Deliverable 8.1:
Associated Documentation of EO AFRICA WRM software

EO AFRICA
Water Management

Document Id.:	P22S1956-44-v2.0	10/10/2024
Approved by:	Cristoforo Abbattista	
Checked by:	Giulio Ceriola	
Client Reference:	ESA Contract No 4000139810/22/I-DT	

Template: pkq001-30-2.4

Document History

List of reviews			
Version	**Authors**	**Date**	**Note**
0.0	Planetek Hellas, Planetek Italia Teams	16/05/2024	1st issue
1.0	Planetek Hellas, Planetek Italia Teams	10/09/2024	Revisions to answer to ESA comments
2.0	Planetek Hellas	20/12/2024	2nd Issue

Table of Contents

List of figures

1. Introduction

This document is the Associated Documentation of the developed software and corresponds to the second release of the deliverable D8 according to ESA Contract No. 4000139810/22/I-DT and the Project Proposal P22S1956-02-v0. This document is annexed to the EO AFRICA software package (P22S1956-43-v1_D8.2), part of the deliverable D8.

1.1. Applicable documents

- EO Africa Water Management" proposal "P22S1956-02-v0"
- "EO Africa Water Management" Negotiation Points P22S1956-03-v0.1
- "EO Africa Water Management" Minutes of the Preparatory Meeting P22S1956-06-v0
- Contract with ESA 4000139810/22/I-DT
- P22S1956-15-v1.1_D15_EO_AFRICA_EXPLORERS_PMP: Project Management Plan

1.1. Acronyms

BT	Brightness Temperature
cal/val	Calibration and Validation
FQDN	Fully Qualified Domain Name
CNAME	Canonical Name
CPU	Central processing unit
CRS	Coordinate Reference System
CSW	Catalogue Service for the Web
CWSI	Crop Water Stress Index
DEA	Digital Earth Africa
DNS	Domain Name System
ECOSTRESS	ECOsystem Spaceborne Thermal Radiometer Experiment on Space Station
EO	Earth Observing or Earth Observation
EO AFRICA	African Framework for Research Innovation, Communities and Applications in Earth Observation
EODC	Earth Observations Data Cube
EPSG	European Petroleum Survey Group
ET_a	Actual evapotranspiration
GeoTIFF	Geographic Tagged Image File Format
GEMET	GEneral Multilingual Environmental Thesaurus
GIS	Geographic Information System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
ICT	Information and Communication Technologies
IDE	Integrated development environment
I/O	Input/Output
LP DAAC	Land Processes Distributed Active Archive Center
LST	Land Surface Temperature
LSTM	Land Surface Temperature Monitoring
NASA	National Aeronautics and Space Administration
NDVI	Normalized Difference Vegetation Index
NIR	Near Infrared
NoR	Network of Resources
OGC	Open Geospatial Consortium
PKH	Planetek Hellas
PKI	Planetek Italia
PRISMA	Hyperspectral Precursor of the Application Mission
SARE	remote Sensing Approach to estimate the Reference Evapotranspiration

SDG	Sustainable Development Goals
TIR	Thermal Infrared Radiation
TRL	Technology Readiness Level
UAV	Unmanned Aerial Vehicle
VIS/NIR	visible/near-infrared
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service

2. EO Africa Water Management Processor

2.1. Input Data Requirements

To ensure seamless processing and accurate analysis, the algorithm has specific input data requirements tailored to optimize performance and reliability. Firstly, input images must be provided in TIFF format, preferably structured as composites containing essential spectral bands relevant to the analysis. A preferred configuration involves a 5-band or 3-band image composite, incorporating the desired bands for comprehensive analysis tasks.

Maintaining uniformity in spatial resolution is crucial for consistent and reliable analyses. Therefore, all input images should ideally share the same spatial resolution, preferably set at 30-meter resolution. Consistency in resolution mitigates potential distortions and inaccuracies, ensuring the integrity of analysis outcomes. Uniformity in image dimensions is essential for seamless integration into the algorithm. Consistent widths and rows across all input images facilitate streamlined processing workflows and minimize the need for additional preprocessing steps. Tools such as QGIS or ArcGIS can be utilized to achieve uniformity in image dimensions, enhancing data compatibility and analysis efficiency. Furthermore, ensuring consistency in the coordinate reference system (CRS) is vital for accurate geospatial analysis. All input images should be in the same CRS, preferably set to EPSG 4326. While this alignment can be achieved using QGIS, the algorithm includes an initial section dedicated to CRS alignment to ensure uniformity.

Each optical data date should have an equivalent thermal data date to enable seamless pairing and analysis. Therefore, the algorithm requires a one-to-one mapping between optical and thermal data dates, ensuring coherence and accuracy in the analysis process. In the naming convention of each input file, the date must be included to facilitate temporal analysis. This enables the algorithm to extract the Julian day accurately, contributing to precise temporal analysis and interpretation.

Input images should capture morning passes to capture reflectance and soil radiation effectively. Additionally, images must be cloud-free or undergo cloud removal processing to ensure data integrity and reliability. a mask must be applied to the initial data to ensure consistency and reliability. While the algorithm already incorporates masking procedures, careful revision and adaptation may be necessary if additional sensor images are included, apart from the known datasets such as ECOSTRESS, Landsat, PRISMA, and Sentinel. This is to avoid no data or calculations with 0 or nan.

