitions done in WP2. The updated diagram, reported in Figure 1, shows a high-level process diagram and the associated system components, each of which is assigned to WP2 and WP3, accordingly.
The process starts when an end-user buys the VISCA solution and registers into the platform. Consequently, the end-user is allowed to load the system with the initial inputs that include the information regarding the vineyard, the irrigation system and the weather station. The vineyard data is composed by the initial data, which contains the specification of the grape variety, the soils, etc., the desired parameter in terms of quantity and quality, and the achieved, which related to the phenological phases and the final outcome after harvest. Next, a monitoring phase starts, during which weather, phenological, and irrigation models provide specific forecasts. According to the model requirements, selected data are fetched from the platform, while the final output is always sent to the platform so that the information can be shown to the end-user though the responsive web app, which is compatible with all screen resolutions and hence can be used from both desktop PC and mobile devices (smartphones and tables). The web app also allows the end-user to give feedbacks according to the observed phenomenon, e.g., phenological phase, or achieved action, e.g., canopy management or crop forcing. Such feedbacks are considered by the models in order to properly update the forecasts.

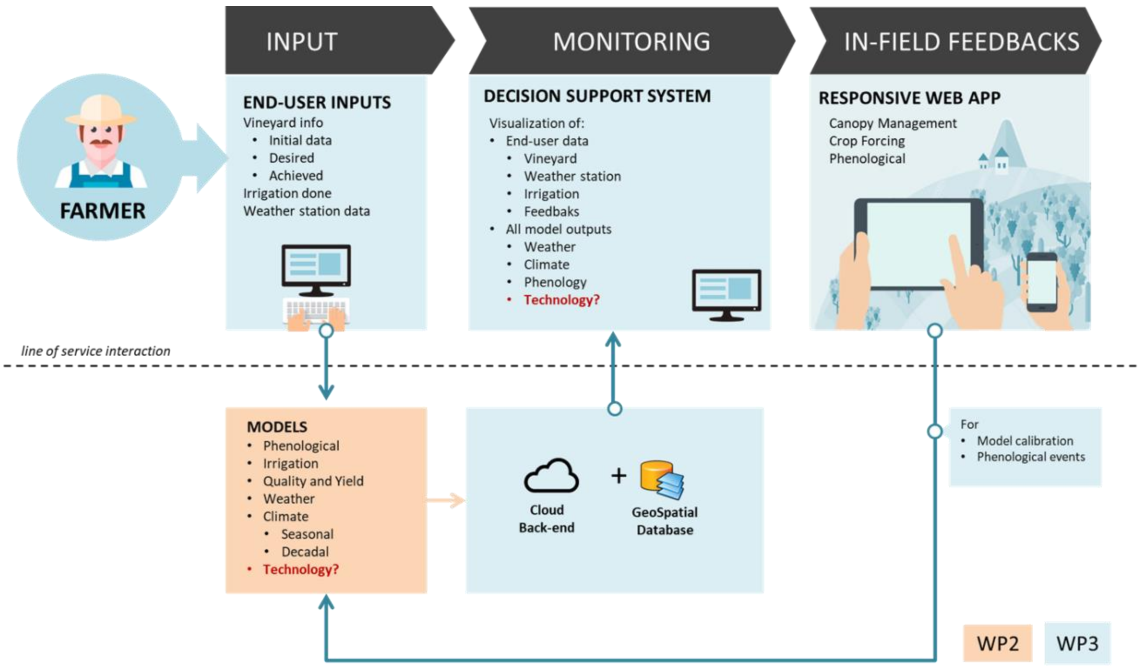


Figure 1. VISCA general system overview

3. Overall System Architecture

The general system overview has been translated into a technical architecture within Task 3.2 "Technical requirement and architectural design".

In order to provide a secure, reliable, scalable and GDPR[4] compliant solution, a Cloud based architecture has been chosen. This also allows to focus on the development of end-user requirements while exploiting advances and ready-to-use tools and services.

In the following, the benefits of the Cloud computing approach are briefly explained, together with the reasons behind the selection of the Cloud provide for VISCA.

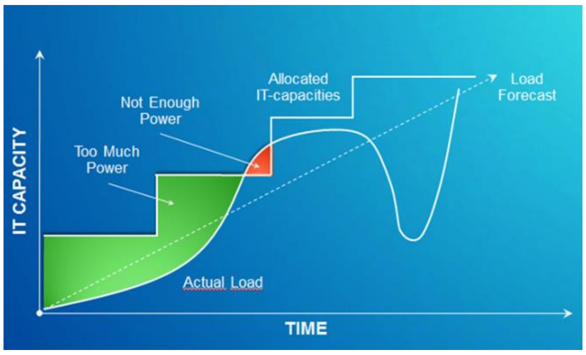
3.1. Cloud Computing

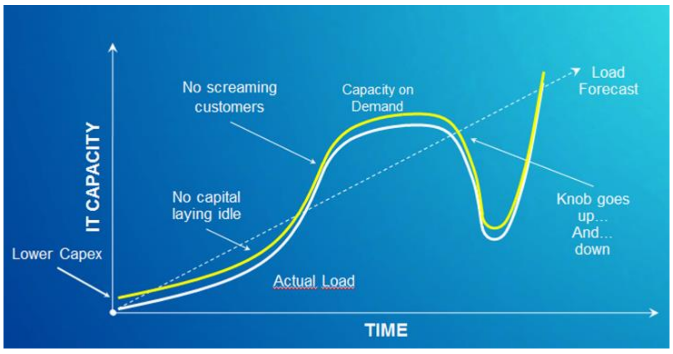
Cloud Computing[5] is a model which enables on-demand access to a shared resource pool through the network. These highly configurable resources can be easily accessed and released with a minimum management effort and a limited interaction with the service provider. Cloud computing solutions are characterised by high availability, scalability and performances. They represent a very profitable opportunity for the industrial sector and they allow to reduce effort, time and costs of developments, scalability, and

[4] https://www.eugdpr.org/

[5] https://csrc.nist.gov/publications/detail/sp/800-145/final

management. Furthermore, the use of services and infrastructures provided by Cloud Computing providers let SME to focus investments on the business and rapidly adapt to market's evolutions and new business opportunities. From a financial perspective, the adoption of cloud-based solutions allows to change the business strategy from the traditional CAPital EXpenditure (CAPEX) to OPerational EXpenditure (OPEX). The benefit of this approach is sketched in Figure 2, and they can be summarized by the slogan "pay as you need".





(a) (b)

Figure 2: Comparison between CAPEX model (a) and OPEX model (b).

The adoption of the traditional CAPEX model implies to buy and dimension computing resources in design phase, which is risky both technically and financially.

With cloud computing solutions (OPEX), it is possible to have the possibility to dynamically adapt the IT resources (i.e., computing power, storage, RAM) in function of the instantaneous requirements. This strategy allows to optimize costs versus needs in terms of computing capacity.

Cloud computing solutions are usually classified in three service models:

- **Infrastructure as a Service (IaaS):** the service provider delivers the complete framework of servers, routers, storage, hardware and virtualisation software; the user is responsible for the operating system, middleware, runtime and applications.
- **Platform as a Service (PaaS):** the service provider delivers the whole hardware and software chain which includes networking and runtime functionalities; the user is responsible for data and application management.
- **Software as a Service (SaaS):** the service provider delivers the whole service, including applications and data; the user just uses the service's functionalities.

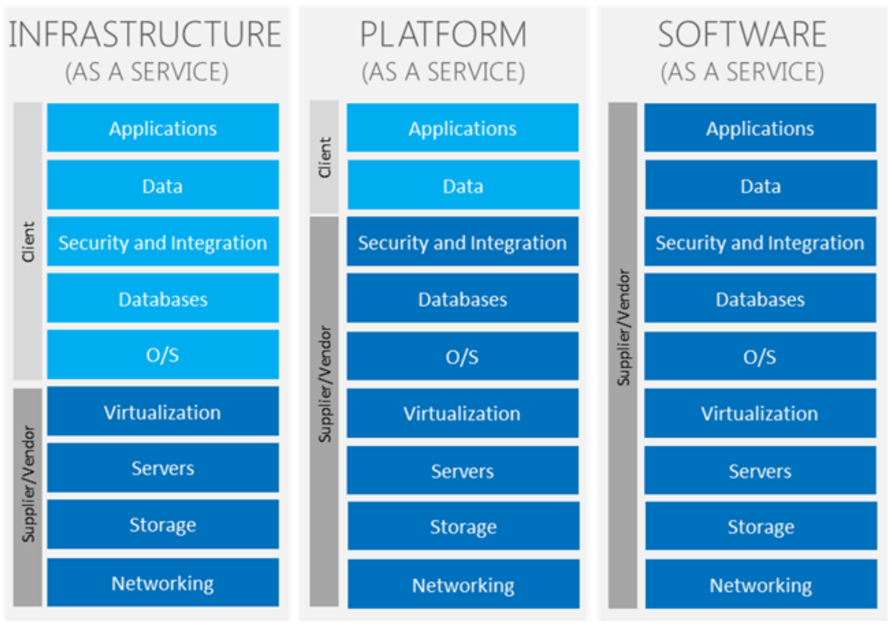


Figure 3: Cloud computing service models

Each of these models have pros and cons[6] and the selection of the service model can differ between modules of the same application. The service model selection is made every time according to the application needs and balancing costs versus benefits. VISCA relies on the IaaS approach for the key software components and minimizes the adoption of PaaS services to easily transfer the solution among Cloud providers in case of future need.

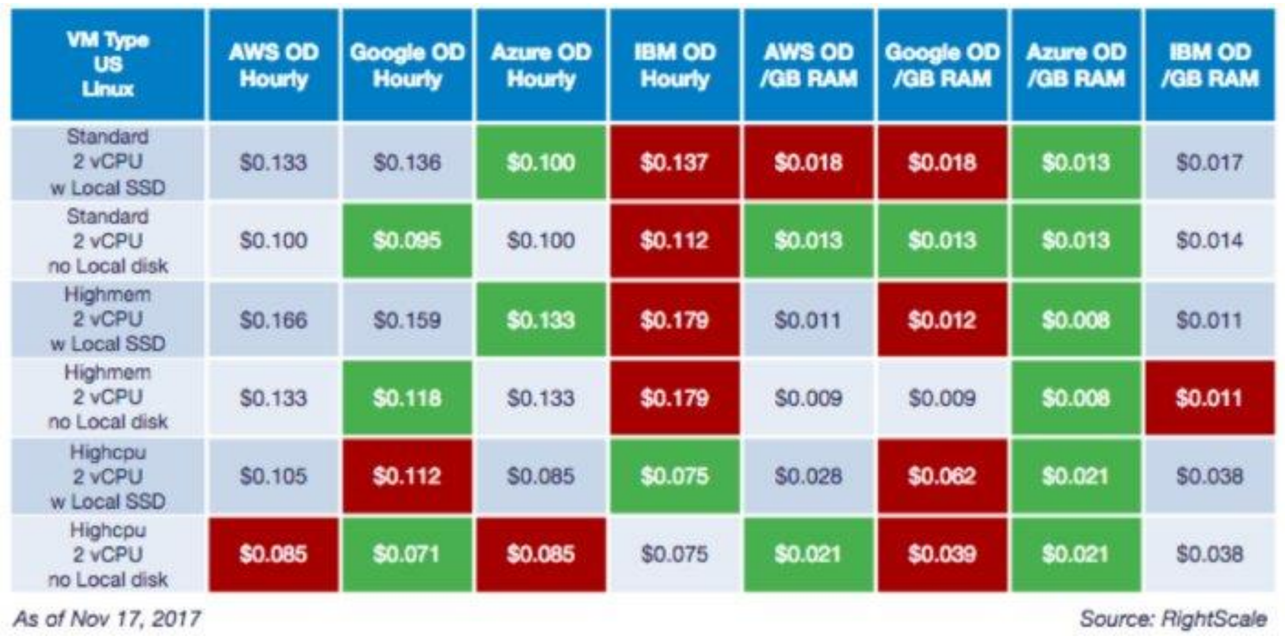
The cost of Cloud Computing solutions is very important and as major cloud providers (AWS, Azure, Google, and IBM) continue to drop the prices of cloud instances, they have added discount options, added instances, and dropped the billing increments, in some cases, from the second billing.

We compare the costs of typically used On-Demand (OD) Virtual Machines (VM) among the key Cloud providers. The result, shown in Figure 4, show that:

- Azure is lowest price for eight scenarios; highest price for one scenario.
- Azure is lowest price for all of the scenarios that include a local SSD and all comparisons based on per GB RAM.
- Azure matches or is lower than AWS for all scenarios.
- Google Cloud is lowest price for four scenarios; highest price for five scenarios.
- Google Cloud tends to be the lowest price when no SSD is needed.
- Google Cloud is higher priced on the “per GB RAM” cost for high CPU due to the fact that it includes less than half the memory of AWS and Azure.
- AWS is lowest price for two scenarios; highest price for two scenarios.
- AWS is most often a middle-priced option.
- IBM is lowest price for one scenario; highest price for five scenarios.

[6] The discussion of pros and cons of Cloud Computing service models is beyond the scope of this deliverable.

Given that the services offered by the different provides are comparable, we select Azure as the Cloud platform for the VISCA project. Moreover, Azure offers an intuitive online dashboard that allows to easily create and manage a wide set of services.



VM Type US Linux	AWS OD Hourly	Google OD Hourly	Azure OD Hourly	IBM OD Hourly	AWS OD /GB RAM	Google OD /GB RAM	Azure OD /GB RAM	IBM OD /GB RAM
Standard 2 vCPU w Local SSD	$0.133	$0.136	$0.100	$0.137	$0.018	$0.018	$0.013	$0.017
Standard 2 vCPU no Local disk	$0.100	$0.095	$0.100	$0.112	$0.013	$0.013	$0.013	$0.014
Highmem 2 vCPU w Local SSD	$0.166	$0.159	$0.133	$0.179	$0.011	$0.012	$0.008	$0.011
Highmem 2 vCPU no Local disk	$0.133	$0.118	$0.133	$0.179	$0.009	$0.009	$0.008	$0.011
Highcpu 2 vCPU w Local SSD	$0.105	$0.112	$0.085	$0.075	$0.028	$0.062	$0.021	$0.038
Highcpu 2 vCPU no Local disk	$0.085	$0.071	$0.085	$0.075	$0.021	$0.039	$0.021	$0.038

As of Nov 17, 2017 *Source: RightScale*

Figure 4: prices of On-Demand (OD) and Virtual Machines (VM) among the key cloud providers. Red indicates the highest price of the cloud providers within a scenario and green represents the lowest price. If there are ties, then both cloud providers are highlighted green or red.

