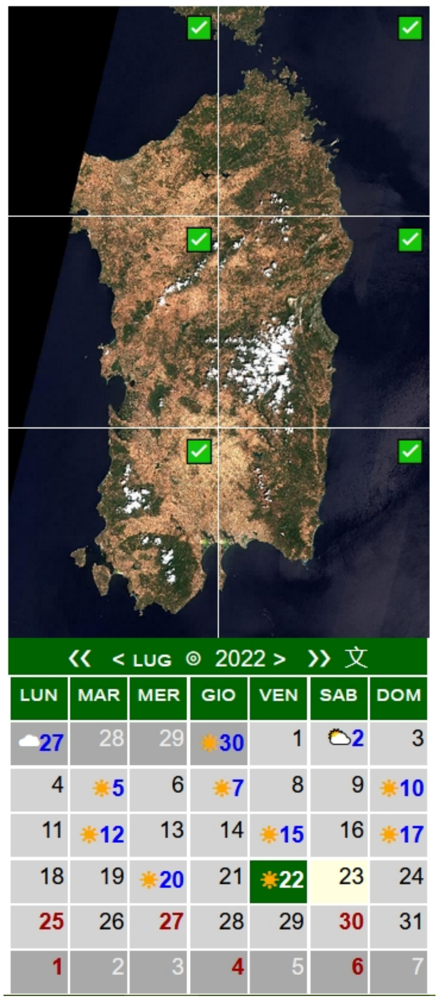
# *IsolaS2* web-app

## Rapid assessment of the utility of the Sentinel-2 MSI land-use data for regional actors

Gavin Brelstaff
CRS4 in Sardinia
2022



| LUN | MAR | MER | GIO | VEN | SAB | DOM |
|---|---|---|---|---|---|---|
| ☁27 | 28 | 29 | ☀30 | 1 | ⛅2 | 3 |
| 4 | ☀5 | 6 | ☀7 | 8 | 9 | ☀10 |
| 11 | ☀12 | 13 | 14 | ☀15 | 16 | ☀17 |
| 18 | 19 | ☀20 | 21 | ☀22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

https://crs4.github.io/IsolaS2/src

## 1. Introduction

*The activity described in this report continues a line of experimental research into the feasibility of client-side, browser-based solutions to certain fields such as electronic-patient records [1], mutli-lingual texts [2–5] and accessibility of satellite imagery over a given place and period. [6,7].*

In 2015 the European Space Agency (ESA) put its Sentinel-2 mission [8] into operation by launching the first of two satellites. Both now produce multi-spectral image (MSI) with a ground resolution as fine as 10m$^2$ – which significantly improves on that provided by USGS's Landsat 7 & 8 satellites [9,10] – while the range of spectral bands remains extensive – see Table 1. As such this data enhances support for land-use monitoring and analysis, within EU regions and across the world.

Although the Sentinel-2 data is made available at no monetary cost from ESA’s Copernicus Sci-Hub portal [11] accessing it remained mostly a domain of well-resourced institutions such as universities, research entities and corporate enterprises until two fairly recent advances:

> 1: the portal known as *EO Browser* [12] has brought a seamless Google-maps like navigation to the MSI data;
>
> 2: the search portal *CreoDIAS Finder* [13] has modernised the way of finding for data over a particular location and a given date-period: as such it is much more responsive compared to ESA's original Sci-Hub.

Nevertheless, it remains hard for less experienced users such as regional planners, citizen scientists, farmers, SMEs and start-ups to gain an intuitive picture of the pattern of data available to them for their region of interest – when they are faced with such global, generalist portals. The repeat patterns can be difficult to decipher since (a) a region can be imaged by more than one orbit, (b) acquisitions can fail and (c) even when successful their images can be obscured by clouds. So here we describe a third advance the *IsolaS2* web-app that displays for a chosen region a month-view of the available data-pattern. It is intended to bring Sentinel-2 access within reach of a wider population, whose concern is typically local rather than global and who may be equipped with just a smartphone, tablet or a basic PC.