2.2. Running the Algorithm and User Guide

2.2.1. Instructions for Setting Up and Running the Project

1. Clone the repository or download the project folder “EOAfrica_Software_Package”

2. Install the Required Packages. Ensure you have Python installed. Open a terminal in the project folder and run: pip install -r requirements.txt

3. Run the Main Script. To execute the eoafricauser.ipynb script, you can use either the terminal or an Integrated Development Environment (IDE) such as Visual Studio Code.

- From Visual Studio Code :
  - Open eoafricauser.ipynb.
  - Run the script by selecting Run all from the menu bar
- From Jupyter Lab :
  - Open a Jupyter launcher window and choose a Python 3 Kernel Notebook.
  - As with any other Python script, first import the necessary libraries.
  - It is highly recommended to organize your files into two separate folders: one for inputs and one for outputs.

Before running the algorithm, ensure that the input data are Level 2 and preprocessed, as this is required for successful compilation.

2.2.2. PRISMA data

For PRISMA data the PRISMA Toolbox shall be used in order to convert the he5 to separate tiffs for the desired bands. The necessary input for this software is in the form of PRS_L2D_STD_XX_XX_X.he5. On the upper right side in the Band preview tab the user can reach the desired band(s) and save it as tiff. The output file will be in the form PRS_L2D_STD_ XX_XX_X _[wavelength of band].tiff

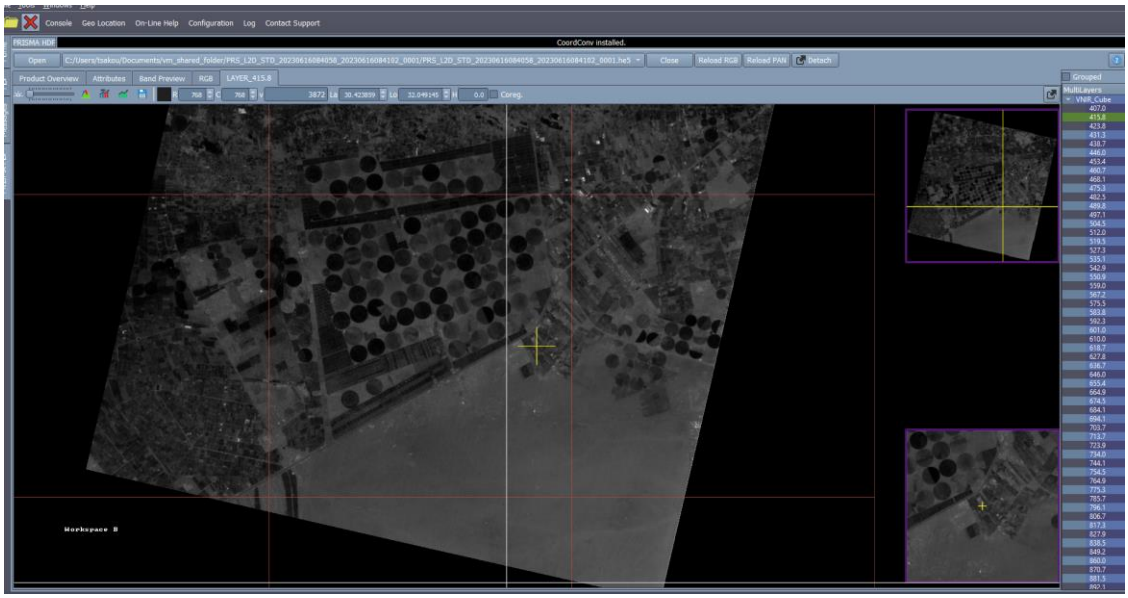

Figure 2-1 PRISMA Toolbox V1.0.0 environment for PRISMA h5 file manipulation and visualization

Having then the desired bands, in this case green, red and Nir the user shall first create an image composite as one tiff.

2.2.3. Sentinel data

Sentinel-2 Level-2 data originally come as SAFE format in the form of a folder named S2A_MSIL2A_XXXX_XX_XX_T36RUU/V_XXXX.SAFE. Inside this folder there are four subfolders and metadata. We are interested in the R10M subfolder which can be found in GRANULE > L2A_T36RU/V_XX_X > IMG_DATA Input data in Sentinel come as jp2 so they need to be converted into the tiff format. Then an image composite needs to be created out of the desired bands (Green, Red, NIR)

2.2.4. Landsat data

Landsat data acquisitions contain most of the information, as they can provide both thermal and optical data. Their initial form is LC08/09_L2SP_176039_XXX_XXX_02_T1.tar and shall be unzipped to extract the desired tiffs. The required bands are green, red, nir , LST and emissivity. Then a new image composite is created with these 5 bands as one tiff product. To facilitate the calculations both Level-1 and Level-2 Landsat data were used because Band 10 (Thermal band) offers as a product the BT in Level-1 and LST in Level-2.

2.2.5. ECOSTRESS data

ECOSTRESS data come as h5 files. Those are neither with proper crs nor geolocated. The preprocessing described here ECOSTRESS geolocation script needs to take place to convert ECOSTRESS swath data products, stored in Hierarchical Data Format version 5 (HDF5, .h5) into projected GeoTIFFs. Those preprocessing steps are distributed as a script by the Land Processes Distributed Active Archive Center (LP DAAC) performed under NASA contract NNG14HH33I. After running the script, we obtain the desired products as tiffs (LST and emissivity), and a new image composite can be created in tiff format.