3.2. Design of the software architecture

Architecture styles place constraints on the software design, each of which provides both benefits challenges. In order to select the best architecture style for our needs, the key feature of the main styles have been reviewed (Table 1).

Table 1: List of the key architectural styles and classification according to key feature and domain type

Architecture style	**Key feature**	**Domain type**
N-tier	Also called multi-tier architecture, the software is engineered to have the processing, data management, and presentation functions physically and logically separated.	Traditional business domain. Frequency of updates is low.
Web-Queue-Worker	Front and backend jobs, decoupled by async messaging.	Relatively simple domain with some resource intensive tasks.
Microservices	Vertically (functionally) decomposed services that call each other through APIs.	Complicated domain. Frequent updates.

CQRS	Read/write segregation. Schema and scale are optimized separately.	Collaborative domain where lots of users access the same data.
Event-Driven Architecture	Producer/consumer. Independent view per sub-system.	IoT and real-time systems.
Big Data	Divide a huge dataset into small chunks. Parallel processing on local datasets.	Batch and real-time data analysis. Predictive analysis using ML.
Big Compute	Data allocation to thousands of cores.	Compute intensive domains such as simulation.

In order to respond to scalability and modularity requirements, both of which allows to ease the exploitation of the final software products, multiple architectural styles have been chosen and combined together. Namely:

- **The microservice** design pattern has been chosen in order to structure the overall architecture because it allows different modules to cooperate together through both synchronous and also asynchronous Application Program Interfaces (APIs)
- **The Event-Driven architecture** has been selected in order to enable a single asynchronous communication bus that implements at the same time the publish and subscribe (pub/sub) messaging, which is a form of asynchronous service-to-service communication used in serverless and microservices architectures. In a pub/sub model, any message published to a topic is immediately received by all of the subscribers to the topic. This approach is used to decouple applications in order to increase performance, reliability and scalability. Specifically, the Microsoft Azure Service Bus[7] is used as the pub/sub broker.
- **Big Data** components are needed in order to store and process the files that will be operationally generated by weather forecasts. In this case, a distributed system file is used. Specifically, the Microsoft Azure Blob Storage[8] is used as the Big Data storage and Apache Spark[9] will be used in order to process data.

The resulting VISCA architecture (Figure 5) shows all the key software modules, which are divided into two main groups:

1. **Climate Service**: all these modules have tight links between each other and they cooperatively deliver the Climate Service to end-users through a set of real-time HTTP APIs for automatic data and signal exchange and through the Responsive Web App for human interactions. All these modules are deployed in the same Cloud data centre so enable fast data transfer and low latency.

[7] https://azure.microsoft.com/en-us/services/service-bus/
[8] https://azure.microsoft.com/en-us/services/storage/blobs/
[9] https://spark.apache.org/

2. **External Modules**: these relates to all software and systems that runs externally to the Climate Service while delivering data. They include the existing end-user's system, e.g., the weather station and irrigation systems, and also the weather and phenological models developed within WP2.

Next, each software module is briefly explained:

- **GeoServer**: it allows to deliver map layers using Open Geospatial Consortium (OGC) standards such as Web Map Service (WMS), which is ideal for both web-based interfaces and mobile connectivity. This component is required in order to visualize all weather forecasts in the Responsive Web Application, and it requires an SQL database with GIS features that stores the layer data, namely the PostgreSQL (with PostGIS extension).
- **Geo Importer**: this component allows to import map layers encoded in commonly used formats such as shapefiles or GeoJSONs into an PostgreSQL database with GIS feature that stores the layer data, namely the PostgreSQL (with PostGIS extension) for GeoServer. This component also includes the VISCA Data Interface (VDI) that allows to receive/provide data from/to external. Also, it is subscribed to the PUB-SUB topic used by the VDI (*newexternaldata*), which is fed with a message every time a new file is uploaded into the platform, and it delivers the result of the importing procedure to external modules through another PUB/SUB broker topic (importer). If the inserted data belongs to the data types to be mapped, the importer inserts it in the GeoServer PostgreSQL and then it creates a layer in the GeoServer. The importer uses another PostgreSQL in order to keep the list of received files together with their metadata and the references to the layer created in the Geoserver, if any. This component also offers a service that lists all the available layers given a spatio-temporal window, implements a purging strategy for map layers, which is required to keep under control the number of layer serviced by the GeoServer, and a re-importing service, through which is possible to import past layers. The re-importing service is triggered by the user through the Responsive Web Application, which propagate this request to the importer using the PUB/SUB broker (*reimport* topic);
- **Visca Data Interface (VDI)**: this component implements a file repository and discovery service through which the External Modules can insert and retrieve files coupled with INSPIRE compliant metadata. When a new data is received by the VDI, a message on the topic *newexternaldata* is sent. The VDI handles pre-defined data types, which are mapped to specific ID (viscatask number).
- **The PUB-SUB broker**: it implements the publish and subscribe messaging system and it offers APIs to send/receive messages from/to topics for all other modules, including external ones. Messages are formatted in JSON[10].
- **Data Layer**: it contains all the databases required by the Climate Service modules. There is one distributed file system for Big Data (Azure Blob Storage), three PostgreSQL with PostGIS extension to handle geographical data (one for the importer plus VDI, one for the back-end, and one for the GeoServer). The *Azure Storage* is adopted by the Visca Data Interface for storing raw files with measures, spatial references and metadata. The properties contained in these files are processed and stored into the *PostgreSQL For Geo Importer*, to allow to access and filter fast the whole data, once uploaded. The properties extracted by the original files are linked to the original sources, thus they can be always reached and downloaded. Moreover, the *PostgreSQL For Geo Importer* stores the

[10] https://www.json.org/

references to the layers represented by the GeoServer and stored into the *PostgreSQL for GeoServer* for a better resource management. The *PostgreSQL for back-end* stores the account credentials and all the information strictly related to the VISCA platform use-cases: in fact, it stores data related to vineyards, predictions and companies involved. All the databases explained so far are relational because information have pre-defined and constant structures. Instead, for storing sensors acquisitions a NoSQL database is considered. As a non-relational database, it guarantees both great flexibility about data formats and fast storage ability.

- **Responsive Web App**: it is the module which the VISCA platform users directly interact with. Through the Responsive Web App, each user has a clear view about its vineyards and related parcels information. The Responsive Web app facilitate the platform usage, giving for each user its own credentials and the chance to insert, retrieve, visualize data as forms, plots and layered maps. It is designed to adapt the contents to every device: it can be accessed by mobile, tablet and/or desktop pc. This component interacts directly with the Backend (for the content management and retrieval) and the GeoServer, from which it fetches feature information about displayed layers.
- **Back-end:** this module aggregates the information from the other components and provides the results through RESTful APIs. It manages the user credentials: each user can register to the platform and edit its information. Moreover, it allows to access, update and delete information about vineyards and related parcels and irrigation blocks. It integrates layers and sensors information with the related data about vineyards and related to the respective users.
- **Sensor Service:** this module interacts with the VISCA Backend and external sensors through RESTful APIs. It collects acquisitions from sensors and stores them into the *NoSQL database for Sensor Service*. Furthermore, it provides the stored data to the Backend.

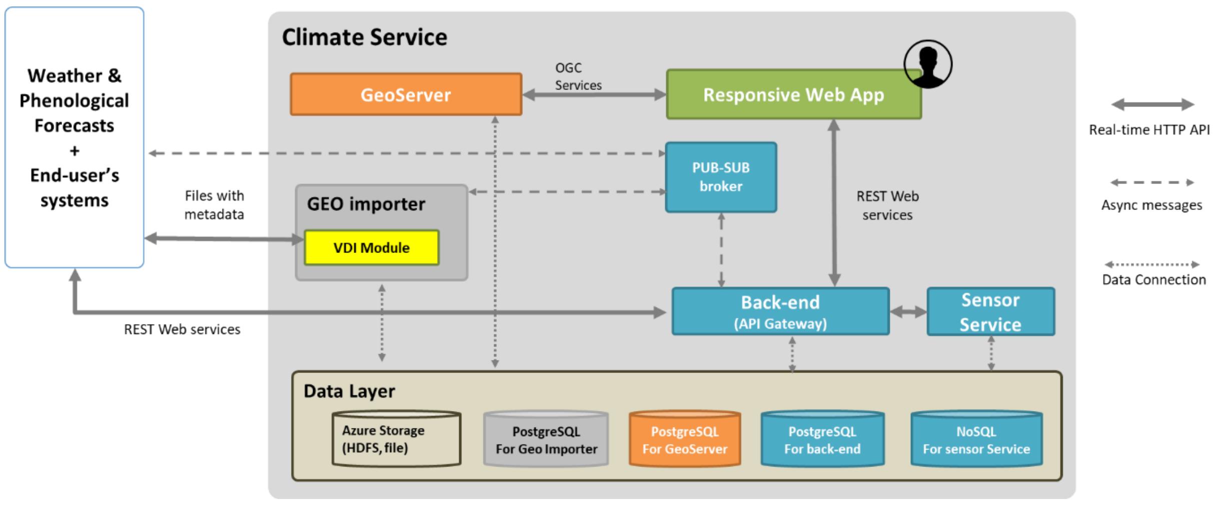


Figure 5: VISCA system architecture

3.2.1. Geo Importer and Visca Data Interface

In this section, two fundamental modules for data management and geo-spatial data importing are presented. The sub-section 2.2.1.1. focuses on the Geo Importer, which deals with spatial information, by fetching, processing spatial file properties and importing them into the GeoServer. In the other sub-section, the VISCA Data Interface is presented. That module is strictly related to the Geo Importer, it allows to manage a wider set of metadata and files, allowing third parties modules to interact with it, by uploading, querying and getting informed about VDI files interchange.

3.2.1.1. Geo Importer service

The Geo Importer is the service responsible for preparing and importing geospatial data into the GeoServer. It includes the VDI for a generic file management extending its functionalities for spatial information. Among the several files managed by the VDI, the Geo Importer deals with geometries and properties related to geospatial data. It is able to manage the following file extensions: .shp, .tif, .tiff, .tff .nc, .json, .geojson.

It offers three main functionalities:

- It imports new geospatial data, when new geospatial files are uploaded into the VDI
- It re-imports geospatial data, when the Responsive Web App requests specifically to reload specific layers
- It lists all the available layers, given a temporal range and a spatial window

The first functionality is shown in the diagram in Figure 6Figure 6: Geo Importer Communication Diagram for a new layer to be imported. Once VDI Module successfully stores metadata and related files, a message (1) is sent to the PUB-SUB broker in the topic *newexternaldata*. The Geo Importer, which is subscribed to that topic, receives the message (2), containing suitable information to reach the data structure previously created. Then, the files and related information about metadata are fetched (3) from the Azure Storage and the PostgreSQL to be further processed for the importing phase. During step (4), the data is processed and prepared to be imported into the GeoServer. Properties and geometries are extracted from the files, their validity (in terms of syntax and quantity) is verified and then adequately stored into the *PostgreSQL For GeoServer* tables. Subsequently, a reference to the processed information is stored in the *PostgreSQL For Geo Importer,* in order to keep tracking of the imported features. In the end, the GeoServer is triggered (5) to create the layer, referring to the information previously stored in the *PostgreSQL For Geoserver* database. In the end, a message describing the process execution outcome is sent to the *PUB-SUB broker* to the *importer* topic. That message includes information about the involved *viscatask*, the *metadata identifier* and a brief *description* about the errors occured.

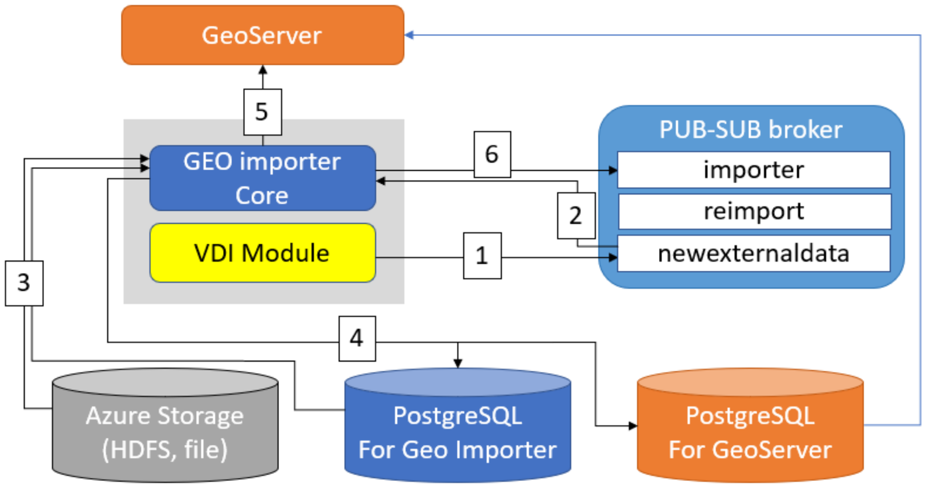


Figure 6: Geo Importer Communication Diagram for a new layer to be imported

The second functionality regards the need to reimport layers from files (Figure 7). It is triggered by the Responsive Web App which, through the Back-end APIs, sends a message (1) containing the metadata identifier to reimport to the *PUB-SUB broker reimport* topic. That topic is exclusively read by the Geo-Importer, which receives the message (2) and follows the exact steps (3, 4, 5, 6) explained for the first functionality. Here again, the last step (6) ensures that everything was executed correctly, reporting details in case of error.

