D5.2: 1st version of integrated and tested APOLLO platform

WP5 – Platform development

Authors: Ioulia Anastasiadou, Apostolos Agrafiotis, Stelios Kotsopoulos, Dimitris Messinis

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687412.
D5.2: 1st version of integrated and tested APOLLO platform

Disclaimer

Any dissemination of results reflects only the author’s view and the European Commission is not responsible for any use that may be made of the information it contains.

Copyright message

© APOLLO Consortium, 2016
This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.
D5.2: 1st version of integrated and tested APOLLO platform

**Document Information**

<table>
<thead>
<tr>
<th>Grant Agreement Number</th>
<th>Acronym</th>
<th>APOLOGO</th>
</tr>
</thead>
<tbody>
<tr>
<td>687412</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Full Title**
Advisory platform for small farms based on earth observation

**Horizon 2020 Call**
EO-1-2015: Bringing EO applications to the market

**Type of Action**
Innovation Action

**Start Date**
1st May 2016

**Duration**
34 months

**Project URL**
-

**Document URL**
-

**EU Project Officer**
Kristiina URPALAINEN-MENON

**Project Coordinator**
Polimachi Simeonidou

**Deliverable**
D5.2 1st version of integrated and tested APOLLO platform

**Work Package**
WP5 – Platform development and integration

**Date of Delivery**

<table>
<thead>
<tr>
<th>Contractual</th>
<th>M12</th>
<th>Actual</th>
<th>M13</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEMO</td>
<td></td>
<td>Dissemination Level</td>
<td>Public</td>
</tr>
</tbody>
</table>

**Nature**

**Lead Beneficiary**
DRAXIS

**Lead Author**
Ioulia Anastasiadou

**Other authors**
Stelios Kotsopoulos, Dimitris Messinis, Apostolos Agrafiotis

**Reviewer(s)**
UBFCE

**Keywords**

**Document History**

<table>
<thead>
<tr>
<th>Version</th>
<th>Issue Date</th>
<th>Stage</th>
<th>Changes</th>
<th>Contributor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>03/04/2017</td>
<td>Draft</td>
<td>Initial structure</td>
<td>DRAXIS</td>
</tr>
<tr>
<td>0.2</td>
<td>11/04/2017</td>
<td>Draft</td>
<td>Contributed the APOLLO architecture</td>
<td>DRAXIS</td>
</tr>
<tr>
<td>0.3</td>
<td>14/04/2017</td>
<td>Draft</td>
<td>Contributed App functionality chapter</td>
<td>DRAXIS</td>
</tr>
<tr>
<td>0.4</td>
<td>17/04/2017</td>
<td>Draft</td>
<td>Contributed the APOLLO progress</td>
<td>DRAXIS</td>
</tr>
<tr>
<td>0.5</td>
<td>19/04/2017</td>
<td>Draft</td>
<td>Refinements</td>
<td>DRAXIS</td>
</tr>
<tr>
<td>0.6</td>
<td>20/04/2017</td>
<td>Draft</td>
<td>Contributed Next steps and executive summary</td>
<td>DRAXIS</td>
</tr>
</tbody>
</table>
D5.2: 1st version of integrated and tested APOLLO platform

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Stage</th>
<th>Notes</th>
<th>Reviewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>20/04/2017</td>
<td>Draft</td>
<td>Initial version sent to reviewers</td>
<td>DRAXIS</td>
</tr>
<tr>
<td>1.1</td>
<td>28/04/2017</td>
<td>Reviewed</td>
<td>Added comments</td>
<td>UBFCE</td>
</tr>
<tr>
<td>2.0</td>
<td>19/05/2017</td>
<td>Final</td>
<td>Feedback incorporated</td>
<td>DRAXIS</td>
</tr>
</tbody>
</table>
# Table of Contents

**Executive summary**.................................................................................................................. 7

1 Progress towards the APOLLO platform .................................................................................. 8
   1.1 Development methodology ............................................................................................... 8
   1.2 Co-creation meetings ...................................................................................................... 8
   1.3 User Experience .............................................................................................................. 8
   1.4 Integration ....................................................................................................................... 9
   1.5 Development of components ......................................................................................... 10
   1.6 Testing ............................................................................................................................. 11
   1.7 Communications ............................................................................................................. 11