Cropping and rescaling

In QGIS (or another desired GIS software) the user shall crop each tiff according to the desired shapefile. This new layer/product will be saved as a tiff, and that should be done either specifying a resolution of 30m (this is the preferred option), or by manipulating the columns and rows so that all inputs have identical width and height (see Figure 2-2).This step is very important in preparation to run the algorithm. This new layer/product will be saved as a tiff with the resolution preferably to 30m or by manipulating the columns and rows manually so all inputs have common width and height which is really important before running the algorithm. The same shall be done with the DEM file, so that its size matches the EO data, and users familiar with Python can do so with an upscaling algorithm and the rasterio library of Python. If the user is familiar with python this can be done with an upscaling algorithm using rasterio library of Python. This process will be the same for all input data regardless the sensor.

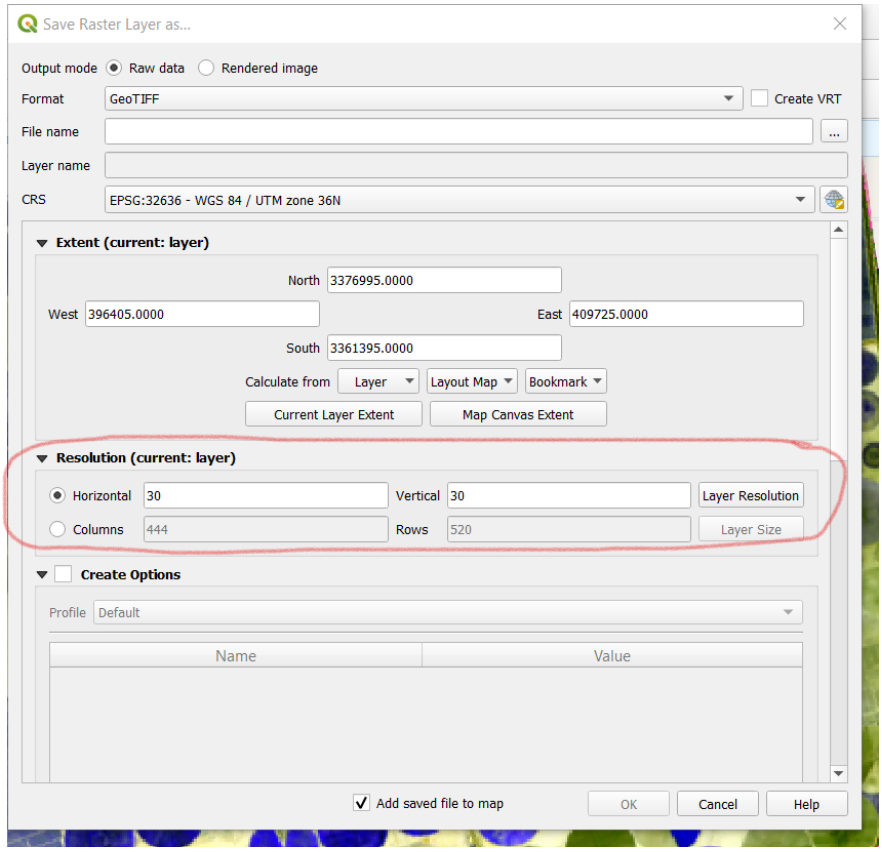

Figure 2-2: Indication of the final and important manual step before saving the final tiff in order to have the desired resolution or the desired number of column and rows

The previous version of the software was referring that that due to the abundance of optical data and the limited availability of thermal data for a given date, multiple data combinations may arise. For example, on the same or a close date, the user may have both PRISMA and Landsat data for optical bands (such as RED and NIR). This setup may necessitate the repeated use of Landsat thermal data, as it could be the only available thermal source, which may then be required to pair with each optical dataset. This is the reason why two different lists needed to be created, one for optical data without repeating files and another for thermal data, where repetition is permissible. This dual-list approach allows the code to loop over optical data and incorporate corresponding thermal data as needed.

This process was more time consuming and needed contribution by the user. For this version of the software the inputs are organized by sensor, where each Landsat date is stored in a designated Landsat folder, and a matching optical dataset is sought within a threshold of ±2 days. With this configuration, the code can iterate over each matching image pair, utilizing both the Landsat thermal image and the closest available optical image for each sensor.

2.2.6. Additional data and user's contribution

2.2.6.1. Real-time data

User's contribution is needed in order to run the code. The input paths need to be defined and the air temperature at 2m height needs to be added by user for the

calculation of CWSI. Air temperature data are not provided by the metadata of the EO data only LST and BT, so they have to be found online in various platforms (e.g climatereanalyzer) and to be added manually for the specific date.

2.2.6.2. In-situ data

Optionally, in case the user has the specific latitude and longitude of the field measurement, there is the possibility to compare with the EO values that have the same coordinates by finding the corresponding pixel value by using the pointing sampling tool Plugin in QGIS. By having the in situ data in CSV for, but especially latitude, longitude and Evapotranspiration the user can open the tool and follow the steps below:

1. In QGIS, go to Layer > Add Layer > Add Delimited Text Layer for the csv.
2. In QGIS, go to Layer > Add Layer > Add Raster layer for the raster of the algorithm.
3. For **Layer containing sampling points**, select the point layer corresponding to the current CSV
    - In the **Geometry Definition** section:

    Point coordinates:

    - Select the column for longitude (usually lon or x) as **X field**.
    - Select the column for latitude (usually lat or y) as **Y field**.
4. For **Layers with fields/bands to get values from**, select the raster file it corresponds to.
5. Run the tool to create a new point layer with the sampled values.
6. Save in an output csv by adding the input's csv columns and the newly created one which is the raster's match of the in situ points.