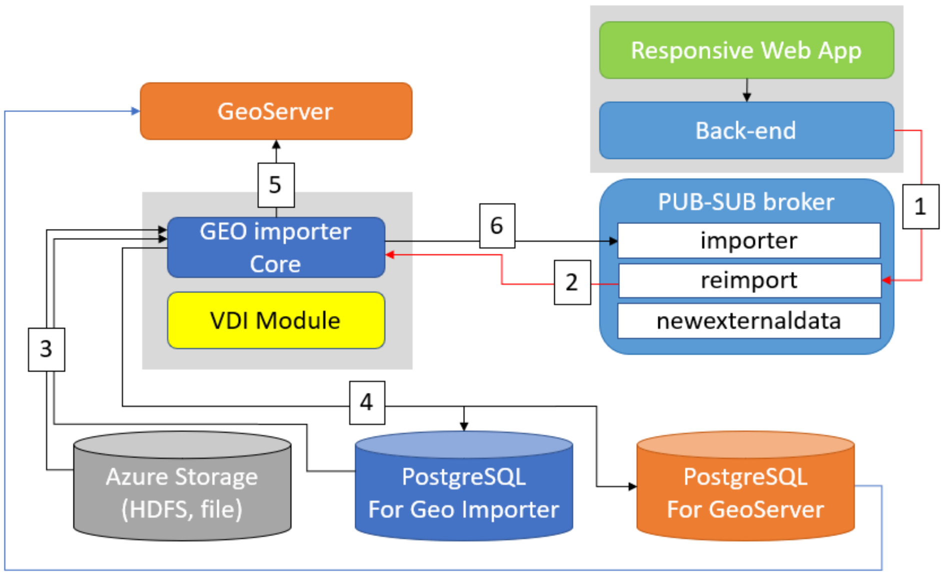


Figure 7: Geo Importer Communication Diagram for a layer to be re-imported

The third and latest functionality provides the chance to retrieve all the imported layers related to a certain area in a specific temporal range. To accomplish that, the *Geo Importer* refers to the *PostgreSQL For Geo Importer* database collecting information about *viscatask* number, the metadata and related file/s that match the specified criteria. On purpose, to allow the maximum flexibility and usability of this functionality, a RESTful API is foreseen.

3.2.1.2. Visca Data Interface

The VISCA Data Interface (VDI) has been defined in terms of two separate components: a *software module*, and an *associated data format*. The former is composed by a series of multiple adapters, each of which is used to extract data and metadata from the data formats at hand, accordingly. On the other hand, the defined data format specifies the format that has to be used in the communication protocol between external VISCA modules and services for data exchange. In this scenario, the VDI seemingly integrates with the VISCA Data Layer (see Figure 8), acting either as a broker for communication to and from the data layer with external VISCA services, and as a mediator for both the Offline and Online Storage. This role is further emphasized in the communication diagrams reported in Figure 9.

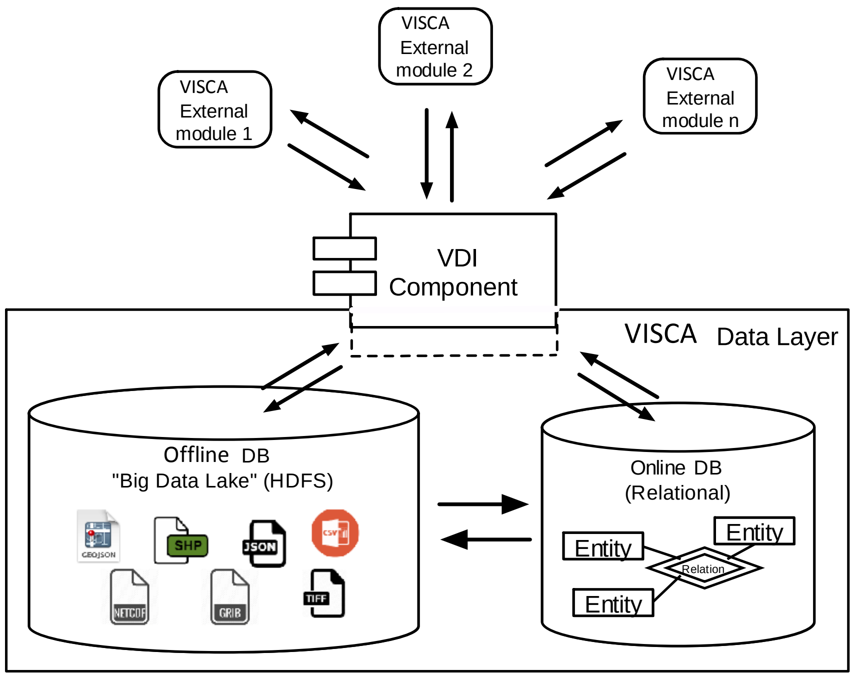


Figure 8: Architectural view of the VDI component as integrated with the VISCA Data Layer

In more detail, Figure 9 describes the process of an external VISCA service which is going to send data (in binary format) to the VISCA Data Layer. In this scenario, the external service issues an HTTP POST request to the VISCA Data Web Service - i.e. a RESTful endpoint specifically set up to receive data from external services. In this example, the request contains a series of binary files sent over the network with a multipart/encoding request, along with corresponding metadata. These metadata are sent by the external service in the VDI format and have the purpose to complement and describe corresponding data. Once data and metadata are provided to the VDI, proper adapters are triggered. These adapters embed the specific rules and algorithms required to manipulate data and metadata from specific data format. The VDI module instructs the different adapters so that rules and conditions to get the data are specified. Then, each adapter selects and fetches the data to be stored into the data storage. On the other hand, in case an external service asks the VISCA Data Layer for specific data, the VDI is responsible to fetch the data from the (online) storage, and transforms obtained results into the VDI defined format to enable the data exchange process. The format defined by the VDI is based on JSON. This JSON format is flexible enough to support dynamic data schemas, which is necessary to support heterogeneous data in multiple formats.

Moreover, the JSON extensions for geospatial and linked data, namely GeoJSON and JSON-LD would support the exchange of data from external sources and the semantic data layer. Finally, a REST style access to the data (via simple HTTP(s) requests) can be enabled, even if diverse data formats are involved in the processing.

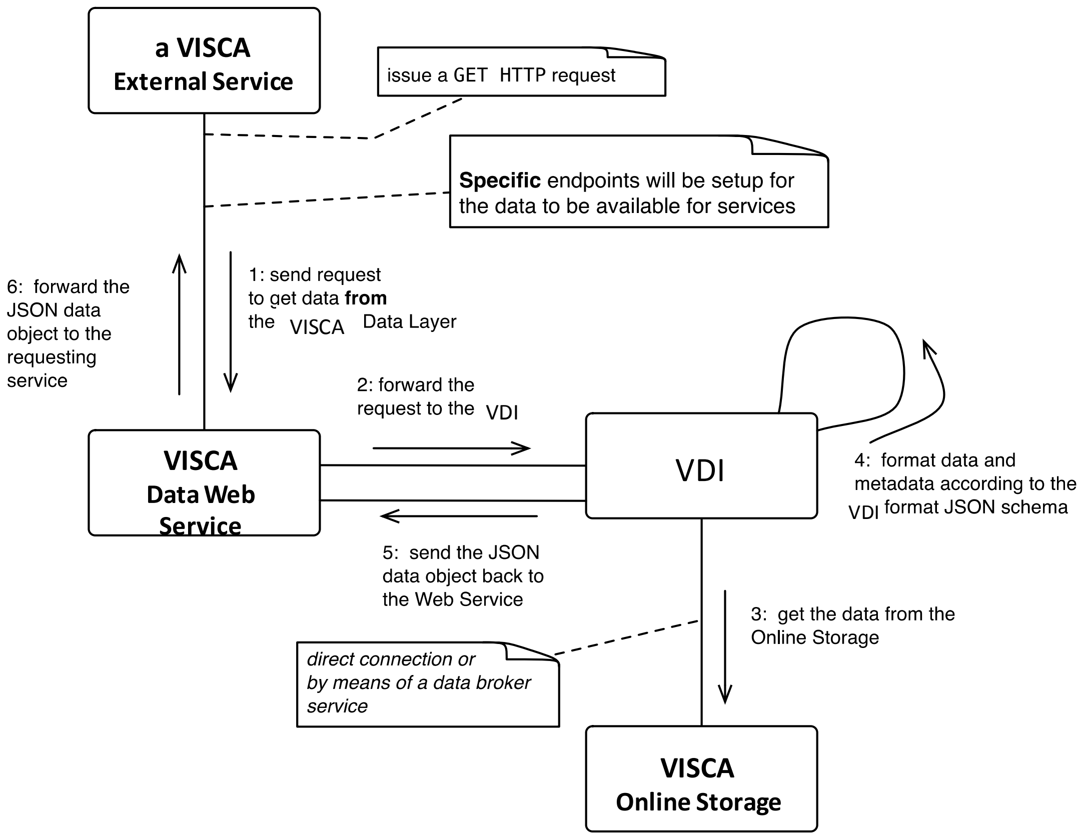

a VISCA External Service
issue a GET HTTP request
Specific endpoints will be setup for the data to be available for services
6: forward the JSON data object to the requesting service
1: send request to get data from the VISCA Data Layer
2: forward the request to the VDI
VISCA Data Web Service
VDI
4: format data and metadata according to the VDI format JSON schema
5: send the JSON data object back to the Web Service
3: get the data from the Online Storage
direct connection or by means of a data broker service
VISCA Online Storage

Figure 9: Communication Diagram for the process to get data from the VISCA Data Layer

3.2.2. Mapping service

Map server component

All VISCA maps are stored as geospatial features in the database selected above. In order to put the information together to user-tailored maps it is necessary to use a piece of software that serves the maps from the VISCA servers, at first to the User-Server Platform and then to the mobile applications of the users. For this functionality a so-called map server is used. This software does basically the same as a web server for a website, but it is specialised for maps.

There are currently only two mainstream options in place for achieving such a task: GeoServer and MapServer (formerly UMN MapServer). A comparison of these is made in the following table:

Table 2: Description of MapServer options

	GeoServer 2.13	**MapServer 7.0.7**
WMS	implementing the OGC standard	implementing the OGC standard
WFS	supports WFS-T	no WFS-T
Technology	J2EE	CGI
Project start	2003	1996
Administration	web tool	support through QGIS, but not very user-friendly
Cartography	standard SLDs	styles are part of mapfile
Services	one WMS/WFS/WCS service for all users	one mapfile for each service
Extensibility	good for Java developers	mapscript, good for PHP developers
License	GNU general public license, version 2.0	MIT License: Free software license originating at the Massachusetts Institute of Technology (MIT).
Evaluation	Due to the web tool for administration and support of WFS-T GeoServer is superior to MapServer	Good product, but administration is not convincing and lack of WFS-T support could pose a problem when extending VISCA.

Based on this table the technology of choice for map serving as part of the GEO Gateway is the open-source product GeoServer in current stable version 2.13 (http://www.geoserver.org)

GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards. Being a community-driven project, GeoServer is developed, tested, and supported by a diverse group of individuals and organizations from around the world.

GeoServer is the reference implementation of the Open Geospatial Consortium (OGC) Web Feature Service (WFS) and Web Coverage Service (WCS) standards, as well as a high performance certified compliant Web Map Service (WMS). GeoServer forms a core component of the Geospatial Web.

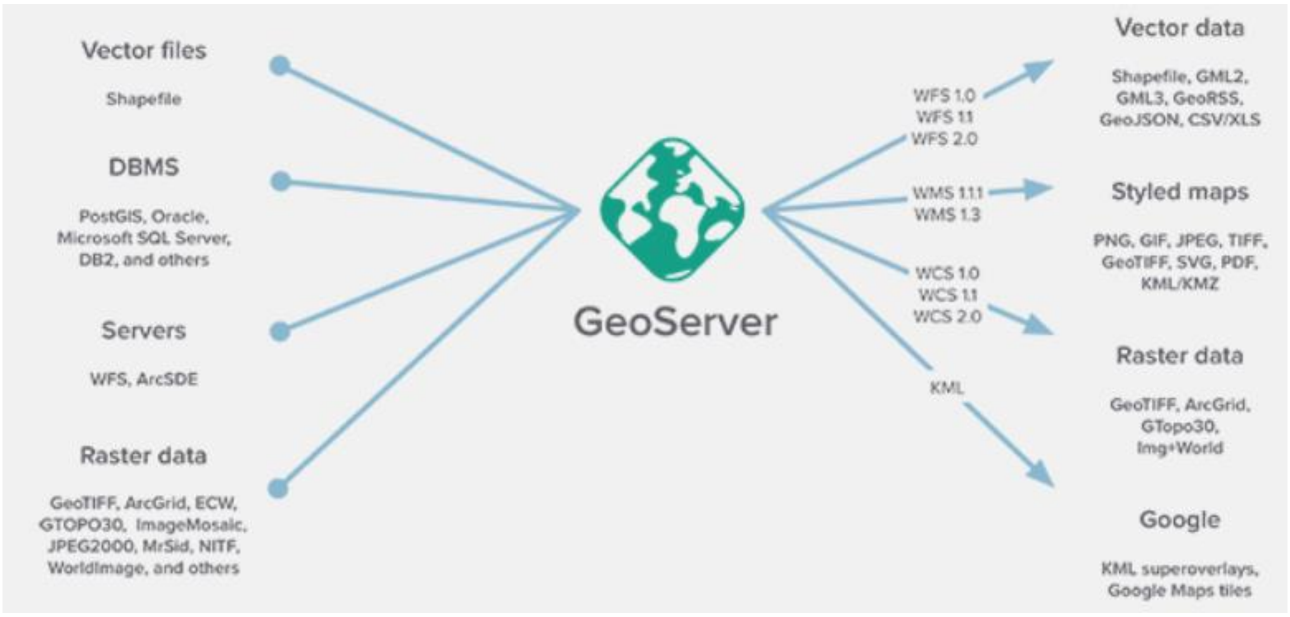

Vector files
Shapefile
DBMS
PostGIS, Oracle, Microsoft SQL Server, DB2, and others
Servers
WFS, ArcSDE
Raster data
GeoTIFF, ArcGrid, ECW, GTOPO30, ImageMosaic, JPEG2000, MrSid, NITF, WorldImage, and others
GeoServer
WFS 1.0
WFS 1.1
WFS 2.0
WMS 1.1.1
WMS 1.3
WCS 1.0
WCS 1.1
WCS 2.0
KML
Vector data
Shapefile, GML2, GML3, GeoRSS, GeoJSON, CSV/XLS
Styled maps
PNG, GIF, JPEG, TIFF, GeoTIFF, SVG, PDF, KML/KMZ
Raster data
GeoTIFF, ArcGrid, GTopo30, Img+World
Google
KML superoverlays, Google Maps tiles

Figure 10: The GeoServer components and architecture (source: OGC)

Caching component

The solution for caching map data is dependent on the map server. As we have decided to use GeoServer the extension GeoWebCache will be used (http://www.geowebcache.org). This component enables the creation of WMTS (Web Map Tiled Services), more specifically, it enables image tiling to speed up the performance and to facilitate data storage capacity as the visualization on the mobile map application can take place offline through pre-cached image tiles. It is possible to specify different zoom levels (levels of detail, scale parameters, e.g. 1:100.000) and the size of the tiles (e.g. 256 pixels).