2 APOLLO Architecture .............................................................................................................. 12

3 App functionality ...................................................................................................................... 17
   3.1.1 Notifications ............................................................................................................... 19
      3.1.1.1 System Notifications ........................................................................................... 19
      3.1.1.2 Emails ............................................................................................................... 21
      3.1.1.3 SMS .................................................................................................................. 22
   3.1.2 Exporting information ................................................................................................. 22
   3.1.3 Consultants / Farmer’s organisations ........................................................................ 22

4 Next Steps ............................................................................................................................... 23
   4.1 Features ........................................................................................................................... 23
   4.2 Performance .................................................................................................................... 23
   4.3 Quality of information provided by the services ........................................................... 23

5 Annexes .................................................................................................................................. 24

Annex I: Tillage Scheduling Service Pseudocode ...................................................................... 24
Table of Figures

Figure 1: Service results wireframe ................................................................. 9
Figure 2: Dashboard wireframe ...................................................................... 9
Figure 3: Dashboard snapshot of the early prototype ........................................ 10
Figure 4: Tillage service results from the early prototype ............................... 11
Figure 5: Data coming through the orchestrator and the sFTP connection ........ 13
Figure 6: APOLLO information flow diagram in the Spatial layer .................. 14
Figure 7: APOLLO business logic layer architecture ....................................... 15
Figure 8: Crop monitoring file configuration file for coloring the output file .... 16
Figure 9: Empty dashboard ........................................................................... 17
Figure 10: Create new field page .................................................................. 18
Figure 11: Dashboard page ............................................................................ 19
Figure 12: Service results page ..................................................................... 21

Table of Tables

Table 1 – Glossary ......................................................................................... 6
Table 2 – Component technologies ................................................................. 14
Table 3 – APOLLO business logic layer technologies .................................... 15

Glossary

<table>
<thead>
<tr>
<th>Abbreviation/Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>MVP</td>
<td>Minimum Viable Product</td>
</tr>
<tr>
<td>REST</td>
<td>REpresentational State Transfer</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UX</td>
<td>User Experience</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
</tbody>
</table>

Table 1 – Glossary
Executive summary

This document aims at describing the progress and accomplishments made in WP5 during the duration of the work package. Furthermore, it tries to describe some of the APOLLO platform functionality that was implemented. Finally, the deliverable offers an overview of the next steps that are envisioned right after the delivery of the platform.

Chapter 1 describes the progress made by the development team since the previous deliverable. Chapter 2 describes in details the architecture of the APOLLO platform with any changes that were adopted during the development period and tries to clarify the reasoning for the major change decisions. Chapter 3 provides some user interfaces of the platform, to give the reader a feeling of the environment, but also to demonstrate the main APOLLO functionality.

The last chapter contains a brief of the next steps that are planned for WP5 either towards developing functionality of secondary importance, which wasn't essential for the first version of the platform, but also to incorporate any feedback received from the pilot users or the latest co-working workshops.
1 Progress towards the APOLLO platform

The main mission of the platform is to combine high volumes of satellite data and images, process them and create new data that are used to deliver reliable agriculture services for farmers. The complexity of the attempt to provide those services, proved to be a challenge for the technical team, which was met by having many discussions only to establish a common terminology and understanding of the domain and to create a more concrete integration plan than the one initially planned in D5.1.

1.1 Development methodology

The team followed an agile methodology (SCRUM\(^1\)) to develop the platform and to have end users (mainly pilot users) involved in the process to generate the team with feedback. The team established a backlog of tasks, created a sprint schedule and started analyzing and working on the first sprints.

In order to verify the decisions taken, the team distributed the sprints in a way, in order to initially work to build a fast prototype with only the minimum essential needed functionality in order to verify the offered functionality through retrospective meetings or the co-working workshops. Further sprints and meetings brought the technical team closer to the end users and helped bridge the gap in understanding and expectations. Many additional minor or major changes or additions were planned to be added to the platform due to that feedback.