2.2.6.3. Error handling

When running the algorithm, several potential errors may occur:

- Error: Mismatched Dimensions
    - Explanation: Input data from different sensors with varying resolutions may lead to arrays with mismatched dimensions, hindering calculations.
    - Solution: Manually resize arrays if dimensional differences are minor (e.g., 100x100 vs. 105x100) or utilize the built-in "resize_arrays" function to adjust arrays to the minimum dimensions.
- Error: Latitude and Longitude Binary Files in Meters
    - Explanation: Latitude and longitude binary files may not be in degrees but in meters, potentially impacting result accuracy.

- Solution: Verify that the coordinate reference system (CRS) is set correctly to 4326 UTM-8 to ensure accurate representation of geographic coordinates.

- Error: Missing Thermal Data for Optical Input
  - Explanation: Lists containing files will have different lengths if thermal data does not correspond to optical input, disrupting data processing.
  - Solution: Select the nearest date with thermal data to ensure compatibility and continuation of calculations. Leverage the threshold function. Ig no thermal data exist the code will raise an error.

- Error: Low Evapotranspiration with ECOSTRESS Thermal Data
  - Explanation: Evapotranspiration estimates may appear unusually low when using ECOSTRESS thermal data, especially if measurements are taken at nighttime or early morning.
  - Solution: Ensure data collection occurs during appropriate times to obtain accurate results, as nighttime or early morning measurements may not accurately represent surface conditions.

- Error: Output Image Size Discrepancy
  - Explanation: Despite having matching columns and rows, output images may appear smaller than the original due to differences in pixel size.
  - Solution: Verify that the output resolution matches the input resolution by resampling the data accordingly using the provided code snippet. Resampling ensures consistency in image size and pixel resolution, aligning output images with the desired specifications and preventing discrepancies in visual representation.

```python
#This function resamples to the desired resolution , now is 30m (desired width and height based on prisma)
def resample_image(input_tif, output_tif, target_width, target_height):

    with rasterio.open(input_tif) as src:
        # Calculate target resolution
        target_resolution_x = src.transform[0] * src.width / target_width
        target_resolution_y = src.transform[4] * src.height / target_height

        resampled_data = []
        # Resample each band of the image
        for band_idx in range(src.count):
            band_data = src.read(band_idx + 1, out_shape=(target_height, target_width), resampling=Resampling.nearest)
            resampled_data.append(band_data)

        # Update the metadata for the output image
        out_meta = src.profile.copy()
        out_meta.update({
            "height": target_height,
            "width": target_width,
            "transform": rasterio.Affine(target_resolution_x, 0, src.bounds.left, 0, target_resolution_y, src.bounds.top),
        })

        # Write the resampled image to the output file
        with rasterio.open(output_tif, "w", **out_meta) as dest:
            for band_idx, band_data in enumerate(resampled_data):
                dest.write(band_data, band_idx + 1)
```

Figure 2-3: Function to verify the target resolution is the desired one and same as the input

3. Algorithm Validation using Digital Earth Africa Platform

The algorithm described has also been implemented and executed on the Digital Earth Africa (DEA) platform. The DEA platform provides access to a vast array of satellite data and tools for analysis across the African continent (already calculated indexes such as temperature, NDVI, crop health index etc.). Users can leverage DEA to analyze environmental changes, monitor agricultural patterns, and manage natural resources.

The algorithm was initially developed and run using Jupyter Lab, an interactive development environment, and then as Jupyter Notebooks within a Docker environment. Jupyter Lab and Jupyter Notebooks allow users to write and run code interactively, making data analysis and visualization straightforward. Docker provides a containerized environment that ensures consistency, ease of deployment, and reproducibility of the analysis. Additionally, it was configured to operate seamlessly within any Conda environment. This setup ensured versatility and accessibility across various computing environments.

In the implementation of the algorithm, Python3 serves as the primary programming language, chosen for its versatility, readability, and extensive ecosystem of libraries. To facilitate the execution and experimentation of the algorithm, Jupyter notebooks serve as the preferred environment. These interactive notebooks provide a user-friendly interface for running code, visualizing results, and documenting insights in a seamless manner. Moreover, the algorithm and associated notebooks are encapsulated within a Docker image, a lightweight containerization solution. This Docker image is constructed within a Linux virtual machine running Ubuntu 22.04 LTS. The underlying hardware configuration features an Intel 11th Gen Core i7 processor.

Furthermore, while the algorithm is primarily designed to run within the specified Docker environment, efforts have been made to ensure compatibility with alternative setups. The codebase is thoroughly tested and verified to run seamlessly within virtual machines using tools such as Visual Studio Code or Conda environments. However, it's important to note that these alternative environments do not include the Docker image, necessitating additional configuration and setup steps for deployment. By accommodating diverse development and execution environments, the algorithm maximizes accessibility and flexibility.

3.1.1. Digital Earth Africa (DE Africa)

The Digital Earth Africa Sandbox is a cloud-based computational platform that operates through a Jupyter Lab environment. It provides a limited, but free compute resource for technical users and data scientists to explore DE Africa data and products. It enables access to remote-sensing data and analysis tools for ad-hoc report generation and rapid development of new algorithms. This analysis environment is continuously improved to meet the needs of users. Digital Earth Africa utilizes the Open Data Cube This includes code examples based on USGS Landsat Collection 2, Level 2 and Copernicus Sentinel-2 Level 2A data that are available globally for use in Open Data Cube implementations. Digital Earth Africa Notebooks provide pip-installable Python modules containing useful tools for analyzing Open Data Cube data, including functions for loading and plotting satellite imagery, calculating band indices, analyzing spatial datasets, and machine learning. These tools can be accessed here: deafrica-sandbox-notebooks. The Digital Earth Africa Notebooks repository (deafrica-sandbox-notebooks) hosts Jupyter Notebooks, Python scripts and workflows for analyzing Digital Earth Africa (DE Africa) satellite

data and derived products. This documentation is designed to provide a guide to getting started with DE Africa, and to showcase the wide range of geospatial analyses that can be achieved using DE Africa data and open-source software including Open Data Cube and xarray.The environment shall look like the image below (Figure 3-1).