We have made several choices regarding software to find an optimal solution. In the following paragraphs each choice is described together with alternatives.

Database component

For storing geospatial data there are several viable solutions, which are summarised in

Table 3: Description of potential database solutions. Most databases can store geometry and geographic information through binary coding. Geodata processing capabilities and spatial functions however are often limited. Proprietary database systems with spatial functionality have a very high performance but do also have rather high prices. Therefore, the focus lies on a high performance, open source spatial database that enables geodata handling and processing per se.

The following criteria were used to evaluate the different types of spatial databases:

- **Spatial functions**
  Most database management systems are capable of encoding and storing geometric and geographic data that can be accessed by external applications. Most of the spatial database extensions are

capable of simple geometric functions such as spatial relations (intersect, union, difference, contain), analysis (buffer, envelope, area) or conversion of encoded data (e.g. WKB, well known binary) into other geodata formats (shapefile, GML). But only a few spatial databases enable the server-side processing of geodata with complex spatial analytical functions.

- **Spatial formats**
  Does the database support the most common geodata formats and the ones that are conform with OGC standards (GML, GeoJSON)?
- **GeoServer support**
  Does the database support web services such as WMS or WFS? Is it possible to automate the data exchange between a GeoServer and the database?
- **Costs**
  Is the system Open Source (free of charge) or proprietary?
- **SQL Compliance**
  Is the internal system compliant with SQL-Standards?
- **Performance**
  How does the database system handle data and how does the performance speed depend on the amount of the stored data (e.g. big data handling)?
- **GUI**
  Is the Graphical User Interface user-friendly, legible and fast?
- **Automation**
  Is an automation of the database possible and what programming/scripting language being most suitable? Is the automation feasible or complex?
- **Interoperability**
  How is the interoperability between other database management systems (DBMS) and geographic information systems (GIS)?
- **Operating System**
  Which OS is supported by the database system?
- **Type**
  Is the DBMS relational (RDBMS), object-relational (ORDBMS) or object-oriented (OODBMS)

Table 3: Description of potential database solutions

Database	Spatial Extension	Spatial functions	Spatial formats	GeoServe rsupport	Costs	SQL complianc e	Performance	GUI	Automation	Interope-rability	OS	Type
Oracle	Oracle Spatial	Very good	Not all	NO	~ 100 K	FULL	Very fast	Good	Complex	Good	Most	ORDBMS
PostgreSQ L	PostGIS	Very good	ALL	YES	Open Source	FULL	Fast	Good	YES, Python, JAVA or C++	Very good	ALL	ORDBMS
MySQL	MySQL Spatial	Good (from Version 5.0)	Not all	NO	Open Source	Partial	Fast	Very good	YES, Python, JAVA or C++	Fair	ALL	RDBMS
Microsoft SQL Server	Spatial	Mediocre	Most	YES	~ 10 K	FULL	Very fast	Good	Complex, .NET	Fair	Most	RDBMS
SQLite	SpatialLite	Good	Most	NO	Open Source	Partial	Very fast, embedded	Good	YES, Most programmin g languages	Good	Most	RDBMS
SQL Anywhere	SQL Anywhere for Spatial Data	Weak	Not all	NO	~ 5 – 20 K	FULL	Fast	No	No	Weak	Win	RDBMS
NoSQL (MongoD B)	No SQL Spatial	Weak	Not all	NO	Open Source	FULL	Very fast	Cloud	YES, C++	Weak	All	Key-Value
MS Access	None	None	Encoded Geometr y	NO	~ 2K per year	Partial	Very fast	Very good	YES, VB	Weak	ALL	RDBMS

According to the evaluation table, we have selected PostgreSQL as the most suitable database management system for VISCA. Especially with the spatial extension PostGIS it is possible to create high-performance solutions for geodata processing and visualization. PostgreSQL also supports the conceptual workflow of importing triggered flood delineation data better than other DBMS due to the native handling of shapefiles and raster files.

The development of a large library of spatial functions, the enabling of spatial indices, the possibility to automate processing with Python scripts and the interoperability with GeoServer and other OGC conform software makes PostgreSQL superior over other database system candidates for VISCA.

Spatial Analysis component

PostGIS is the spatial extension of the object-relational database management system PostgreSQL. Through PostGIS it is possible to handle, store and create geographic data. It contains a large library of functions, spatial indices, coordinate systems, geometry transformation and storage. The following table gives a short overview about different spatial functions that can be implemented within VISCA:

Table 4: Examples of relevant PostGIS functions

PostGIS function	PostGIS nomenclature	Purpose
Buffer	ST_Buffer	Creating spatial buffers around objects, e.g. to examine possibly affected buildings or roads within a distance/buffer zone of a river
Intersection	ST_Intersection	Intersecting flood layer with other topographic objects (infrastructure, buildings) to determine affected features
JSON export	ST_AsGeoJSON	Export vector data to GeoJSON exchange format that can be integrated as a layer on mobile or web map applications
Transform and project	ST_Transform	Transforms dataset to another coordinate system or projection
Union	ST_Union	Groups different layers into a single layer (for the creation of daily flood layers)
Envelope	ST_Envelope	Creates the minimum bounding geometry to get the actual area of interest

Cartographic representation

As GeoServer supports the cartographic styling of features (in the case of VISCA delineation polygons) through Styled Layer Descriptors (SLDs), the published geodata will inherit also a consistent symbology before being provided to the visualization platform. The SLD will describe the colour, contour width and opacity of the delineation layer.

A SLD is a tag-structured XML-based file that can be assigned to all layers within a specific class. A small excerpt for the colouring of a delineation polygon (polygon fill and polygon contour) would be as following.

```
<PolygonSymbolizer>
  <Fill>
    <CssParameter name="fill">#58ACFA</CssParameter>
    <CssParameter name="fill-opacity">0.5</CssParameter>
  </Fill>
  <Stroke>
     <CssParameter name="stroke">#0B0B61</CssParameter>
     <CssParameter name="stroke-width">0.5</CssParameter>
  </Stroke>
</PolygonSymbolizer>
```

The most current OGC (Open Geospatial Consortium) Web Map Service (WMS) specification (http://www.opengeospatial.org/standards/wms) supports the ability for an information provider to specify very basic styling options by advertising a pre-set collection of visual portrayals for each available data set. However, while a WMS currently can provide the user with a choice of style options, the WMS can only tell the user the name of each style. It cannot tell the user what portrayal will look like on the map. More importantly, the user has no way of defining own styling rules. The ability for a human or machine client to define these rules requires a styling language that the client and server can both understand. This language can be used to portray the output of Web Map Servers, Web Feature Servers and Web Coverage Servers.

In many cases, however, the client needs some information about the data residing on the remote server before a reasonable request can be made. This led to the definition of new operations for the OGC services in addition to the definition of the styling language. There are two basic ways to style a data set. The simplest one is to colour all features the same way. For example, one can imagine a layer advertised by a WMS as “hydrography” consisting of lines (rivers and streams) and polygons (flood delineations). A user might want to tell the server to colour the insides of all polygons in a light blue, and colour the boundaries of all polygons and all lines in a darker blue. This type of styling requires no knowledge of the attributes or “feature types” of the underlying data, only a language with which to describe these styles. A more complicated requirement is to style features of the data differently depending on some attribute.

SLD profile of WMS defines the operation that fulfils this need, called DescribeLayer. This operation returns the feature types of the layer or layers specified in the request, and the attributes can be discovered with the DescribeFeatureType operation of a WFS interface or the DescribeCoverageType of a WCS interface.

3.2.3. Responsive Web Application

The Responsive Web Application plays a key role in the project, because it is the real point of interaction between the project stakeholders and the system. It needs to deliver and accurately display a high amount of data, avoiding latencies. Moreover, the contents must be adapted according to the device display size. Provided that requirements, a comparison (see Table 5: Comparison among the most widely used JavaScript frameworks) among the most actively used state-of-the-art JavaScript frameworks was made in order to choose the most suitable for developing the VISCA frontend and mobile application.

NPM Trends[11] provided the results shown in **Error! Reference source not found.**, which shows that Facebook R eact nowadays is more adopted compared to Angular by far.

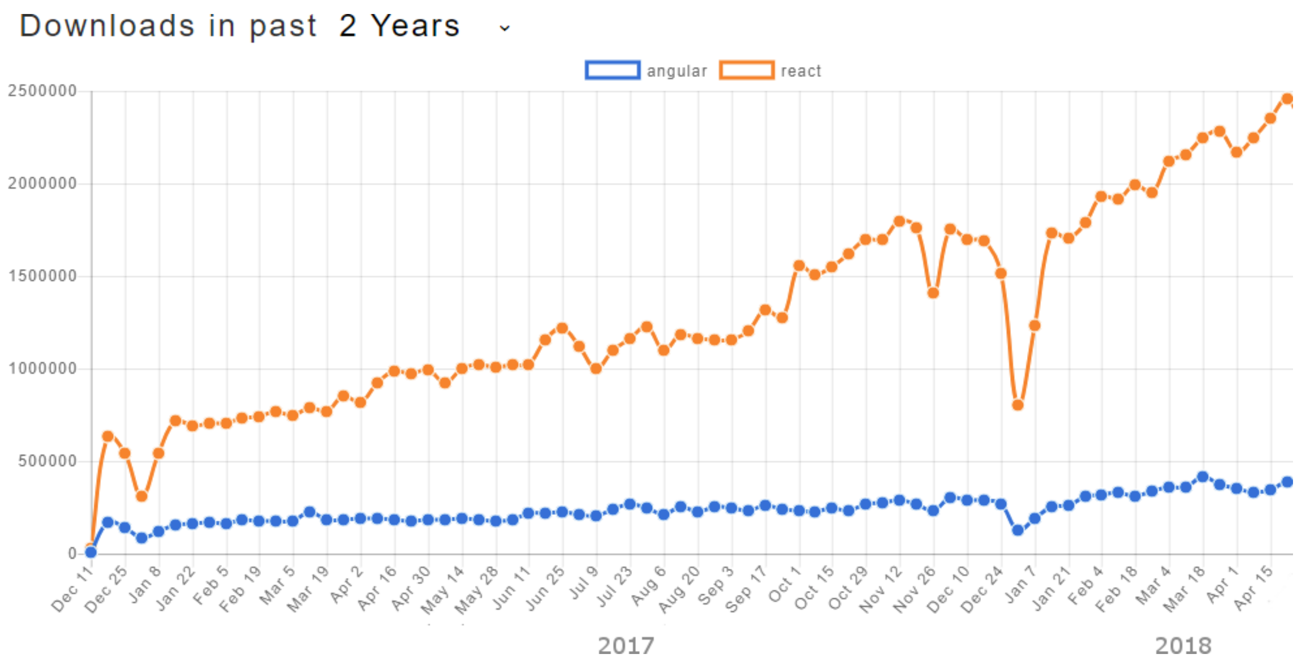


Figure 11: NPM trend plot about GitHub downloads among the best open-source JS frameworks.

In the past two years (since September 2018) Angular JS released four new versions with several breaking changes and, according to the following link[12], other important changes are foreseen in the short period. Because of its frequent updates, Angular it is not considered in this project as the best solution that could ensure stability for long time. Therefore, the choice was made on ReactJS, which exploits the redux-flux design pattern and has proven to be very efficient for both large and small applications on both mobile and desktop environments. The redux-flux pattern is a specific version of the one-way-flow pattern. Furthermore, the ReactJS developers' community is very large and there is good support and maintainability.

ReactJS actually is only the presentation layer of a React-Redux application, thus regarding mainly the user interface and routing. The point of strength of React is that its modules, called component, are designed for maximizing reusability, while performance are boosted by manipulating an optimized shadow DOM instead

[11] http://www.npmtrends.com/angular-vs-react
[12] https://angular.io/guide/releases

of accessing the actual DOM directly, propagating only necessary UI mutations. More details can be found on the React official website[13].

The other component of a React-Redux applications is Redux, which controls and models the application state and how state mutations are propagated to the views using the so-called providers. Redux pattern is a specific implementation of the flux pattern, which in turn belongs to the family of the one-way-flow patterns.

Redux thus controls how data are accessed, represented and mutated in the app state by describing how data sources have to be integrated into the app state. The state of a Redux application is saved to a single store, which may also be serialized, and it is created by composition of the connected Redux modules. More details can be found on the Redux documentation[14].

As a result of this analysis, the React+Redux solution is chosen.

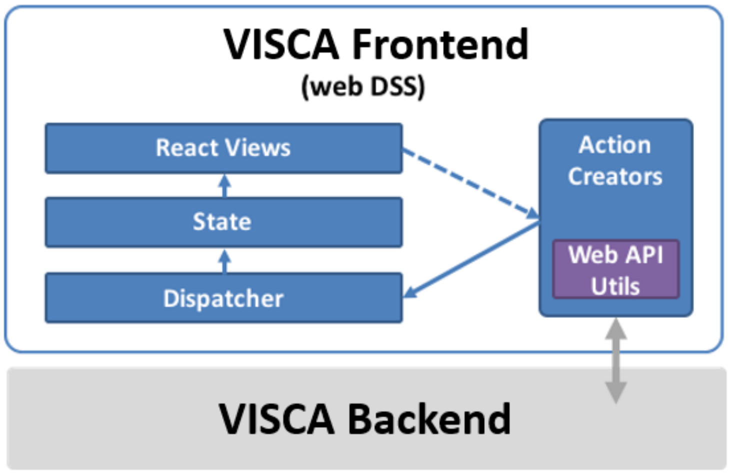


Figure 12: VISCA Front-end software architecture

The Redux-Flux pattern establishes a unidirectional data flow with a single store for the application state. This ensures that the views will always respect the changes of the app state, which are fired by actions. Actions can be related to UI interactions or other events. This allows to separate presentation from abstract representation of the app state, while boosting performances and simplifying component reusability and scope isolation.
The selected React-Redux design allows to separate view rendering from app state representation, writing better organized and more maintainable and reusable code.