| Band-id | Resolutions | | | Type | Peak | Bandwidth |
|---|---|---|---|---|---|---|
| B01 | | | 60m | Aerosol | 443 nm | 36 nm |
| B02 | 10m | 20m | 60m | Blue | 490 nm | 98 nm |
| B03 | 10m | 20m | 60m | Green | 560 nm | 45 nm |
| B04 | 10m | 20m | 60m | Red | 665 nm | 38 nm |
| B05 | | 20m | 60m | Visible IR | 705 nm | 20 nm |
| B06 | | 20m | 60m | Visible IR | 740 nm | 18 nm |
| B07 | | 20m | 60m | Visible IR | 783 nm | 28 nm |
| B08 | 10m | | | Panchromatic | 842 nm | 140 nm |
| B8A | | 20m | 60m | Narrow NIR | 865 nm | 32 nm |
| B09 | | | 60m | Water vapour | 945 nm | 26 nm |
| B10 | | | 60m | SWIR Cirrus | 1374 nm | 30 nm |
| B11 | | 20m | 60m | SWIR | 1610 nm | 142 nm |
| B12 | | 20m | 60m | SWIR | 2190 nm | 240 nm |

*Table 1. Sentinel 2 Multispectral bands and their approximate properties*

## 2. Motivation

CRS4 presents here a web-app named *IsolaS2* – intended as an entry point for quickly assessing the convenience of using Sentinel-2 MSI land-use data across a particular region or island. MSI stands for Multi-Spectral Imagery since there are 12 distinct spectral bands available – of which we focus on those most useful for monitoring vegetation across small-to-medium tracts of land.

As such this application intersects several key themes at CRS4: Space technologies, Environmental monitoring, and Agri-Tech – and taps our original core competence: i.e. that of integrating state-of-the-art internet technologies towards the perceived Regional needs, in a way extensible to solutions elsewhere.

Our central motivation is to lower the architectonic barriers experienced by unpractised users at the contemporary web-based Sentinel-2 data providers, and so boost the uptake of such data: In particular by explicitly removing:

1. the need to register, login, respect timeouts, and endure throttled downloads – at ESA's *SciHub* site*;*
2. the need to pay service subscription to download data – at *CreoDIAS* and *EO Browser;*
3. the need to adopt a cloud computing paradigm in order to easily access or download the data from the *Google Storage* Sentinel-2 repository [14].

This web-app has become feasible due to several recent state-of-the-art innovations in the delivery of images and data over the web – as will be detailed later – these include:

1. the advent of the free *CreoDIAS* Finder REST service to find available Sentinel data;
2. the ability to source thumbnail images from diverse online resources and swap them in when one or other goes down;
3. the ability of browsers to mix composite images from the pixels in pairs of thumbnails;
4. the ability to deep-link to given date/geo-location combinations at the *EO Browser* site;
5. the free ongoing Sentinel-2 repository at *GoogleStorage*;
6. the ability to implement a free CORS proxy using node.js on *Cloudflare Workers*.

Conceived first for the island of Sardinia, our web-app is configurable to run at other geo-locations – i.e. those representable as a few contiguous 100x100 km UTC tiles – e.g. Cyprus, Corsica the Hebrides, Gotland, Zanzibar, Yorkshire, etc. as shown in the Gallery Section. Indeed once the user navigates to the page configured for their geo-location all they need do is point, click, observe and download MSI data, where ever they deem it appropriate. Example of configurations are given on GitHub in the src directory of the project https://github.com/crs4/IsolaS2. And follow the implementation details in subsection 7.1 below.

The web-app is intended to be intuitive to use and its purpose easily discoverable – nevertheless the mini user guide below details its uses and helps define some terms used in this report.