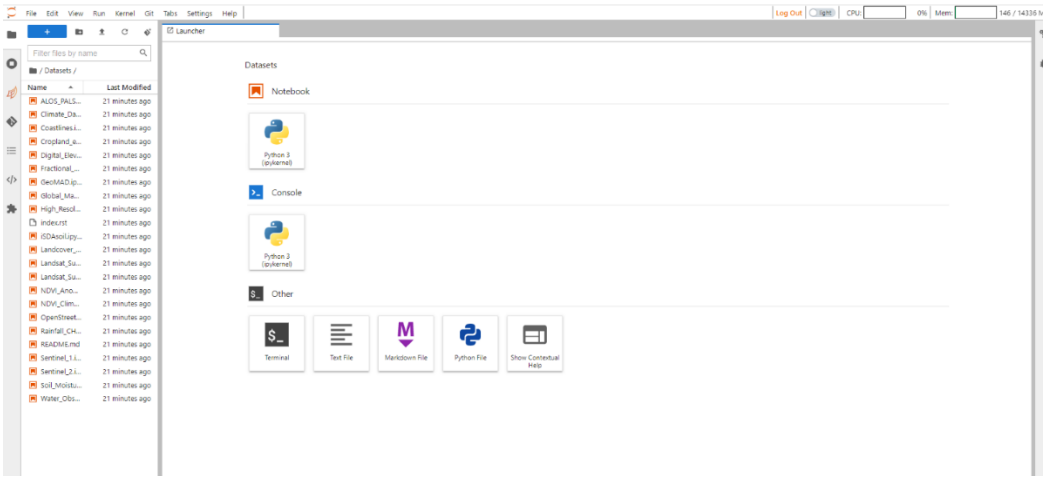

Figure 3-1 Jupyter Notebook environment where python scripts, notebooks or CLI can be created

3.1.2. Docker image of the platform

Utilizing the Digital Earth Africa platform, which hosts its notebooks on Git, a Docker image was locally built using the Digital Earth Africa Sandbox Docker build configuration. This allowed for the implementation of notebooks in a local Jupyter environment, leveraging the capabilities of the Digital Earth Africa tools without the need to operate directly within the platform. This local setup was advantageous because platform performance was limited due to low CPU and slow processing speeds. The docker is a software platform that allows to build, test, and deploy applications quickly. That is why we used it to build the DE Africa platform locally. The benefits of this choice are:

- It has an Independent Operating System and it is maintained well in time with future changes and advances of someone's working environment (Ubuntu, windows etc)
- There was already a Jupyter notebook created for docker purposes and could be found easily from online resources
- There is no constraints in resources such as GPU and CPU especially for big data / data dimensionality
- Even if the digitalearthafrica is down or offline the Algorithm still operates

3.2. Installation Instructions

Step 1:
A simple local environment that can be used to test the JupyterHub system can be started using Docker Compose with the command :
docker-compose up and browsing to *http://localhost:9988*, adding the token that is displayed on user's terminal after starting the system.

Step 2:
To run connected to a database in one of the DE Africa systems, the user will need to start a Kubernetes port forwarding process with a command: ***port-forward -n service deployment/pg-proxy 5432:5432***

Step 3:
It will be needed to set up a file in the root of this folder .docker.env with connection details in it. Use the .docker.env.example as a template for this file and then run the Docker Compose environment without a Postgres database, so use the command to start it :
docker-compose -f docker-compose.yml up

This will ignore the docker-compose.override.yml file, which provides a Postgres container.
Any files in the ./notebooks folder of the platform are mounted in the user's home folder. Running this with the appropriate commands creates the Docker image.
This Docker image with the docker yammer file (or communication) launch the Jupyter notebook of this project. Usually, this image is created once and there is no need for its recreation (except it is deleted or change the notebook). Once this is done, the user runs the "docker compose up" and the Jupyter lab notebook of the project is launched containing the notebooks of the DE Africa, the requirements and the libraries. The development of the algorithm begins. For the time being, the vagrant (or just the dockerfile and the docker yammel) are enough to run the platform successfully. In Figure 3.7 the Jupyter notebook environment, where the script is developed, is presented.

The libraries are automatically entered by the generated requirements.txt document, Any new library needed to run the algorithm will be placed inside this document and the Dockerfile will be updated and sees it.

Step 4:
For the deployment, the stack shall be initialized first with the command :
docker-compose exec jupyter datacube -v system init ,
where the part related to " datacube -v system init " initializes the Database Schema.

The datacube system init tool can create and populate the datacube database schema. In order to start the local Jupyter stack, the command:
docker-compose -f docker-compose.yml up ,
or as referred previously the docker compose up is needed.

Step 5:
To build a new docker image, the user inside a folder called docker deploy (or something related) shall run the building image command :
docker build -t NAME -f Dockerfile
and then run the newly created image docker run NAME python notebooks/SCRIPT.py.