[13] https://facebook.github.io/react/
[14] http://redux.js.org/

Table 5: Comparison among the most widely used JavaScript frameworks

Framework Comparison	Xamarin	Angular	REACT + REDUX
High code reuse	Low code **Not possible to do webapp**, hence no code reuse between webapp and mobile. Possible reuse of some classes between backend and mobile **Not free for commercial app**	Component based, high reusability. Compatibility between web and mobile with **Ionic** framework (Cordova-based), which is **not free for commercial app**. More research needed. **One Way Flow pattern**	Redux code can be shared between web and mobile app. React code is incompatible with React-Native (since RN uses native UI components), but Components can be used on mobile on a Cordova-based app. **One Way Flow pattern (Redux-Flux)**
Maturity, documentation, support	Mature, is around the internet since some years. Recently the Xamarin IDE was embedded into Visual Studio, also for Mac (Beta)	Out of beta since September 15, 2016. Good documentation, but not as mature as the other frameworks.	Very mature, used in production by big companies like Netflix, Facebook, Yahoo, Microsoft. Well documented and supported by Facebook.
Performance	Good, comparable with React Native	Very good performance, comparable with React. optimized for mobile	Very good, probably the fastest with support by big companies. Even more performant (and mostly compatible) solutions include Inferno and React. support for mobile is excellent
Modularity and maintainability	It's C#. Modularity strongly depends on pattern.	Based on typescript, which is different from Javascript (higher learning curve).	Modular by definition, code can easily be split and work can be divided among teams.
Future lookout	Microsoft bought it recently, so it is likely to see it around for the next years Appealing for .NET dev community. Interest not growing	Available Angular 6 as beta. Since 2016, four newer versions with breaking changes has been added. New versions are scheduled from the beginning of September 2018 (link)	High utilization, will be maintained for a long time. Strong increase in interest
Preference	**None webapp not possible**	**Mid**	**High**

The main components of this architecture are:

- **Action Creators**: these functions define which action are dispatched in reaction to events such as user interaction, network, system or sensor events. Action Creators can be synchronous or asynchronous. In order to communicate with the VISCA backend, Action Creators will take advantage of the Web API module.
- **Dispatcher**: depending on the triggered Action Creators, the dispatcher has the task of propagating events that mutate the app state. Since Redux contemplates a single store for the app state, all the actions, defined as plain objects, are dispatched by this central store.
- **State**: it encapsulates the state of the application and it is read-only, in the sense that neither views nor network calls can directly manipulate it. The way the state can mutate is defined through so called "reducers", which are pure functions that given the current state and an action, return the next state. It can be thought in terms of a Finite State Machine (FSM) whose states are

a composition of each module states, and reducers are a composition of each module reducers, exploiting modularity and reusability. Each module thus defines a portion of the app state that is combined to the other portions and stored into the single store. The state is therefore a hierarchical object tree which serves as single source of truth.

- **React Views**: roughly corresponding to the V of the classic MVC pattern, React Views are made of reusable components that only render a portion of the current app state, which is propagated by mean of immutable properties in a hierarchical way. User Interface (UI) interactions are then propagated to the app state by calling the Action Creators, that will update the app state and finally propagate to the interested view components, when needed.

The frontend will include an additional module, i.e., the Web API Utils. It will be tasked to act as a liaison with the backend, managing all interactions, both inbound and outbound.

3.2.4. Backend

The VISCA Backend is the module that centralizes the information managed by the other modules. It needs to be technologically detached from the other structures and must provide information related to the platform content.

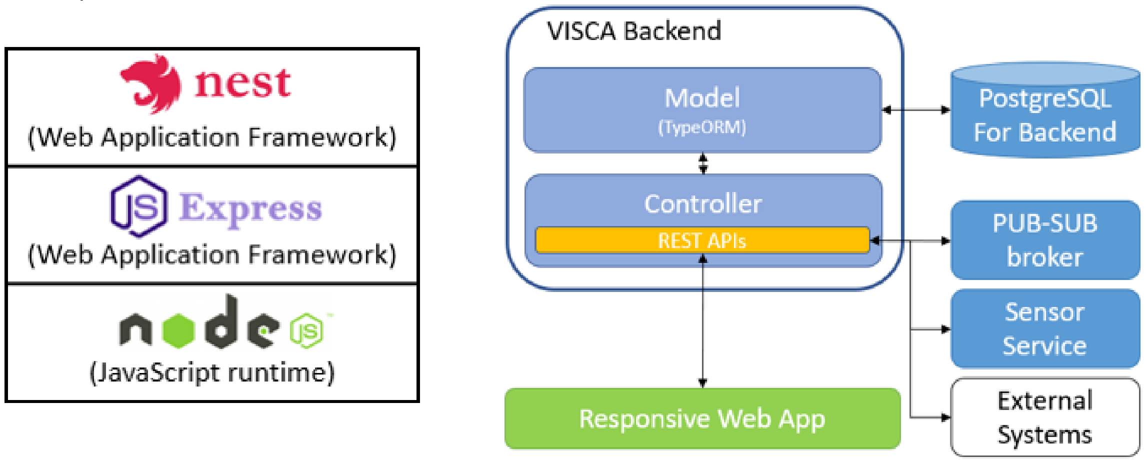


Figure 13: The Backend architecture and the adopted technologies. On the left side, the Nest.JS framework stack is shown. On the right side, a representation of the Backend architecture is presented.

It is developed with *Nest.JS*[15], a JavaScript Web Application Framework based on *Node.JS*[16] and *Express.JS*[17] frameworks. The first one is one of the most popular JavaScript Runtime Environment to build Server-side applications. NodeJS is an open source project, actively maintained since May 2009. Moreover, Express.JS is a JavaScript Web Application Framework which extends the Node.JS functionalities allowing the creation of RESTful APIs. Nest.JS adds another abstraction level, by

[15] https://nestjs.com/
[16] https://nodejs.org/it/
[17] http://expressjs.com/it/

integrating TypeORM[18], a TypeScript-based ORM frameworks which permits to easily interact with entities and properties stored in the database (according to the overall VISCA architecture, the *PostgreSQL for Backend* database).
These frameworks follow a classic **MVC** pattern, where the **M**odel is linked to the database entities by means of TypeORM and the **C**ontroller design, exposes and manages the REST APIs. The **V**iew component, foreseen in the classical pattern, is needless in the Backend, because all the information needed is provided through the Controller.

3.2.5. Sensor Service

The objective of the Sensor Service is to transfer data from sensors to the VISCA platform. Due to numerous clients which send data at the same time, the Sensor Service needs a fast structure that could deal with several requests and store incoming data. Once stored, data is processed and retrieved by the Backend. On purpose, a fast and lightweight architecture has been designed and shown in Figure 14.

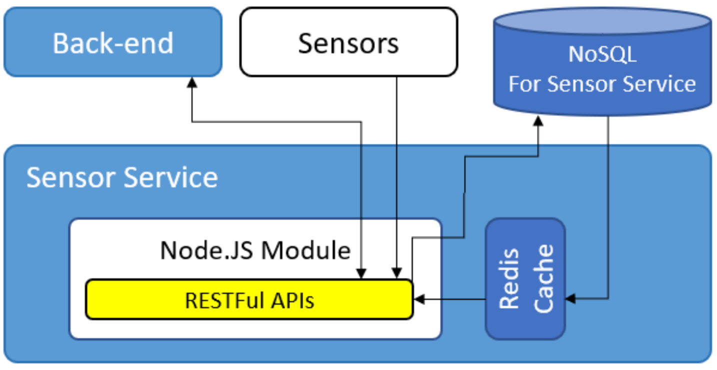


Figure 14: The Sensor Service architecture

The Sensor Service module leverages on the Node.JS component to provide RESTful APIs to interact with other modules. In order to guarantee velocity in dealing with several requests, no other frameworks are foreseen to extend Node.JS functionalities. This choice is advisable when the component to be developed needs to be very fast and it is logically simple, discouraged in the other situations. A NoSQL database is then adopted to store data coming from sensors Figure 15 shows a comparison among the most adopted non-relational databases supporting geospatial data: for this project, a MongoDB has been chosen. Considering that NoSQL databases are faster than relational databases during insertion operations (where new data are stored), but slower than relational databases during reading operations, a Redis Cache module is foreseen. Redis Cache is an open source, in-memory data structure: in this architecture it is adopted to temporally store data previously computed and yielded from the NoSQL database, in order to avoid further database requests when the same query is presented more times. Thus, when the Sensor Service receives a request concerning database reading operations, the Redis Cache is first checked whether it already contains the requested data, otherwise the database query is performed.

[18] http://typeorm.io/

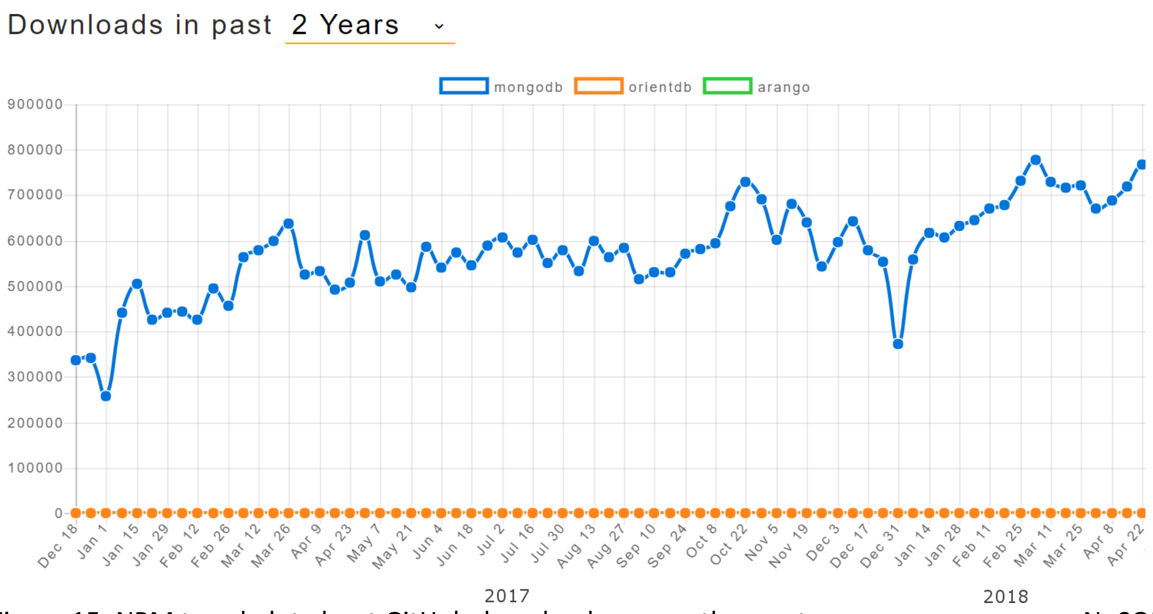


Figure 15: NPM trend plot about GitHub downloads among the most common open-source NoSQL databases supporting geospatial data

3.2.6. Architecture explanation of the software components in the external modules

For weather forecasts (short term, mid term) and climate, a High Performance Computing (HPC) cluster is needed to run the WRF model. For this project, a master node and 4 ASUS computing nodes with the following details are used:

- CPU: 2 x Intel XeonTM E5-2680 v4 processors with 14 cores (28 cores in total).
- Memory: 4 x 16 GB DDR4 2133 MHz ECC REG (64 GB in total).
- Disk: 2 x HD 2TB, SATA 6 Gb/s 7.200 r.p.m 3,5 128MB Enterprise Cloud Storage 512e.

This makes a total of 112 cpus and 256 GB of memory.

The HPC cluster is managed by an open source software ecosystem that provides a redundant master node, virtual machines to allocate the critical services, monitoring tools, centralized installation using Kickstart, a shared filesystem, a login node, LDAP and Munge authentication services and a centralized log facility. It is based on Linux Redhat Centos 7.3.

The WRF model has been compiled using the Intel C++ and Intel Fortran Compilers (version 2017.2), and the Intel MPI Library. The model is run through the Slurm batch jobs scheduler. Slurm has full control over CPU and memory usage, is integrated with MPI, supports job preemption and has suspension and resume capabilities.

The seasonal forecasts are computed using the BSC facilities: MareNostrum (Figure 16), which is a supercomputer based on Intel Xeon Platinum processors, Lenovo SD530 Compute Racks, a Linux Operating System and an Intel Omni-Path interconnection. See below a summary of the general-purpose cluster system:

- Peak Performance of 11.15 Petaflops
- 384.75 TB of main memory
- 3,456 nodes:
- 2x Intel Xeon Platinum 8160 24C at 2.1 GHz
- 216 nodes with 12x32 GB DDR4-2667 DIMMS (8GB/core)
- 3240 nodes with 12x8 GB DDR4-2667 DIMMS (2GB/core)
- Interconnection networks:
    - 100Gb Intel Omni-Path Full-Fat Tree
    - 10Gb Ethernet
- Operating System: SUSE Linux Enterprise Server 12 SP2

Figure 16: MareNostrum facility at BSC

In the next subsections, the process that is implemented within the external models is briefly described.

3.2.6.1. Short-term forecast

Operational short-term forecast of the most important meteorological fields in vineyards are going to be predicted with a regional weather forecast model, the Weather Research and Forecasting (WRF) Model. Within the scope of the VISCA project, an atmospheric modelling system state-of-the-art developed by the National Center for Atmospheric Research (NCAR) and the National Center for Environmental Prediction (NCEP). This regional model is essential to obtain accurate forecast for high horizontal resolution. The model are run and the data are stored and processed in the Meteosim cluster. Moreover, the output data is post-processed and formatted according to internal standards compatible with the INSPIRE directive.

WRF model solves the primitive equations of the atmosphere with different physical parameterizations of: boundary layers, surface radiation, clouds, etc. These parameters depend on the geographical area where the model runs. Within the scope of VSICA project and users requirements, a previous calibration and validation of the model will be carry out for three different demo-areas to determine the best option domains, model physics and model dynamics.

Based on the user requirements and for the project purposes, twice a day the WRF model on a 1 km horizontal resolution will run up to 48 h. The weather variables like temperature, precipitation, relative humidity, wind speed and downward short-wave flux radiation with a time resolution of 1 h will be provided to the VDI in geoJSON format.

3.2.6.2. Mid-term forecast

Operational mid-term forecast for temperature, precipitation, relative humidity, wind speed and downward short-wave flux radiation are downloaded from the National Oceanic and Atmospheric Administration (NOAA). The data used is from the Global Ensemble Forecast System (GEFS), a weather forecast model made up of 21 separate forecasts, or ensemble members. The data is downloaded in GRIB format, stored and processed in the Meteosim cluster.

The GEFS attempts to quantify the amount of uncertainty on a forecast by generating an ensemble of multiple forecasts, each minutely different, or perturbed, from the original observations. For the purposes of VISCA project, for each demo-site the forecasting data is taken for the different ensemble members and post-processed to obtain the probability of being in certain predefined bin.