1.2 Co-creation meetings

During January and February, the WP5 team participated in the organization of the first co-creation meeting. A presentation containing all the information of what had been built by that time was created to be used during the meetings along with specific questions that needed to be addressed to the end users. This resulted in valuable feedback from all such meetings, like the type of information users would like from the weather service, how they would like to pay for such services and how they want the information to be presented. That feedback was also used at some of the iteration the team had with the UI/UX expert that was used in the project.

1.3 User Experience

It was a perception among technical partners that the platform should offer a streamlined experience to its users being friendly and clean not making it hard for them to discover the offered functionality. Therefore the team decided to hire a UI/UX\(^2\) expert who would help build an attractive and intuitive set of applications. That meant spending time with the expert to explain him the concept and help him produce something meaningful and beautiful. However, in that first attempt the results weren’t satisfactory and it proved that during the process, WP5 partners were very much striving and guiding the expert on details that should have been his expertise. It was soon decided to search for a new

---

\(^1\) [https://en.wikipedia.org/wiki/Scrum_(software_development)](https://en.wikipedia.org/wiki/Scrum_(software_development))

partner with specific experience in farming or EO applications. The team managed to find that professional and soon enough things kicked off following a more reliable structure:

- Initial discussions regarding APOLLO, its’ vision and sharing of the related documents with the expert and meetings to discuss and answer questions;
- The expert created wireframes to depict the main functionality of APOLLO that helped us a) understand that he was in line with the APOLLO concept and b) demonstrate to us the placement of crucial controls in the user interface and the information that should be presented to the end users;
- Those sketches were also presented to other users and the team had discussions with the expert to propose slight changes or provide clarifications;
- Based on the feedback the expert created the first version of the mockups and some screenshots of how the app will look were also produced.

Below are two examples of the sketch/wireframe work created by the second expert:

![Figure 1: Service results wireframe](image)
![Figure 2: Dashboard wireframe](image)

### 1.4 Integration

From the previous deliverable, the team established an integration plan to connect the APOLLO subsystems and to also connect to the various sources to download and process the data needed for the agricultural models. When the implementation phase started, discussions among the involved partners sparked the need to specify additional details that were important when trying to glue the components together. Although partners in the consortium had worked together in similar integration attempts in the past, the nature of APOLLO and the need to build a scalable production environment,
which integrates the various components under a unified umbrella-system, proved to be a more demanding problem than was originally thought and planned.

Some of the problems that are worth mentioning here are:

- Initially mock data was used to build and glue the various subsystems together. Due to that, importers and APIs were developed without having in mind problems like the amount of data that will be handled, the duration of time-heavy processes etc. When the real data came, it became evident that we needed to add a “pre” and “post” processing stage to assist the data process and make it compatible with the existing infrastructure. However there were cases that the team needed to go back and revise the integration plan as well as the developed endpoints.

- When partners from WP3 starting feeding data to WP4 or WP5, the team realized several changes per the initial plans. Data needed to be transformed to other formats (e.g. GRIB, TIFF, CSV) and each reported data in different coordinate systems which tried to be combined into something common.

- The first tests with the MVP revealed that the rasterized geographic data which were used in all parts of APOLLO (from retrieval to visualization), were not providing a very smooth visual representation to the end users, especially when they needed to use the zoom in feature of the map control. In order to offer a better experience to the end users, it was decided to convert the rasterized data into vectors through an interpolation algorithm to smooth out the produced images.

### 1.5 Development of components

All the essential APOLLO components for the first release of the platform were developed in time. The four services were developed as described from WP4 (with slight adjustments) and the rest of the platform modules for handling any other functionalities (e.g. for storing and retrieving data from/to the database, transformations, transactions, APIs etc.)

To better facilitate meetings with end users and the co-creation workshops, the team worked early in the project to build a prototype with minimum functionality upon which the various components were integrated, tested and demonstrated. Indicative mockups of that early prototype can be seen in Figure 3 and Figure 4.

**Figure 3: Dashboard snapshot of the early prototype**
1.6 Testing

During development of the components, the team developed the essential mechanisms to test and validate the behavior compared to the original plan as it was described in D5.1 and reassure that the platform will function securely and reliably. The testing procedure helped the team track and resolve issues both in code but also in the logic of the events in APOLLO.