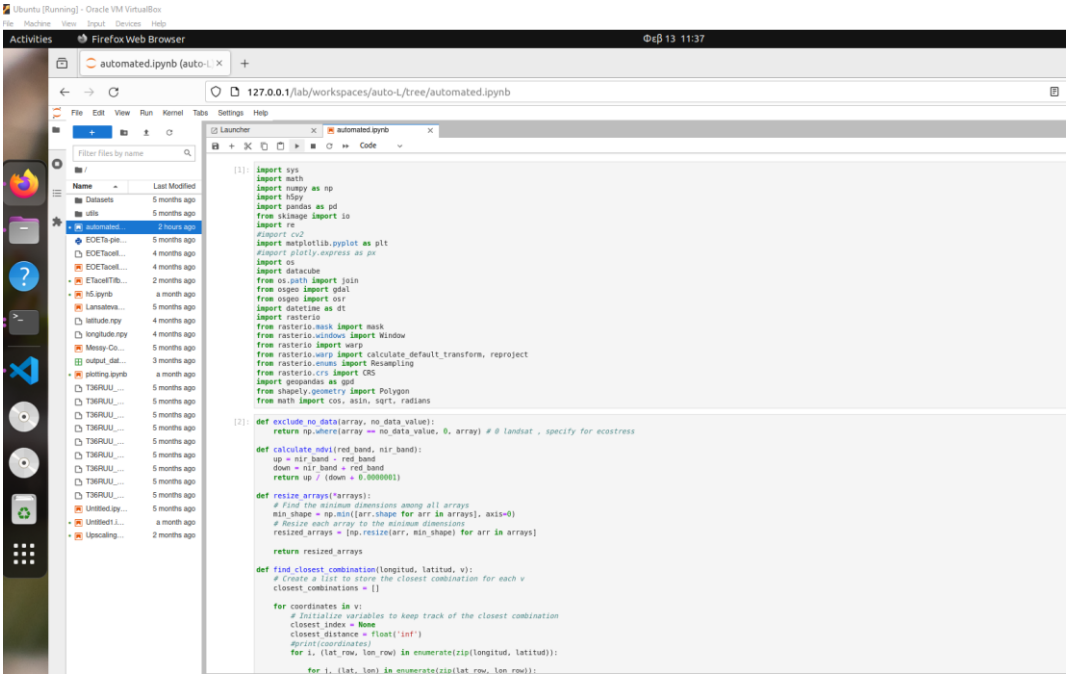

Figure 3-2: Environment and preview of the suggested algorithm for Evapotranspiration calculation developed in Linux VM.

4. EO Africa Water Management platform

The **EO Africa Water Management Platform** has been developed as a central hub to showcase the findings from the EO Africa Water Resource Management (WRM) project. This platform not only presents the project's results but also provides users with a wealth of insights and additional features designed to enhance their experience.

One of the key aspects of the platform is its user-friendly **dashboard viewers**, which allow users to easily navigate through data visualizations and analytics. These dashboards are tailored to present complex information in a clear and accessible way, empowering users to make informed decisions about water resource management.

By integrating various functionalities, the EO Africa Water Management Platform aims to support stakeholders—from policymakers to researchers—in their efforts to manage water resources sustainably.

4.1. Output Protype

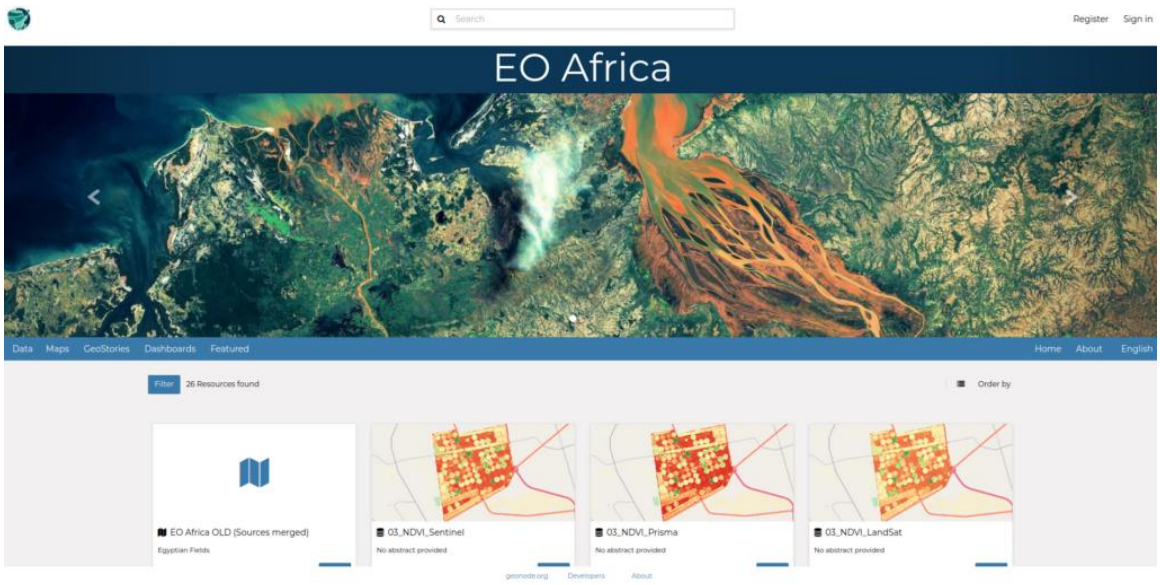

Figure 4-1: Landing page of the Prototype of the EO Africa Platform

4.1.1. Access

Access at [EO AFRICA WRM web platform]