The final product is delivered to the VDI in a CSV format as a forecast time series to be used by the phenological IRTA models. Moreover, this data is also provided in geoJSON format. The data is updated once per day where the time resolution is about 3 h and the forecasting length of 240h (10 days).

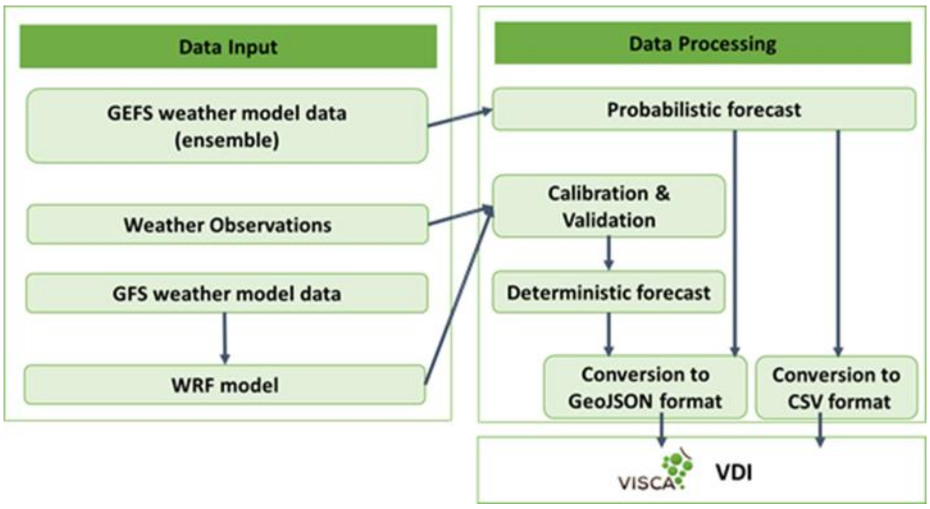


Figure 17. MET data processing flow chart regarding VISCA data (both short and medium-term data)

3.2.6.3. Climate change

The result of this task will be a report with information related to the climate change on Europe that should be useful to the vineyard management. This report includes a study of the climate change at high resolution from the projections of climate models in 2 different scenarios: RCP4.5 (moderate forcing) and RCP8.5 (strong forcing).

The task will be based on 2 data sources at European level (both available in netCDF format):

- Climate observations, from the last version of the E-OBS data set
- Dynamically-downscaled climate projections, from the WCRP CORDEX project (domain EUR-11)

This regards an analysis of precipitation and mean, maximum and minimum temperature projected for the next decades. Currently we are defining the preliminary results based on the analysis of the 2 data sources, but it will be focused on the study of the projected changes in climatology, variability and also, possibly, changes in many indices related with extreme events and other agricultural-specific. We will apply the Change Factor (CF) method in order to obtain projected values of temperature and precipitation.

The outcoming maps from the report will be provided to the VDI platform for its visualization in geoJSON format. The results will have a decadal frequency and a spatial resolution of 0.22x0.22 degrees.

All the necessary computation will be made at Meteosim's facilities, using our own equipment and storage.

3.2.6.4. Seasonal models

Operational seasonal predictions of temperature and precipitation are downloaded from the Copernicus Climate Change data Service (C3S) in GRIB format from the seasonal prediction systems considered, e. g. European Centre for Middle-range Weather Forecasting System 5 (SEAS5). These data are stored in the BSC infrastructure and formatted in NetCDF according to internal standards compatible with the INSPIRE directive.

The biases detected in the seasonal predictions of temperature and precipitation are corrected within the BSC, both from global and local perspectives. For that purpose, BSC applies a calibration method to improve the reliability of the forecasts and a downscaling technique to provide information in the demo-sites: Raïmat (Codorniu), Quinta-do-Ataide (Symington) and Mirabella (Mastroberardino).

The verification of these already bias-adjusted predictions against observational references is performed within the BSC, considering the re-forecasts available from the seasonal prediction systems covering the period 1981-2015. The used observational references are both the Japanese 55-year Reanalysis (JRA-55) and the in-situ observations from the three demo sites.

The final product to be delivered in the seasonal service consists in the tercile probability of the variables considered and ranked probability skill score, RPSS, as a skill measure. It is formatted into GEOJSON files and uploaded to the VDI. Simultaneously, the forecasts for the next seven months in the three demo-sites are formatted in a .csv format to be ingested by the phenological IRTA models. In Figure 18: Service's intermediate product development within the BSC we depict the workflow of the work performed within the BSC before uploading the data to the VDI.

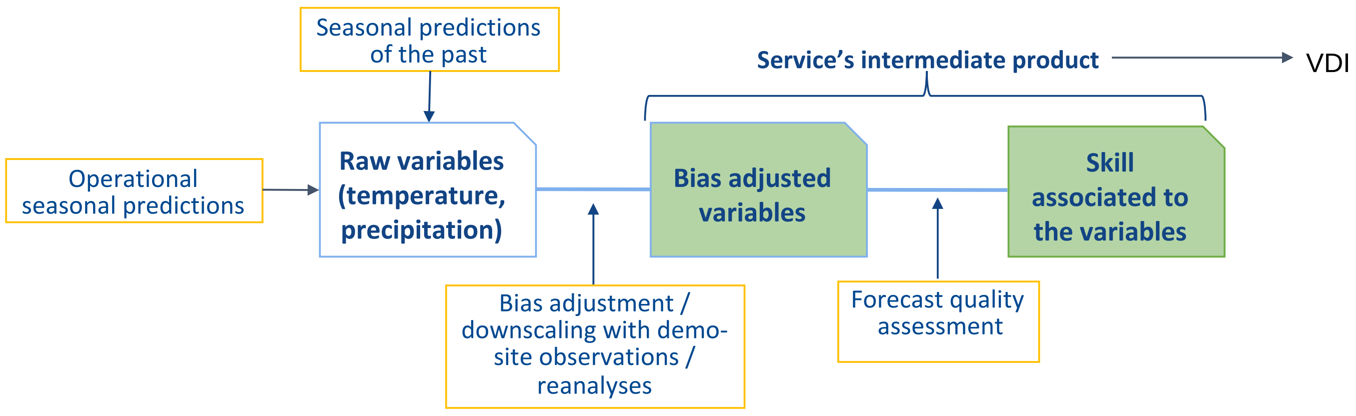

Seasonal predictions of the past
Service's intermediate product
VDI
Operational seasonal predictions
Raw variables (temperature, precipitation)
Bias adjusted variables
Skill associated to the variables
Bias adjustment / downscaling with demo-site observations / reanalyses
Forecast quality assessment

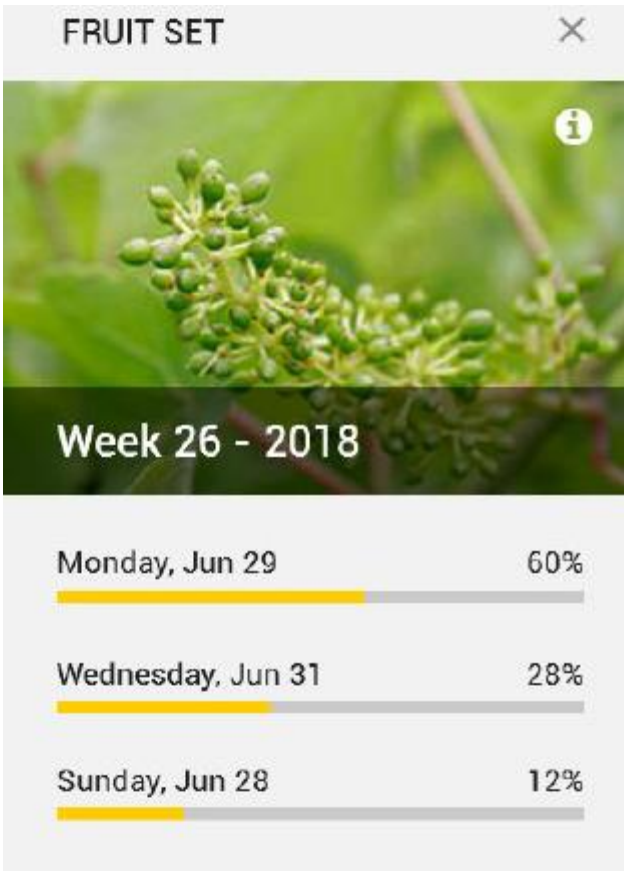
Figure 18: Service's intermediate product development within the BSC

3.2.6.5. Phenological models

Phenological forecasts are one of the core services of VISCA and it is highly important in order to improve the effectiveness of crop management, starting from the bud break up to the harvest. Phenological models run periodically, namely every week, and they estimate the occurrence of the phenological phases for all parcels of all users in the VISCA platform.

First, the phenological module gathers all input data from the platform, namely: parcel information (soil composition, variety, parcel orientation, spacing, etc.), past weather data, weather forecasts, irrigation done, phenological phases already achieved. All these data can be gathered using the back-end API (for vineyard and irrigation), the Visca Data Interface (VDI) (for weather forecasts) and the sensor service API (for past weather data). Next, the models have to run for all parcel, and finally provide the result, i.e., the phenological predictions, to the VISCA platform using the back-end API.

After the predictions become available, the responsive web application can show them to the user as shown in Figure 19, where a mock-up of the web app represents how to display a probabilistic phenological forecast computed starting from seasonal weather forecasts.


FRUIT SET

Week 26 - 2018

Monday, Jun 29 60%

Wednesday, Jun 31 28%

Sunday, Jun 28 12%

Figure 19: Example of phenological forecast computed with seasonal weather forecast.

4. Data Formats and Structure

In this section the main data entities are detailed.

First of all, the user profile is defined, which is composed by the information reported in Table 6. For each organization (end-user), the VISCA user are hierarchically structured in one or more company responsible (farmers), who are programmatically inserted, and a group of collaborators, who can be added by any organization responsible. The main user management process is depicted in Figure 20. Note that every user login into the system with user and password.

Table 6: Profile information of farmer (organization responsible) and the company collaborators

FARMER	COLLABORATOR
Username*	**Username***
Name*	Name*
Surname*	Surname*
Email*	Email*
Phone	Phone
Company (organization)	inherit Company (organization)
Address	inherit Address
Nationality	inherit Nationality

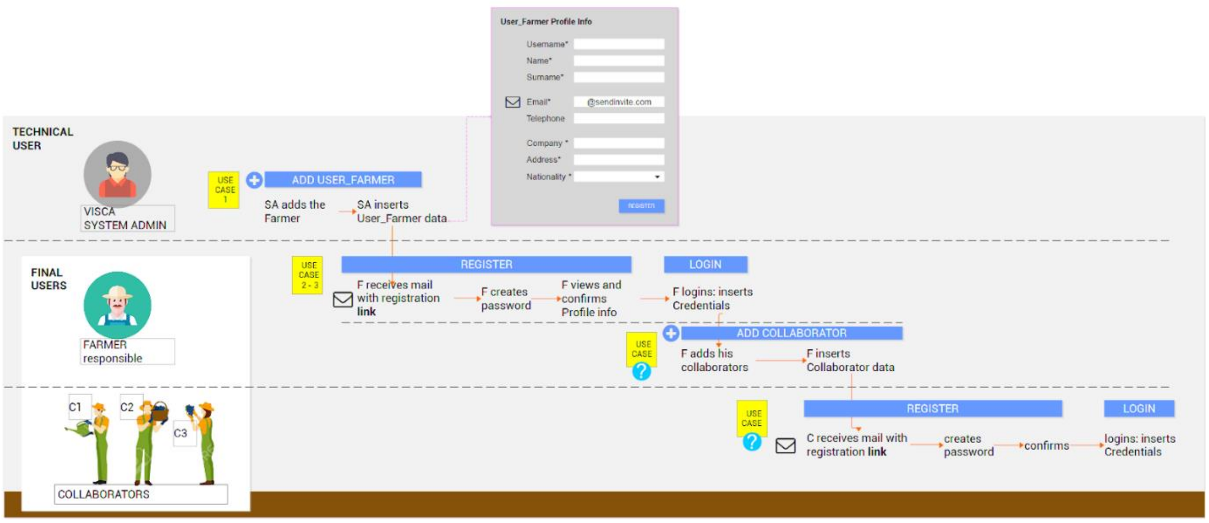


Figure 20: User registration procedure

The key information element of the VISCA system is the vineyard, which is owned by a company and that is divided into sub-elements called parcels. Parcels are considered homogeneous entities with respect to the grape variety, the soil composition, the orientation, the plant spacing, and in general to all characteristic of the vine. Together with the vineyard, another key element is the irrigation system, which is always owned by a company and it is installed in a given vineyard. The irrigation system is divided in irrigation blocks that can be separately controlled by the irrigation system and that can feature different irrigation techniques. After an in-depth revision of the information required in order to characterize the vineyard and the irrigation system, and to allow the execution of phenological and irrigation models, a common definition has been achieved and validated with end-users. (reported in Table 7).

Table 7: definition of initial information for vineyard and irrigation system.

Parameter	Unit of measure	Category	Format
Country	-	Vineyard	string
Company	-	Vineyard	string
VinayardName	-	Vineyard	string
ParcelCode	-	Vineyard	string
ParcelGeometry	-	Vineyard	geometry
Grape variety	-	Vineyard	string
Facing	degree	Vineyard	numeric
RowSpacing	meters	Vineyard	numeric
PlantSpacing	meters	Vineyard	numeric
Trellis	-	Vineyard	string
Orientation	degree	Vineyard	numeric
Soil texture (% sand)	percentage	Vineyard	numeric
Soil texture (% clay)	percentage	Vineyard	numeric
Soil Depth	meters	Vineyard	numeric
Slope	percentage	Vineyard	numeric
Trunk Height	meters	Vineyard	numeric
Rootstock	-	Vineyard	string
Emitter Discharge rate	mm/hour	Irrigation system	numeric
Type of irrigation system	-	Irrigation system	string
Emitters per plant		Irrigation system	Numeric
ParcelGeometry	-	Irrigation system	geometry

The aforementioned data are considered as initial and they do change very infrequently (range of 10 years). Conversely, each season the end-user can define some desired parameters, including the final quality traits, such as sugar, acidity alcoholic level, and the yield. All these parameters can be manually defined at the parcel level by the end-user. The full set of desired parameters (reported in Table 8) was subject to a long discussion between phenological modelers and end-users, and it also includes the stress thresholds that are needed by the irrigation model.

Table 8: Seasonal desired parameter for each parcel.