## 3. IsolaS2 – illustrated for the Island of Sardinia

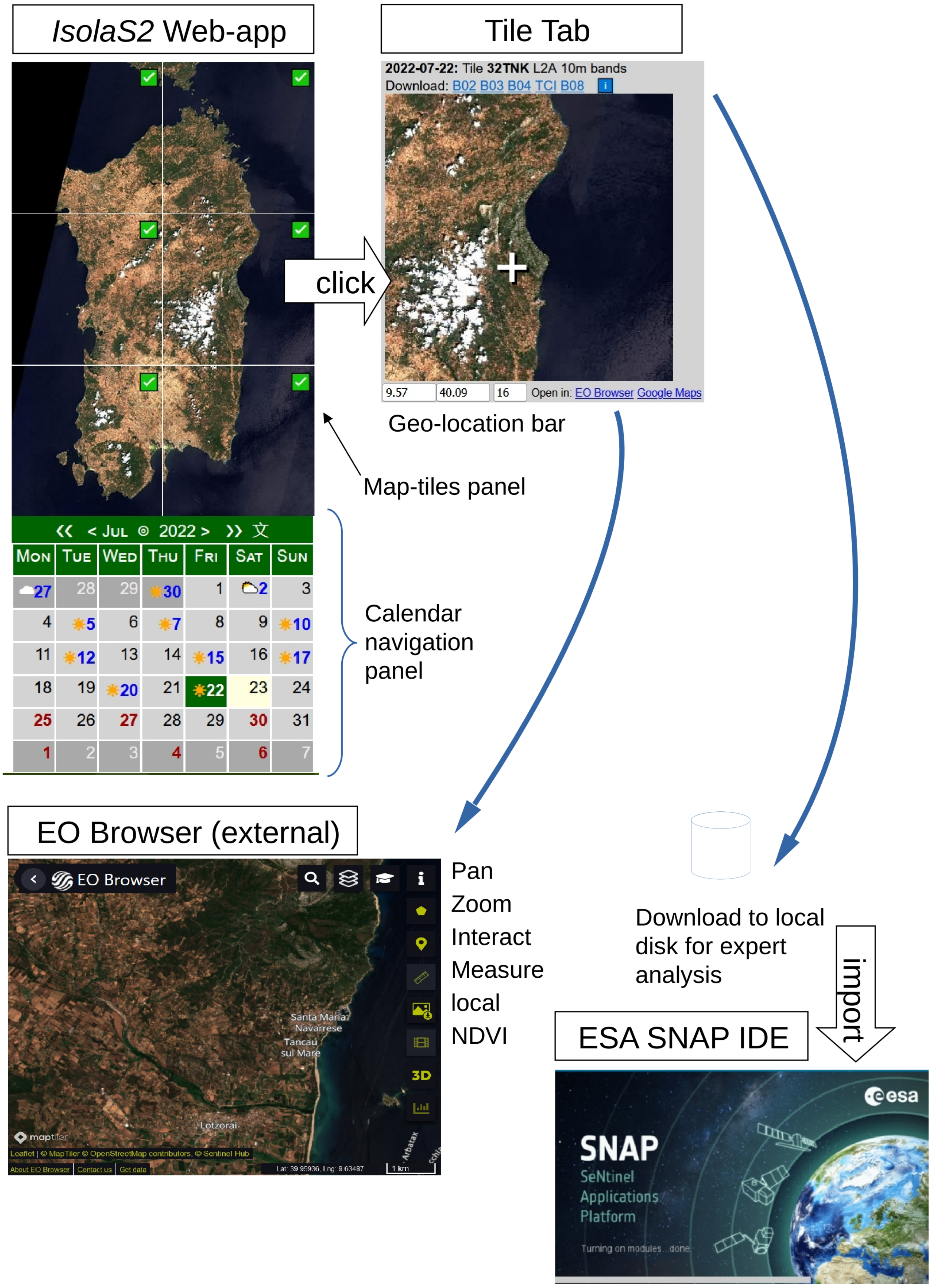

*IsolaS2* Web-app
Tile Tab
2022-07-22: Tile 32TNK L2A 10m bands
Download: B02 B03 B04 TCI B08
click
9.57 | 40.09 | 16 | Open in: EO Browser Google Maps
Geo-location bar
Map-tiles panel
<< < Jul ⊚ 2022 > >> 文
Mon Tue Wed Thu Fri Sat Sun
Calendar navigation panel
EO Browser (external)
Pan
Zoom
Interact
Measure
local
NDVI
Download to local disk for expert analysis
import
ESA SNAP IDE
SNAP
SeNtinel Applications Platform
Turning on modules...done
esa

***Fig. 1.*** *The Web-app with the Tile-Tab and further destinations (EO Browser & SNAP)*

## 4. Mini User Guide

Fig. 1 illustrates the use of the web-app. On the left the **Web-app** shows how the page first appears to the user – divided into an upper and a lower panel.

**The lower panel** acts as a calendar for navigating between acquisition dates, where:

- The current date is high-lit in yellow.
- The green highlight indicates the date of the acquisition being shown in the upper panel.
- Red numerals indicate future scheduled acquisition dates.
- Blue numerals indicate dates of existing acquisitions – each bearing a symbol.
- Each symbol (☁⛅☀) indicates the likelihood of obscuring cloud cover on that date.
- Clicking on a day marked by a symbol updates the upper panel to display its images.
- The top two rows of the calendar panel indicate the month, year, and the days of the week.
- Clicking on the 文 toggles the language of the month and day names – e.g. to Italian.
- Clicking on the < or > moves back or forward by a month.
- Clicking on the << or >> moves back or forward by a year.
- Clicking on the circle symbol returns to today’s month.