1.7 Communications

The WP team mainly used conference calls to communicate and streamline. The calls were initially ad-hoc, but usually they took place every third week. Over the last few months the team focused on implementing the remaining functionality but also changes that occurred and were exclusively related to the backend of the system. The increased workload demanded a closer cooperation with the other technical partners, to ensure the smooth integration of the system components. Due to that the team increased the communication frequency and started having regular weekly calls to discuss progress and solve integration issues. This action helped clear out misunderstandings instantly and keep track of the progress as well as the delays.
2 APOLLO Architecture

The APOLLO architecture plan was followed and made more specific, as it was initially described in D5.1. Changes were introduced in several parts of the architecture as indicatively described below:

- **Integration with the soil moisture processing chain**
  During the implementation of the orchestrator, it became evident that the soil moisture service made more sense to be called on a per field request instead of a single call, that was initially thought to use to return all the parcels. In addition, the requests were enriched with per field soil-type variables which proved essential for the services to give more accurate soil moisture results. Those options for soil type were: Sand, Clay, Silt, and Organic Matter.

- **Meteo Tool integration**
  Initially the plan was for the Meteo Tool to offer an API (“Meteo receiver service”) that would be used by the APOLLO platform for communication and exchange of data. Instead the team preferred to establish an sFTP connection from Meteo tool to the APOLLO system and “push” there the appropriate forecasts in CSV format. This change was introduced in order for APOLLO to keep an open door to other services similar to the Meteo tool, to push their data onto its servers. Furthermore, the team preferred to keep a data acquisition approach like the soil moisture handling which was also pushed in the sFTP folder.

  In terms of actions sequence, once the Meteo tool finishes uploading forecast data to the sFTP folder, a “finish signal” is sent to the APOLLO orchestrator which then triggers the essential workflows and grab the data.

- **Importing Data into Rasdaman**
  The initial plan was to use the WSCT utility to import georeferenced data into Rasdaman Petascope. Unfortunately, this utility was not properly working when attempting to save over the network, so the team exposed a second endpoint (sFTP connection) to collect and import partner’s data into rasdaman locally.

---

3 Secure File Transfer Protocol, a network protocol used for secure file transfer

4 The WCS Transaction extension (WCS-T) defines a standard way of inserting, deleting and updating coverages via a set of web requests http://rasdaman.org/wiki/WCSTGuide
**Updates to the Orchestrator's structure and data workflow in the Spatial layer**

The team changed slightly some of the Orchestrator steps like session creation in mongo db, splitting the geojson with all the parcels into per parcel geojsons in order to support outgoing requests to the soil moisture processing chain. An indicative operational workflow for the Orchestrator is:

- Task 1) Create a session for the current day in mongodb
- Task 2) Download Parcels and meta data from the APOLLO API
- Task 3) Trigger the Soil moisture service (API) to request the Soil Moisture per parcel
- Task 4) Wait for finish/error signals from any of the subsystems (Soil moisture service, EO biophysical parameters and meteorological service, Meteo tool)
- Task 5) Upon “finish” receival from the soil moisture service: Migrate the TIFF files from sFTP to the model staging area
- Task 6) Upon “finish” receival from the EO biophysical parameters and the meteorological service: Communicate with Rasdaman and receive Leaf Area Index, Biomass, NDVI and Chlorophyll index per parcel (this data was imported tin Rasdaman in TIFF files)
- Task 7) Upon “finish” receival from the Meteo Tool: Migrate the CSV files from sFTP to the model staging area
- Task 8) Prepare the inputs for the advisory models (preprocessor) and output them in TIFF format. Actions include resampling and file transfer. Inputs to the preprocessor are the parcels and metadata from step 2 and data retrieved by Rasdaman.
- Task 9) Calculate the advisory models and output them in TIFF files. Input to the models is coming from the preprocessor (TIFF files) and the Meteo CSV files from task 5.
- Task 10) Transform the output from the models (postprocessor) so it can be fed a) back into Rasdaman b) to the Mapserver stack for visualization purposes. Actions include the change
to vectors, and the production of different formats (e.g. in shapefile). The output is provided in file TIFF format.

The orchestrator was developed in NodeJS.

Figure 6 presents data flow from the APOLLO main sources (WP3, WP4) fed into the advisory models and pushed to MapServer.