Parameter	Unit of measure	Category	Format
Stress threshold Preveraison	percentace ETc	Vineyard	numeric
Stress threshold Veraison	percentace ETc	Vineyard	numeric
Stress threshold Post Harvest	percentace ETc	Vineyard	numeric
Technology	-	Vineyard	String
Sugar Level	Brix	Vineyard	numeric
Acidity Level	grams/litre tartaric acid	Vineyard	numeric
Alcoholic level	% vol	Vineyard	numeric
Yield	kg/Ha	Vineyard	numeric

While the season is ongoing, VISCA will weekly predict the phenological phases and the irrigation to be applied in each irrigation block. The former has been collectively defined according to the Baggiolini scale, while the latter is expressed in mm/day. Despite the models will be not ready to predict the final quality traits and the yield, such parameters have been considered and left for future use. The full list of predicted parameters is reported in Table 9.

Table 9: Main predicted parameters.

Parameter	Unit of measure	Category	Format	Description
BB Bud Break (C)	-	Phenological	week of year ISO 8160	Baggiolini: 50-60% - green tissue on the full cordon
B Blooming (I)	-	Phenological	week of year ISO 8160	Baggiolini: 50-60% - flowers on all cordon
FS Fruit Set (J)	-	Phenological	week of year ISO 8160	Baggiolini: full
VR Veraison (M)	-	Phenological	week of year ISO 8160	Baggiolini: nearly full (>90%)
HV Harvest	-	Phenological	week of year ISO 8160	Depending on end users previous criteria (Desired: Sugar, Acidity and so on)
Leaf fall	-	Phenological	week of year ISO 8160	50-60% of dropped leaves
Irrigation	mm/day	Vineyard	numeric	
Sugar Level	Brix	Vineyard	numeric	future
Acidity Level	grams/litre tartaric acid	Vineyard	numeric	future
Alcoholic level	% vol	Vineyard	numeric	future
Yield	kg/ha	Vineyard	Numeric	future

Finally, end-user can declare at the parcel level the achieved parameters that include the phenological phases, the applied technology (crop forcing, canopy management), the final quality traits and yield. Note that the achieved parameters will be used by the phenological models in order to have the initial conditions for the predictions. The full list of the aforementioned achieved parameter can be obtained by merging desired and predicted, and it is omitted for brevity.

Additionally, the achieved parameters contain the irrigation applied and the weather as observed from the stations owned by the Company or in close proximity of the considered vineyards. Given the heterogeneity of available weather stations, a generalized structure has been defined in order to handle any kind of sensor, fixed or moving. Note that the weather station is represented by the entity Station, which can be equipped my one or more sensors (e.g., temperature, humidity, pressure, etc..). Every sensor outputs streams of homogeneous measurements (e.g., average temperature, max temperature, etc..). The Entity Relation scheme related to the weather station which is implemented within the Sensor Service is reported in Figure 21.

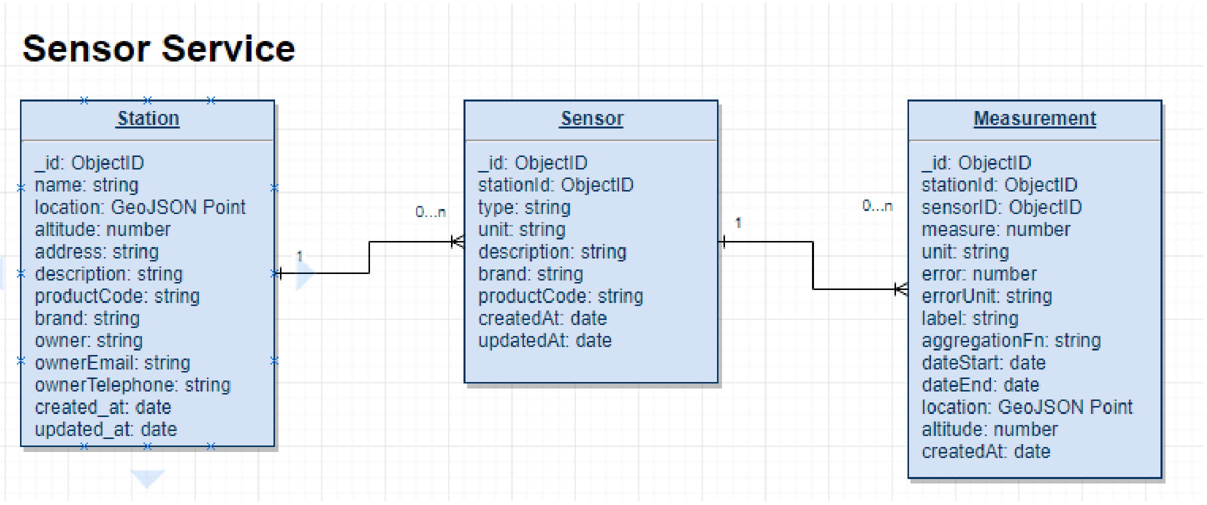


Figure 21: Entity Relation scheme of the weather station, which is implemented within the Sensor Service.

In order to summarize the definition done, the Overall Entity relation scheme as implemented by the back-end is summarized in Figure 22.

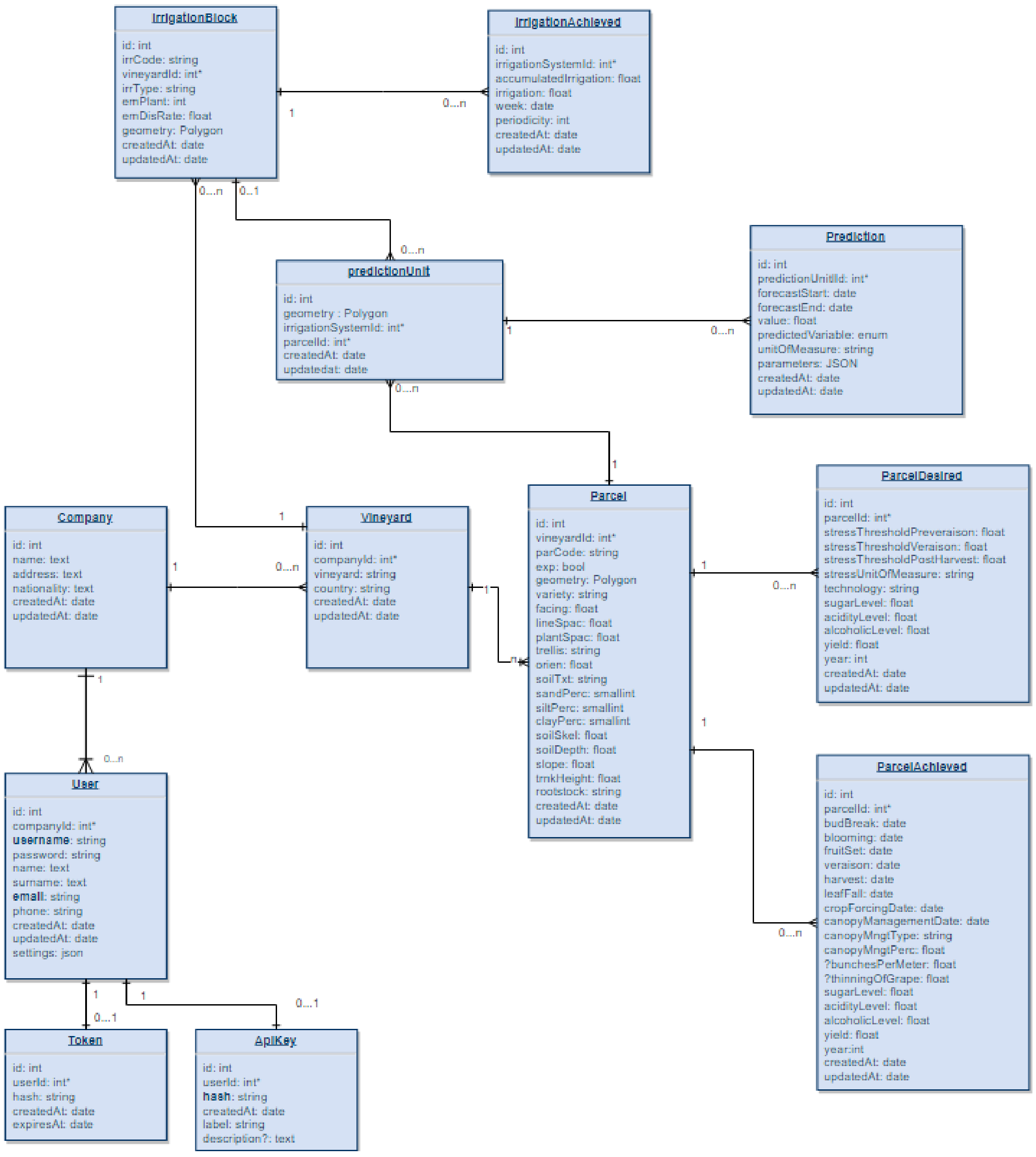


Figure 22: Overall back-end Entity Relation (ER) diagram.

Note that in the ER diagram it is reported the entity Prediction Unit. This concept has been introduced in order to allow the phenological model to run on homogeneous area by vineyard characteristic and irrigation. This concept was strictly needed because the irrigation system can be deployed with a specific geographical layout that can be different from the vineyard (parcel) one.

The data specification activity encompassed also the definition of the INSPIRE compliant metadata and the weather forecast data, which are fully listed in Appendix I and Appendix II, respectively.

The data format of all data exchange related to the Application Program Interfaces of all software module is JSON[19] or geoJSON[20], according to the cases. These choice results from JSON being the de-facto standard in all web-based application around the world.

5. Technical Requirements

This section lists the initial technical requirements identified so far, which are reported in Table 5-1 that gathers all technical requirements resulting from the preliminary analysis done with all partners. This initial list will be updated in the subsequent deliverables.

Table 10. Initial list of technical requirements

Req. code	Category	Description	KPIs
TR1	General	Service availability (all services)	99.9% on a yearly basis
TR2	Models	Data sources that will be used for forecast computation	Vineyard data, irrigation, weather, weather forecasts
TR3	Models	Requirements for weather forecast models	See Table 11
TR4	Models	Requirements for seasonal forecast models	See Table 12
TR5	Models	Requirements for phenological forecast models	See Table 13
TR6	Models	Requirements for irrigation forecast models	See Table 14
TR7	Models	Requirements for decadal climatic models	See Table 15
TR8	Responsive Web Applicaton	Browser compatibility	Chrome 53+ Firefox 49+ Edge 38+
TR9	GeoServer	Service availability	99.9% on a yearly basis
TR10	GeoServer	Map layer visualisation on web-based frontend	Within 3 seconds
TR11	GeoServer	Service completeness (all map tiles shown)	In 9 of 10 cases all map tiles of the required layer are loaded

[19] https://www.json.org/
[20] http://geojson.org/

Table 11. Detail KPIs for weather forecasts

Characteristics	**Models**	
	WRF	**GEFS**
Forecast Intervals [h]	Up to 48	Up to 240
Update intervals [h]	12	24
Variables	Temperature Wind Speed Precipitation Global Radiation Relative Humidity	Temperature Wind Speed Precipitation Global Radiation Relative Humidity
Resolution	1 km	demo-point (original data at 0.5 degree)
Area	demo-area	Europe

Table 12. Detail KPIs for seasonal forecast models

Characteristics	**Models**
	ECMWF System 5 (SEAS5)
Forecast Intervals [h]	1-month / 3-months (season)
Update intervals [h]	Every month
Variables	Tmax, Tmin, Tmean, Precip
Resolution	0.7 x 0.7 degrees
Area	Global

Table 13. Detail KPIs for phenological models

Characteristics	**Models**			
	BRIN	Berry Model	Crop-Syst	Leaf Model
Forecast Intervals [h]	168	168	168	Up to 240
Update intervals [h]	Same as seasonal forecast	Same as seasonal forecast	Same as seasonal forecast	24
Variables	Daily Mean Temperature (past and forecasted) Start Month of Seasonal Forecast Crop forcing date	Daily Mean Temperature (past and forecasted) Start Month of Seasonal Forecast Crop forcing date	Daily Mean Temperature (past and forecasted) Start Month of Seasonal Forecast Crop forcing date	Berries sugar content Leaf fall date
Resolution	Parcel	Parcel	Parcel	Plant scale (average)
Area	Demo-area	Demo-area	Demo-area	Demo site

Table 14. Detail KPIs for irrigation models

Characteristics	**Models**
	VSIM
Forecast Intervals [h]	168
Update intervals [h]	168
Variables	Temperature Wind Speed Precipitation Global Radiation Relative Humidity Stress threshold Clay percentage Silt percentage Gravel fraction Latitude Longitude Altitude Crop Forcing Date
Resolution	168
Area	168

Table 15. Detail KPIs for decadal climatic models[21]

Characteristics	**Models**
	CCCma-CanESM2, CNRM-CERFACS-CNRM-CM5, ICHEC-EC-EARTH, IPSL-IPSL-CM5A-MR, MIROC-MIROC5, MOHC-HadGEM2-ES, MPI-M-MPI-ESM-LR, NCC-NorESM1-M
Forecast Intervals [years]	10
Variables	Temperature (mean, max., min.) Precipitation
Resolution	0.22° × 0.22°

[21] These models come from CORDEX EUR-11 data set and they have been dynamically downscaled

Appendix I

In Table 16, the INSPIRE[22] compliant metadata are listed. It has been defined selecting the mandatory field of the INSPIRE directive and it is applied to all data that will be inserted into the Geo Importer because through the VDI it implements a data repository capable of sharing data (files) and service map layers through OGC standards.