**The upper panel** is a grid showing contiguous UTC tiles where each tile is initially filled in with it geographically mapped image from the latest Sentinel-2 acquisition. Tiles are marked as follows:

- ● a yellow circle (not shown) indicates a tile for which the “raw” level 1C product has been successfully acquired by ESA.
- ✓ A green tick box replaces the yellow circle – typically after six hours – once ESA has processed the acquisition and published it as level 2A product suitable for land-use analysis. Clicking on a green-tick tile opens a new browser-tab labelled the **Tile Tab** in Fig 1
- ⧅ A ‘break’ symbol replaces those above in the rare case of a split tile – explained later. Its tile also becomes clickable when the level 2A product is available.
- ○ A grey circle indicates a tile left blank. This occurs when the tile either lies outside of the satellites swath, or outside of our specified region of interest and has not been requested, or when the data is not yet published.

**The Tile Tab** allows users to effectively zoom-in further and download MSI data to disk.

- A click on the tile-image established a point of interest (white cross in Fig 1.) and fills the geo-location bar with the corresponding longitude and latitude coordinates.
- A click on the link marked **EO Browser** opens another tab containing that external service centred on the selected longitude latitude coordinate – with an adjustable zoom-factor (16).
- The links marked B02, B03, B04, TCI or B08 when clicked initiate the download of the named 10m MSI band images for the given tile.
- Downloads are from Google Storage's Sentinel-2 repository and are typically rapid once they become available about a day following acquisition.
- Users may then visualize and or analyse the downloaded MSI data locally – e.g. on a workstation using ESA's SNAP software [15], a GIS or by using Gdal based scripts [16] or even on a PC using IrfanView [17].

## 5. Must-have requirements

Beyond the *no-barriers* rationale indicated above, we set the following ‘must-have’ requirements for the web-app to be considered successful. These aim to make its running costs negligible. The requirements include:

**R1** The web-app must require minimal maintenance – i.e. have no traditional back-end, middleware or database. That is: Zero stack, not Full-stack!

**R2** The web-app must, instead, source all the data it consumes (tile-images, data-searches and product metadata) purely from live online services – and directly within the browser.

**R3** The web-app must play fair and so not hammer those live online services – by tactful deployment of the browser’s cache and its local storage API.

**R4** The web-app must fit to the user’s screen: be it on a mobile, a tablet, laptop or desktop device.

**R5** The web-app must be configurable to other geo-locations of similar size and shape, and also allow the calendar’s natural language locale to be localised.

**R6** The web-app must require the installation of no software beyond that of a modern web browser: i.e. need no plug-ins, no browser extensions and no native applications.

**R7** The web-app must gracefully handle edge cases. In particular, where ESA publishes two land-use products for a given tile, rather than just one – or when one or more of the thumbnail-image servers goes offline for a period.

These requirements effectively fix our development upon the three basic **web technologies**:

**HTML –** to specify the layout of the calendar panel and to customise the grid of tiles in order to map the region of interest.

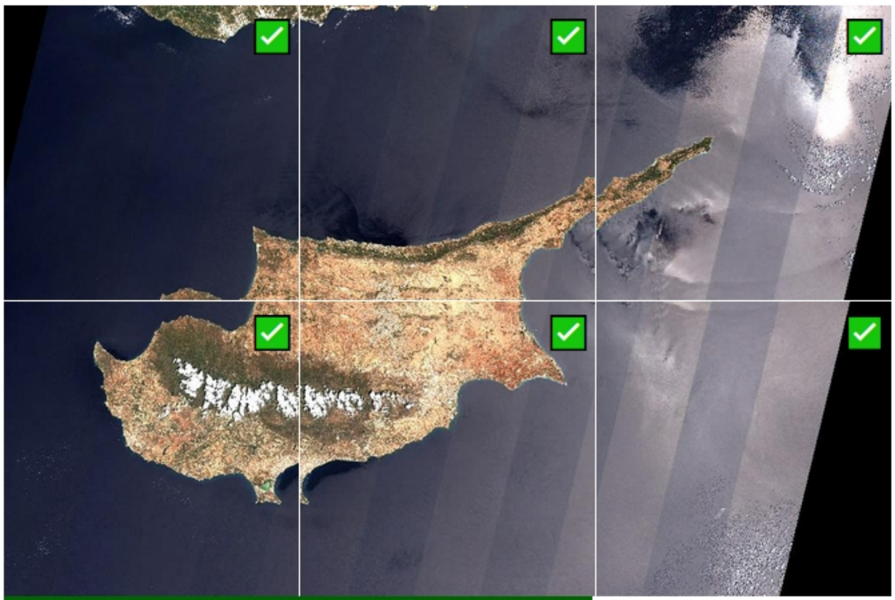
**CSS –** to style the page content and to be responsive to the user’s screen form-factor.