![Image of data flow diagram]

*Figure 6: APOLLO information flow diagram in the Spatial layer*

The technologies used in the modules in the diagram above are presented in the table below:

<table>
<thead>
<tr>
<th>Modules</th>
<th>Technology used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soil moisture service</td>
<td>Python and node.js</td>
</tr>
<tr>
<td>Meteo Tool</td>
<td>Operation workflow runs with a Bash script calling code prepared in Fortran to handle the preprocessor, actual model work and post processor</td>
</tr>
<tr>
<td>Biomass, NDVI, Leaf Area index, Chlorophyll Index</td>
<td>Python</td>
</tr>
<tr>
<td>Pre-Processor</td>
<td>Python</td>
</tr>
<tr>
<td>Processor (models):</td>
<td>All models were developed in Python</td>
</tr>
<tr>
<td>1. Tillage Scheduling</td>
<td></td>
</tr>
<tr>
<td>2. Irrigation Scheduling</td>
<td></td>
</tr>
<tr>
<td>3. Crop Growth Monitoring</td>
<td></td>
</tr>
<tr>
<td>4. Crop Yield Estimation</td>
<td></td>
</tr>
<tr>
<td>Post-Processor</td>
<td>Python</td>
</tr>
<tr>
<td>Orchestrateur</td>
<td>Node.js</td>
</tr>
<tr>
<td>Storage</td>
<td>File system and Rasdaman</td>
</tr>
</tbody>
</table>

*Table 2 – Component technologies*
Changes to the APOLLO business logic layer architecture

The APOLLO business logic layer architecture was slightly changed. Initially the plan was to use Mapserver with Mapscript. However Mapscript was causing a big load on Mapserver and made it very slow in responses. The team examines several other configuration options and in the end decided to remove Mapserver/Mapscript with Mapnik, an open source toolkit for rendering maps, which could handle the required work without overloading the server. An indicative operational workflow for the information flow in between the APOLLO applications and the database is:

Task 1) User creates a request in the APOLLO web or mobile app.

Task 2) The web server (apache/nginx) receives the request and responds back with the requested page.

Task 3) User’s browser parses the page which contains also calls to the API and Mapnik as well as for files (images, javascript code, etc).

Task 4) Any calls to the API are executed in the PHP pages to apply the APOLLO business logic and/or retrieve the respective information from the database (postgis).

Task 5) Any calls to Mapnik (WMS) are executed in C++ to visualize data over a map. The main source that feeds the Mapnik service are the files that were produced from running the models in the “data preparation workflow”.

![Figure 7: APOLLO business logic layer architecture](image)

The technologies used in the modules in the diagram above are presented in the table below:

<table>
<thead>
<tr>
<th>Modules</th>
<th>Technology used</th>
</tr>
</thead>
<tbody>
<tr>
<td>APOLLO API</td>
<td>PHP using Laravel API</td>
</tr>
<tr>
<td>Database storage</td>
<td>PostGIS</td>
</tr>
<tr>
<td>File Storage</td>
<td>A common file storage based on linux</td>
</tr>
<tr>
<td>Map server</td>
<td>Mapnik</td>
</tr>
<tr>
<td>Web server</td>
<td>Apache/Nginx</td>
</tr>
</tbody>
</table>

*Table 3 – APOLLO business logic layer technologies*
- **Advisory Models implementation**

The four models described in WP4 were implemented into code. Initially the plan was to use available fortran implementations which would be called from python or node.js (the orchestrator). However, performance tests revealed that this was a risky approach relying largely on system resources and the amount of data being processed.

Due to that, the models were developed from scratch, initially in pseudocode and finally in python. The pseudocode aimed to assist the communication between developers (WP5) and scientists (WP4).