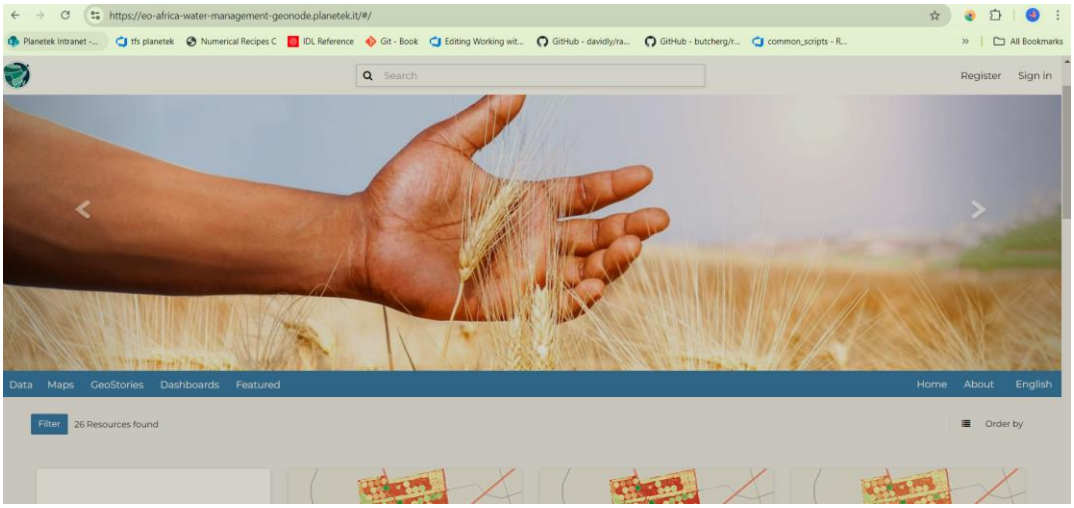

Figure 4-2: Log in/Registration page of the platform, showing overview of products

Logging in to the platform is not required, allowing anyone to access valuable information freely. The **output product maps** are open to the public, ensuring broad accessibility. At the top of the interface, users can easily navigate between **Maps** and **Dashboards**. When selecting the **Maps** option, a menu reveals various layers, showcasing three key product maps: **Crop Water Stress Index (CWSI)**, **Normalized Difference Vegetation Index (NDVI)**, and **Evapotranspiration**. Users have the flexibility to view all these maps simultaneously or focus on a specific product of interest. Each map is further categorized by the type of sensor used, providing options like **Landsat NDVI**, **Sentinel NDVI**, and **PRISMA NDVI**, making it easier for users to find the data they need.

In the main map area, a convenient **time series bar** allows users to select specific dates to view corresponding product maps. They can also opt for an automatic playback feature, which enables them to visualize changes over time effortlessly and enhancing user experience.

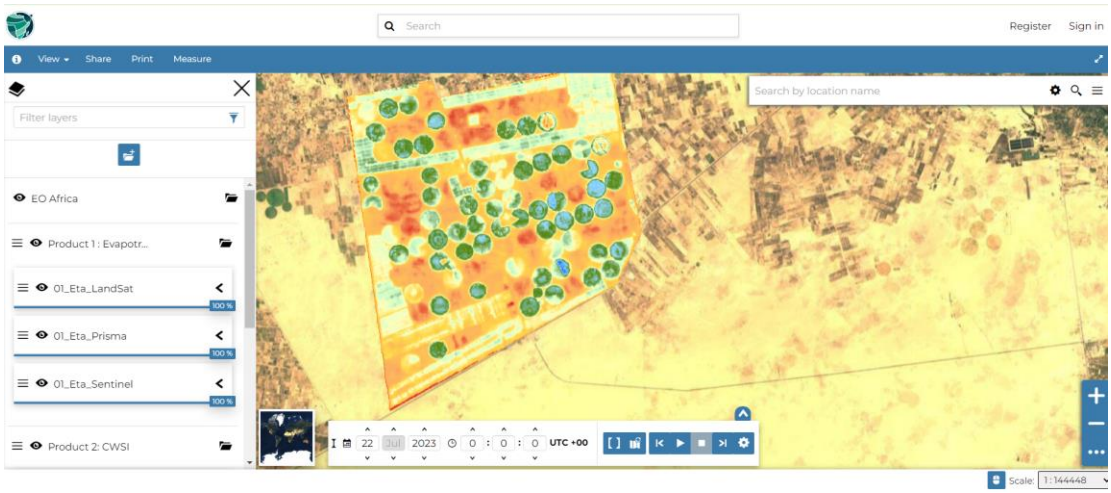

Figure 4-3: In the Map menu, Timeseries slide bar providing all Output products managed by date of acquisition

When users choose to view the **Dashboard**, they are greeted with a comprehensive catalog of results based on specific field measurements. One of the key features displayed is the **Evapotranspiration (ET)** values, which are calculated using advanced algorithms that process Earth Observation (EO) data from various sensors for each date. These values are presented in a clear line plot, where the **Y-axis** represents the Evapotranspiration values measured in millimeters per day (mm/day), while the **X-axis** denotes the corresponding field visit dates. This visual representation allows users to easily track and analyze changes in evapotranspiration over time.