Table 16: INSPIRE compliant metadata

Minimum set of metadata elements necessary to comply with Directive 2007/2/EC

A) Metadata according to INSPIRE Metadata Regulation (COMMISSION REGULATION (EC) No 1205/2008 of 3 December 2008 implementing Directive 2007/2/EC of the European Parliament and of the Council as regards metadata)

ref	**group of metadata elements**	**metadata element**	**description**
1	IDENTIFICATION		
1.1		Resource Title	This a characteristic, and often unique, name by which the resource is known. The value domain of this metadata element is free text.
1.2		Resource Abstract	This is a brief narrative summary of the content of the resource. The value domain of this metadata element is free text.
1.3		Resource Type	This is the type of resource being described by the metadata. The value domain of this metadata element is defined in Part D.1. 1.1. Spatial data set series (series) 1.2. Spatial data set (dataset) 1.3. Spatial data services (services)

[22] https://inspire.ec.europa.eu/

1.4		Resource Locator	The resource locator defines the link(s) to the resource and/or the link to additional information about the resource. The value domain of this metadata element is a character string, commonly expressed as uniform resource locator (URL).
1.5		Unique resource identifier	A value uniquely identifying the resource. The value domain of this metadata element is a mandatory character string code, generally assigned by the data owner, and a character string namespace uniquely identifying the context of the identifier code (for example, the data owner).
1.6		Coupled Resource	If the resource is a spatial data service, this metadata element identifies, where relevant, the target spatial data set(s) of the service through their unique resource identifiers (URI). The value domain of this metadata element is a mandatory character string code, generally assigned by the data owner, and a character string namespace uniquely identifying the context of the identifier code (for example, the data owner).
1.7		Resource language	The language(s) used within the resource. The value domain of this metadata element is limited to the languages defined in ISO 639-2.
2	CLASSIFICATION OF SPATIAL DATA AND SERVICES		
2.1		Topic category	The topic category is a high-level classification scheme to assist in the grouping and topic-based search of available spatial data resources. The value domain of this metadata element is defined in Part D.2. (Topic categories in accordance with EN ISO 19115) - see CELEX_32008R1205_EN_TXT.pdf for details
2.2		Spatial Data Service Type	This is a classification to assist in the search of available spatial data services. A specific service shall be categorised in only one category. The value domain of this metadata element is defined in Part D.3. (discovery, view, download, transformation, invoke)
3	KEYWORD		If the resource is a spatial data service, at least one keyword from Part D.4 shall be provided. If a resource is a spatial data set or spatial data set series, at least one keyword shall be provided from the general environmental multilingual thesaurus (GEMET) describing the relevant spatial data theme as defined in Annex I, II or III to Directive 2007/2/EC. For each keyword, the following metadata elements shall be provided:

3.1		Keyword value	The keyword value is a commonly used word, formalised word or phrase used to describe the subject. While the topic category is too coarse for detailed queries, keywords help narrowing a full text search and they allow for structured keyword search. The value domain of this metadata element is free text.
3.2		Originating controlled vocabulary	If the keyword value originates from a controlled vocabulary (thesaurus, ontology), for example GEMET, the citation of the originating controlled vocabulary shall be provided. This citation shall include at least the title and a reference date (date of publication, date of last revision or of creation) of the originating controlled vocabulary. There can be more than one because the keywords can own to more than one vocabulary
4	GEOGRAPHIC LOCATION		
4.1		Geographic Bounding Box	This is the extent of the resource in the geographic space, given as a bounding box. The bounding box shall be expressed with westbound and eastbound longitudes, and southbound and northbound latitudes in decimal degrees, with a precision of at least two decimals.
5	TEMPORAL REFERENCE		This metadata element addresses the requirement to have information on the temporal dimension of the data as referred to in Article 8(2)(d) of Directive 2007/2/EC. At least one of the metadata elements referred to in points 5.1 to 5.4 shall be provided. The value domain of the metadata elements referred to in points 5.1 to 5.4 is a set of dates. Each date shall refer to a temporal reference system and shall be expressed in a form compatible with that system. The default reference system shall be the Gregorian calendar, with dates expressed in accordance with ISO 8601.
5.1		Temporal extent	The temporal extent defines the time period covered by the content of the resource. This time period may be expressed as any of the following: — an individual date, - an interval of dates expressed through the starting date and end date of the interval, — a mix of individual dates and intervals of dates.
5.2		Date of publication	This is the date of publication of the resource when available, or the date of entry into force. There may be more than one date of publication.
5.3		Date of last revision	This is the date of last revision of the resource, if the resource has been revised. There shall not be more than one date of last revision.
5.4		Date of creation	This is the date of creation of the resource. There shall not be more than one date of creation.

6	QUALITY AND VALIDITY		
6.1		Lineage	This is a statement on process history and/or overall quality of the spatial data set. Where appropriate it may include a statement whether the data set has been validated or quality assured, whether it is the official version (if multiple versions exist), and whether it has legal validity. The value domain of this metadata element is free text.
6.2		Spatial Resolution	Spatial resolution refers to the level of detail of the data set. It shall be expressed as a set of zero to many resolution distances (typically for gridded data and imagery-derived products) or equivalent scales (typically for maps or map-derived products). An equivalent scale is generally expressed as an integer value expressing the scale denominator. A resolution distance shall be expressed as a numerical value associated with a unit of length.
7	CONFORMITY		The requirements referred to in Article 5(2)(a) and Article 11(2)(d) of Directive 2007/2/EC relating to the conformity, and the degree of conformity, with implementing rules adopted under Article 7(1) of Directive 2007/2/EC shall be addressed by the following metadata elements:
7.1		Specification	This is a citation of the implementing rules adopted under Article 7(1) of Directive 2007/2/EC or other specification to which a particular resource conforms. A resource may conform to more than one implementing rules adopted under Article 7(1) of Directive 2007/2/EC or other specification. This citation shall include at least the title and a reference date (date of publication, date of last revision or of creation) of the implementing rules adopted under Article 7(1) of Directive 2007/2/EC or of the specification.
7.2		Degree	This is the degree of conformity of the resource to the implementing rules adopted under Article 7(1) of Directive 2007/2/EC or other specification. The value domain of this metadata element is defined in Part D.
8	CONSTRAINT RELATED TO ACCESS AND USE		
8.1		Conditions for Access and Use	This metadata element defines the conditions for access and use of spatial data sets and services, and where applicable, corresponding fees as required by Article 5(2)(b) and Article 11(2)(f) of Directive 2007/2/EC. The value domain of this metadata element is free text. The element must have values. If no conditions apply to the access and

			use of the resource, 'no conditions apply' shall be used. If conditions are unknown, 'conditions unknown' shall be used. This element shall also provide information on any fees necessary to access and use the resource, if applicable, or refer to a uniform resource locator (URL) where information on fees is available. Additional Requirements on Metadata according to regulation 1312/2014: The technical restrictions applying to the access and use of the spatial data service shall be documented in the metadata element "CONSTRAINT RELATED TO ACCESS AND USE" set out in Regulation (EC) No 1205/2008.
8.2		Limitations on Public Access	When Member States limit public access to spatial data sets and spatial data services under Article 13 of Directive 2007/2/EC, this metadata element shall provide information on the limitations and the reasons for them. If there are no limitations on public access, this metadata element shall indicate that fact. The value domain of this metadata element is free text.
9	RESPONSIBLE ORGANISATION		Organisations responsible for the establishment, management, maintenance and distribution of spatial data sets and services
9.1		Responsible party	This is the description of the organisation responsible for the establishment, management, maintenance and distribution of the resource. This description shall include: — the name of the organisation as free text, — a contact e-mail address as a character string. Additional Requirements on Metadata according to regulation 1312/2014: The responsible party set out in Regulation (EC) No 1205/2008 shall at least describe the custodian responsible organisation, corresponding to the Custodian responsible party role set out in Regulation (EC) No 1205/2008.
9.2		Responsible party role	This is the role of the responsible organisation. The value domain of this metadata element is defined in Part D.
10	METADATA ON METADATA		
10.1		Metadata Point of Contact	This is the description of the organisation responsible for the creation and maintenance of the metadata. This description shall include: — the name of the organisation as free text, — a contact e-mail address as a character string.
10.2		Metadata Date	The date which specifies when the metadata record was created or updated. This date shall be expressed in conformity with ISO 8601.

10.3		Metadata Language	This is the language in which the metadata elements are expressed. The value domain of this metadata element is limited to the official languages of the Community expressed in conformity with ISO 639-2.

B) Additional metadata elements according to INSPIRE Implementing Rules for interoperability of spatial data sets and services (Commission Regulation (EU) No 1089/2010; Art. 13 Metadata required for Interoperability) and relevant amendments

		Coordinate Reference System	Description of the coordinate reference system(s) used in the data set.
		Distribution Format	Description of the computer language construct(s) specifying the representation of data objects in a record, file, message, storage device or transmission channel.
		Spatial Representation Type	The method used to spatially represent geographic information. Codelist (see B.5.26 of ISO 19115), following INSPIRE Data specifications only vector, grid and tin should be used.
		Temporal reference system	Description of the temporal reference systems used in the dataset.
		Character encoding	The character encoding used in the data set.

Additional metadata have been defined in order to manage the internal operations. The most important additional metadata is the viscaTask, which is used in order to identify a given data type. The list of additional metadata is reported in Table 17.

Table 17: List of additional VISCA metadata

11	**VISCA internal metadata**		Metadata required by VISCA
11.1		filename	The final filenames have to be generated by the VDI. Filename also used by Importer in case the file is raster and must be served as a map.
11.2		lead_times	array of lead times that matches the files sent to the VDI (right ordering must be kept to associate each file with the correct lead time and create the right filename according to the naming convention)
11.3		visualization_orders	array of visualization order that is used from the frontend to properly stack layers (right ordering must be kept to associate each file with the correct visualization order)
11.4		source_name	comma separated values of the source used. E.g., satellite names, model names
11.5		acquisition_date	satellite acquisition date
11.6		layer_attributes	to handle attributes of the entire layer
11.7		**VISCA task**	VISCA task in number of four digits, where the first two digits are the task number, reference to the Gantt, followed by an incremental number that specifies the data type

Appendix II

This appendix reports the list of the weather forecasts that will be sent to the VDI. The grey coloured lines represent data that will not be transformed as map layers.

Table 18: List of weather forecasts that will be produced and sent to the VDI

viscatask	Category	Measure	Type	Produced by	Spatial resolution	Update frequency	Temporal resolution	Maximum lead time	Values
1001	Weather station data	many							
2201	Short Term Weather forecast	Rain	Deterministic	MeteoSim	1km	12h	1h	up to 48 h	in mm for each polygon
2202	Short Term Weather forecast	Wind Speed	Deterministic	MeteoSim	1km	12h	1h	up to 48 h	in m/s for each polygon
2203	Short Term Weather forecast	Temp	Deterministic	MeteoSim	1km	12h	1h	up to 48 h	in Celsius for each polygon
2204	Short Term Weather forecast	Relative Humidity	Deterministic	MeteoSim	1km	12h	1h	up to 48 h	in percentage for each polygon
2205	Short Term Weather forecast	Global radiation	Deterministic	MeteoSim	1km	12h	1h	up to 48 h	W m-2
2220	Short Term Weather forecast	ETo	Deterministic	IRTA	1km	12h	24h	up to 48 h	mm/day
2251	Medium Term Weather forecast	Rain	Probabilistic	MeteoSim	0.5 degree	24h	3h	240h	median (used to define the polygon) and in the properties the median plus array of probabilities for each bin (0,0-0.1,0.1-10, >10) in mm/day

2252	Medium Term Weather forecast	Rain, wind Speed, Temp, Relative Humidity, Global Radiation	Probabilistic	MeteoSim	0.5 degree	24h	3h	240h	all members
2253	Medium Term Weather forecast	Wind Speed	Probabilistic	MeteoSim	0.5 degree	24h	3h	240h	median (used to define the polygon) and in the properties the median array of probabilities for each bin (0,0-0.5,0.5-2,>2) in m/s
2254	Medium Term Weather forecast	Temp	Probabilistic	MeteoSim	0.5 degree	24h	3h	240h	median (used to define the polygon) and in the properties the median plus an array of probabilities in bins of 2 degree starting from -18 up to 44
2255	Medium Term Weather forecast	Relative Humidity	Probabilistic	MeteoSim	0.5 degree	24h	24h	240h	median (used to define the polygon) and in the properties the median array of probabilities for each bin (0-30,30-60,60-90, 90-100) in percentage
2256	Medium Term Weather forecast	Maximum Temperature	Probabilistic	MeteoSim	0.5 degree	24h	24h	240h	median (used to define the polygon) and in the properties the median plus an array of probabilities in bins of 2 degree starting from -18 up to 44
2257	Medium Term Weather forecast	Minimum Temperature	Probabilistic	MeteoSim	0.5 degree	24h	24h	240h	median (used to define the polygon) and in the properties the median plus an array of probabilities in bins of 2 degree starting from -18 up to 44
2258	Medium Term Weather forecast	Global radiation	Probabilistic	MeteoSim	0.5 degree	24h	24h	240h	median (used to define the polygon) and in the properties the median plus an

									array of probabilities in bins of 64 W/m2 starting from 0 up to 1024
2260	Medium Term Weather forecast	ETo	Probabilistic	IRTA	0.5 degree	24h	24h	240h	number
2601	Seasonal Forecast	Rain	Probabilistic	BSC	25km	1 month	1 months	6 months	array of probabilities for each bin (below, normal, above) together the thresholds and the skill
2651	Seasonal Forecast	Rain	Probabilistic	BSC	1 point for each weather station	1 month	1 months	6 months	array of probabilities for each bin (below, normal, above) together thresholds and the skill
2602	Seasonal Forecast	Rain	Probabilistic	BSC	25km	1 month	3 months	6 months	array of probabilities for each bin (below, normal, above) together thresholds and the skill
2652	Seasonal Forecast	Rain	Probabilistic	BSC	1 point for each weather station	1 month	3 months	6 months	array of probabilities for each bin (below, normal, above) together the thresholds and the skill
2660	Seasonal Forecast	Rain	Probabilistic	BSC	1 point for each weather station	1 month	1 months	6 months	all members
2603	Seasonal Forecast	Temp	Probabilistic	BSC	25km	1 month	1 months	6 months	array of probabilities for each bin (below, normal, above) together the thresholds and the skill
2653	Seasonal Forecast	Temp	Probabilistic	BSC	1 point for each	1 month	1 months	6 months	array of probabilities for each bin (below, normal, above) together the thresholds and the skill

					weather station				
2604	Seasonal Forecast	Temp	Probabilistic	BSC	25km	1 month	3 months	6 months	array of probabilities for each bin (below, normal, above) together the thresholds and the skill
2654	Seasonal Forecast	Temp	Probabilistic	BSC	1 point for each weather station	1 month	3 months	6 months	array of probabilities for each bin (below, normal, above) together the thresholds and the skill
2670	Seasonal Forecast	Temp	Probabilistic	BSC	1 point for each weather station	1 month	1 months	6 months	all members
2501	Climate Forecast	Temp	Probabilistic	MeteoSim	0.11/0.22 deg	no update	10 y	up to year 2070	array of statistics for each season (mean, mean change, uncertainty, ...)
2502	Climate Forecast	Rain	Probabilistic	MeteoSim	0.11/0.22 deg	no update	10 y	up to year 2070	array of statistics for each season (mean, mean change,uncertainty, ...)