An indicative pseudocode documentation of one of the advisory models can be found in the end of this document in Annex I: Tillage Scheduling Service Pseudocode

- **Advisory Models Configuration file**

For creating the output of each one of the four services described in WP4, the technical team designed a mechanism that parses configuration files (not code) and translates it into coloring rules. This method was used to paint appropriately the output of each model in georeferenced tiff files. The benefit behind this method was that it allowed for easier tests and modifications on the model’s behavior by changing the coloring rules without making code changes and therefore having to rebuild and deploy code. Below is an example of such a configuration file.

```
CropMonitoring:
  color_file : /opt/models/Config/color.txt
  CropChanges:
    range 1:
      condition : "(minCrop <= x) & (x < minCrop + e)"
      color : '100'
    range 2:
      condition : "(minCrop + e <= x) & (x < minCrop + (2*e))"
      color : '200'
    range 3:
      condition : "(minCrop + (2*e) <= x) & (x < minCrop + (3*e))"
      color : '300'
    range 4:
      condition : "( minCrop + (3*e) <= x) & (x < minCrop + (4*e) )"
      color : '400'
    range 5:
      condition : "(minCrop + (4*e) <= x) & (x < minCrop + (5*e))"
      color : '500'
    range 6:
      condition : "( minCrop + (5*e) <= x) & (x < minCrop + (6*e))"
      color : '600'
    range 7:
      condition : "(minCrop + (6*e) <= x) & (x < minCrop + (7*e) )"
      color : '700'
    range 8:
      condition : "(minCrop + (7*e) <= x) & (x < minCrop + (8*e))"
      color : '800'
    range 9:
      condition : "( minCrop + (8*e) <= x) & (x < minCrop + (9*e) )"
      color : '900'
    range 10:
      condition : "x >= (minCrop + (9*e))"
      color : '1000'
```

*Figure 8: Crop monitoring file configuration file for coloring the output file*
3 App functionality

This first version of the app is an MVP\(^5\) that will be used by the pilot users to validate the services that APOLLO offers and the main platform functionalities. We followed the MVP approach to create a product just as complete as needed, for the first users to give us feedback so that we are then able to add additional features and characteristics. Below is a description of the main functionality that was included in this first version which, along with the feedback that we will receive from the pilots, will be used to create the final version of the platform at the end of the project.

The users can register via the web and land on the dashboard which to begin with is empty.

![Empty dashboard](image)

*Figure 9: Empty dashboard*

In order for them to start using the system and be able to get results from the 4 services, they have to add their fields in the system. They need to supply the following information for each field:

- Field name
- Crop type
- Seed type
- Irrigation system
- Soil type
- Planting date
- Harvest date
- Field location

\(^5\) Minimum Viable Product: A product that contains just about these features that are necessary for early users in order to provide feedback for the product.
Figure 10: Create new field page

Other than the field location, all the other form fields are commonly used form fields that most users are familiar with like dropdowns, free text fields and date picker calendars. The users will be able to draw their fields on the map. The list of all the fields that the farmer entered can be found on the dashboard, so every time the users log in, they can see a list of their fields along with summary information about any updates coming from any of the four services for each of their fields.
3.1.1 Notifications

The users through the settings section of the website are able to choose how they want to receive notifications. Specifically, they are able to choose the type of notifications emails, sms, mobile push notifications. Should the users wish, they can add a second email address that will also receive notifications for their fields. A verification process will follow to avoid spamming through the platform. Analytically the type of notifications the system provides:

**3.1.1.1 System Notifications**

If new information becomes available for any of the fields, the farmer will be able to click on the service icon displayed next to the field (it will be obvious to them when new information becomes available) and will be redirected to a page where the users can view more information about the
service results. The detailed service results depend on the type of the service and are described below.

- Tillage service
  A seven-day color indication displaying the optimal day to till along with soil workability and soil moisture maps

- Irrigation service
  A seven-day color indication displaying the optimal day to irrigate along with soil moisture, irrigation decision, Crop ET and crop water needs maps

- Crop growth monitoring
  Maps bearing information about Biomass, LAI, Crop vigor (NDVI) and Chlorophyll index.

- Yield estimation
  A prediction of the yield to be produced in tones will be displayed as a result of this service along with yield estimation and crop residue maps. In case of wheat being the crop type, a crop quality map will also be displayed.

Along with the maps the users can also see useful statistics. A user will be able to see the yield distribution of a crop type of all the users in his/her region and on the graph the user will be able to see where they stand along with a message explaining that "the field performance is better than 57% of the users in your region".
Figure 12: Service results page

3.1.1.2 Emails

The system will send emails to the users when there are updates from the services displaying useful information for them. Specifically, there will be one email notification for all four services in order not to create lots of noise for the end users. This email will contain the following information:

- **Tillage service**
  A seven-day color indication displaying the optimal day to till

- **Irrigation service**
  A seven-day color indication displaying the optimal day to irrigate

- **Crop growth monitoring**
  An indication that there is new information available and the user has to log in to the app to see them
• Yield estimation
  A prediction of the yield to be produced in tones
If the users want more information they will be able to click on a link within the email that will redirect them on the app to see a complete list of information.