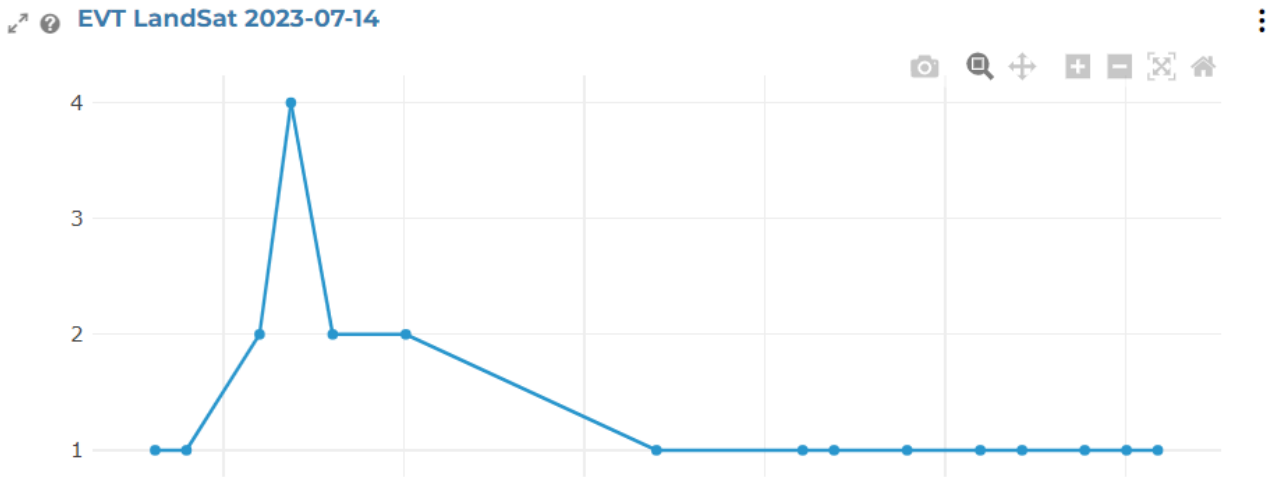

Figure 4-4: Dashboard providing line plot of EVT acquired with vector data for the 14th of July 2023

Figure 4-5: Dashboard providing line plot of EVT acquired with vector data for the 21st of July 2023

The interactive nature of the Dashboard enhances the user experience, empowering researchers, agricultural planners, and environmentalists to make informed decisions based on real-time data, based on the season and the month (stage of growth of the plant).

5. FAIR Principles

The main stack is based on GeoNode, an open-source platform that facilitates the management and sharing of geospatial data. It integrates with various geospatial tools and supports collaborative data management. The FAIR principles can be applied to the management and dissemination of geospatial data within GeoNode. Here's how GeoNode can adhere to each of the FAIR principles:

Findable:

a. (Meta)data are assigned a globally unique and persistent identifier:

   GeoNode Implementation: GeoNode can assign unique identifiers (such as UUIDs) to datasets. These identifiers ensure that each dataset can be uniquely referenced and retrieved.

b. Data are described with rich metadata:

   GeoNode Implementation: GeoNode supports the creation and management of comprehensive metadata for each dataset. Users can input detailed descriptions, keywords, and other relevant information.

c. Metadata clearly and explicitly include the identifier of the data they describe:

   GeoNode Implementation: In GeoNode, metadata records include the unique identifier of the associated dataset, ensuring clear linkage between metadata and data.

d. (Meta)data are registered or indexed in a searchable resource:

   GeoNode Implementation: GeoNode's internal search functionality indexes both data and metadata, allowing users to search for and discover datasets efficiently.

Accessible:

a. (Meta)data are retrievable by their identifier using a standardized communications protocol:

   GeoNode Implementation: GeoNode supports standard protocols such as HTTP, OGC WMS, WFS, and WCS for data retrieval. These protocols are widely used and standardized in the geospatial community.

   i. The protocol is open, free, and universally implementable:

   GeoNode Implementation: The protocols used by GeoNode, such as HTTP and OGC standards, are open and free, ensuring broad accessibility.

   ii. The protocol allows for an authentication and authorization procedure, where necessary:

   GeoNode Implementation: GeoNode provides authentication and authorization mechanisms, allowing controlled access to datasets where necessary, ensuring secure access management.

b. Metadata are accessible, even when the data are no longer available:

GeoNode Implementation: In GeoNode, metadata can remain accessible in the platform even if the associated datasets are removed, ensuring ongoing access to descriptive information.

Interoperable:

a. (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation:

   GeoNode Implementation: GeoNode supports standardized metadata schemas such as ISO 19115 and FGDC, which are widely accepted in the geospatial community.

b. (Meta)data use vocabularies that follow FAIR principles:

   GeoNode Implementation: GeoNode can be configured to use standardized vocabularies and ontologies that align with FAIR principles, such as the GEMET thesaurus for environmental terms.

c. (Meta)data include qualified references to other (meta)data:

   GeoNode Implementation: GeoNode allows for the inclusion of references and links within metadata records to related datasets and metadata, promoting data interlinking.

Reusable:

a. (Meta)data are richly described with a plurality of accurate and relevant attributes:

   GeoNode Implementation: GeoNode's metadata management system enables the inclusion of comprehensive and accurate metadata attributes, facilitating data reuse.

b. (Meta)data are released with a clear and accessible data usage license:

   GeoNode Implementation: Users can specify and include licenses for datasets in GeoNode, making the terms of use clear and accessible.

c. (Meta)data are associated with detailed provenance:

   GeoNode Implementation: GeoNode supports the documentation of data provenance within metadata, providing context and history of the data for future users.

d. (Meta)data meet domain-relevant community standards:

   GeoNode Implementation: GeoNode's adherence to widely accepted geospatial standards ensures that (meta)data conform to community standards, promoting interoperability and reuse.

By aligning with the FAIR principles, GeoNode enhances the discoverability, accessibility, interoperability, and reusability of geospatial data, facilitating better data sharing and collaboration within the geospatial community.