3.1.1.3 SMS
If the users have opted for sms notifications, they will receive an sms informing them that there is new information available for them and they can find out more by logging to the APOLLO platform.

3.1.2 Exporting information
The users are able to export maps in shape file and kml format from the system to be used for example by other software that the APOLLO users might be using, or to import them in navigation systems that could help them optimize their tillage route.

3.1.3 Consultants / Farmer’s organisations
Other than farmers, consultants and farmers organizations will be able to register to the APOLLO platform and get information similar to that of the farmers. Once a consultant registers they will be able to add a new client (farmer) to the system and then add new fields in the same way that a farmer adds his own fields. The consultants will have a summary view of all their clients’ fields and will be able to select a specific farmer to view his/her fields and check if there are any new notifications from any of the four services.
4 Next Steps

Our plan is to continue improving the platform both from a feature perspective, but also from a perspective of performance and quality of provided information. As mentioned above, this first version forms an MVP and we plan to build on top of it and improve it over the next couple of years.

4.1 Features

There are some features that we would like the final platform to have, but we consider them not to be critical for the pilot period. Hence, we decided to proceed to the implementation of those features while the pilot operations are taking place. Examples of these features are the Payment plan that we will work with the partners responsible for the dissemination and the exploitation, the creation of admin users, the forum, the option for the user to buy add-ons, the option for a consultant to compare data from two of his clients and the option for a user to share data with a consultant.

Furthermore, from the first round of co-creation meetings we received some valuable feedback which we plan to assess and decide which of these suggestions will be incorporated to the platform based on the importance and difficulty of the task. Given that more co-creation meetings will follow and the pilot period will begin soon we expect more feedback from the end users and more feature changes will come up.

4.2 Performance

After the first period of usage from the pilot users, we will be able to better assess the performance of the platform and make decisions on what should be changed (if anything), so that the platform produces and displays results in a timely manner. That might result in rewriting parts of the services or assign more hardware resources to specific parts of APOLLO.

4.3 Quality of information provided by the services

During the pilot operations, the quality of the services will be assessed and changes in the implementation of the services might arise. We will be in constant contact with the pilots in order to receive feedback and make any necessary changes.
5 Annexes

Annex I: Tillage Scheduling Service Pseudocode

This annex describes the steps needed to calculate the tillage scheduling in pseudocode format.

**Step1:** Parameters and Data

- Import Crop Parameters (Crop Type, Sowing Date)
- Import Soil Data (Sand, Silt, Clay, Organic Matter)
- Import Weather Data (Yesterday Observations, 7-day Forecasts)
- Import Satellite Data (NDVI)
- Import Soil Moisture (Satellite or Yesterday Initial Soil Moisture)

**Step2:** Calculate Soil Hydraulic Properties

```plaintext
OrgC = OrgMat/1.724
BulkD = 1.510-0.113*OrgC
ThetaS = 0.838-0.283*BulkD+0.0013*Clay
ThetaR = 0.015+0.005*Clay+0.014*OrgC
Alpha = exp(-2.486+0.025*Sand-0.351*OrgC-2.617*BulkD-0.023*Clay)
NVG = exp(0.053-0.009*Sand-0.013*Clay+0.00015*(Sand)**2)
```

**Step3:** Define Soil Type

```plaintext
if((Sand>=85).and.((Silt+1.5*Clay)<=15)) then
    SoilType="Sand"
else if((Sand>=70).and.(Sand<=91).and.(Silt+2*Clay)<=30)) then
    SoilType="Loamy Sand"
else
        SoilType="Sandy Loam"
        SoilType="Loam"
    else if((Silt<=80).and.(Clay<12)) then
        SoilType="Silt"
else if(((Silt>=50).and.(12<=Clay).and.(Clay<=27)).or.((80<=Silt).and.(Silt<=50).and.(Clay<12)) then
        SoilType="Silt Loam"
        SoilType="Sandy Clay Loam"
        SoilType="Clay Loam"
else if((27<=Clay).and.(Clay<=40).and.(Sand<20)) then
        SoilType="Silty Clay Loam"
else if((Sand>=45).and.(Clay<=35)) then
        SoilType="Sandy Clay"
else if((Clay>=40).and.(Silt>=40)) then
        SoilType="Silty Clay"
else if((Clay>=40).and.(Sand<45).and.(Silt<40)) then
```
Step4: Calculate Soil Water Parameters

\[
PWP = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 15000)^{NVG}}
\]

```python
select case (SoilType)
    case("Sand")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 100)^{NVG}}
    case("Loamy Sand")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 100)^{NVG}}
    case("Sandy Loam")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 100)^{NVG}}
    case("Loam")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 330)^{NVG}}
    case("Silt Loam")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 330)^{NVG}}
    case("Silt")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 330)^{NVG}}
    case("Sandy Clay Loam")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 330)^{NVG}}
    case("Clay Loam")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 500)^{NVG}}
    case("Silty Clay Loam")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 330)^{NVG}}
    case("Sandy Clay")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 500)^{NVG}}
    case("Silty Clay")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 500)^{NVG}}
    case("Clay")
        FC = \Theta_R + \frac{(\Theta_S - \Theta_R)}{1 + (\alpha \times 500)^{NVG}}
end select
```

Step5: Calculate Crop Coefficient

\[
K_{cb} = 1.5625 \times NDVI - 0.1

fc = 1.318 \times NDVI - 0.1877

Ke = 0.25 \times (1 - fc)

Kc = K_{cb} + Ke
\]

Step6: Initialize Model

```python
if (Satellite_Soil_Moisture = True) then
    Soil_Moisture(0) = Satellite_Soil_Moisture
else
    Soil_Moisture(0) = \frac{(Yesterday_Soil_Water(0) - Kc \times Observed_Evapotranspiration + Observed_Precipitation)}{30}
end if

Soil_Water = Soil_Moisture(0) \times 30

Crop_Evapotranspiration(0) = Forecast_Evapotranspiration(0) \times Kc
```
Step7: Calculate Soil Water Budget and Apply Tillage Rules

```
do i=1,6
    Crop_Evapotranspiration(i) = Forecast_Evapotranspiration(i) * Kc
    Soil_Water(i) = Soil_Water(i-1) - Crop_Evapotranspiration(i-1) + Precipitation(i-1)
    Gravimetric_Soil_Moisture(i) = (Soil_Water/30)/BulkD
end select
```

select case(SoilType)
    case("Sand")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Loamy Sand")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Sandy Loam")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Loam")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Silt Loam")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Silt")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Sandy Clay Loam")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Clay Loam")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Silty Clay Loam")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Sandy Clay")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Silty Clay")
        LTL = ...
        UTL = ...
        Wopt= ...
    case("Clay")
        LTL = ...
        UTL = ...
        Wopt= ...
```
if (LTL <= Gravimetric_Soil_Moisture(i) < LTL + 1/3*(Wopt-LTL)) then
    Tillage(i) = Orange
else if ( LTL + 1/3*(Wopt-LTL) <= Gravimetric_Soil_Moisture(i) < LTL + 2/3*(Wopt-LTL)) then
    Tillage(i) = Yellow
else if ( LTL + 2/3*(Wopt-LTL) <= Gravimetric_Soil_Moisture(i) < Wopt) then
    Tillage(i) = Green
else if (UTL >= Gravimetric_Soil_Moisture(i) > UTL - 1/3*(Wopt-UTL)) then
    Tillage(i) = Orange
else if ( UTL - 1/3*(UTL-Wopt) >= Gravimetric_Soil_Moisture(i) > UTL - 2/3*(UTL-Wopt)) then
    Tillage(i) = Yellow
else if (UTL - 2/3*(UTL-Wopt) >= Gravimetric_Soil_Moisture(i) > Wopt) then
    Tillage(i) = Green
end do

**Step8: Print Results**

```plaintext
do i = 0,6
    write(Tillage.tif) Tillage (i)
end do
```
This project is co-funded by the European Union

END OF DOCUMENT