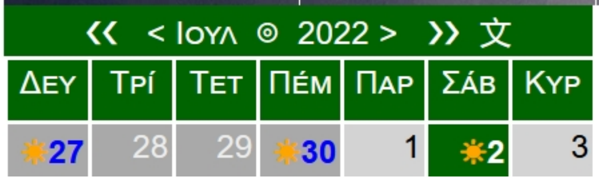
**JS –** i.e. JavaScript, to fetch (as JSON, XML) live data and marshal the necessary dynamic content and to implement interactive actions.

The result thus becomes a static web-site in the form of a directory of HTML, CSS and JS files. Consequently the web-app also runs locally from disk (without server) – a useful feature during development or configuration. The source directory is published/released by uploading it to any hosting system that respects directory structures and relative addressing – e.g. Firebase [18] or GitHub Pages [19] (but not WordPress nor OneDrive, nor Google Drive nor Dropbox, etc.) .

## 6. Gallery of Islands and Regions

*Cyprus*

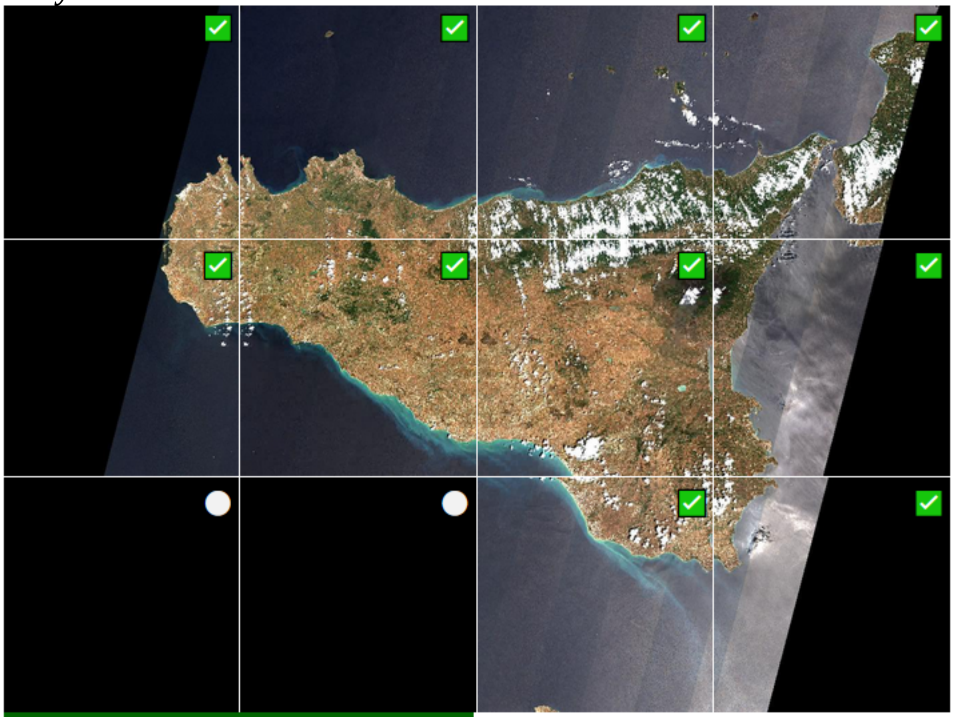


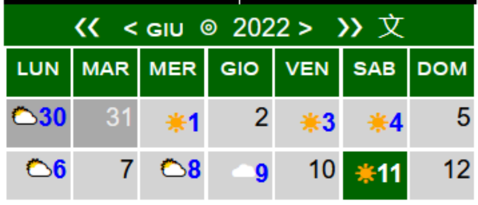


‹‹ < Ιουλ ⊚ 2022 > ›› 文

| Δευ | Τρί | Τετ | Πέμ | Παρ | Σάβ | Κυρ |
|---|---|---|---|---|---|---|
| ☀27 | 28 | 29 | ☀30 | 1 | ☀2 | 3 |

*Sicily / Sicilia*

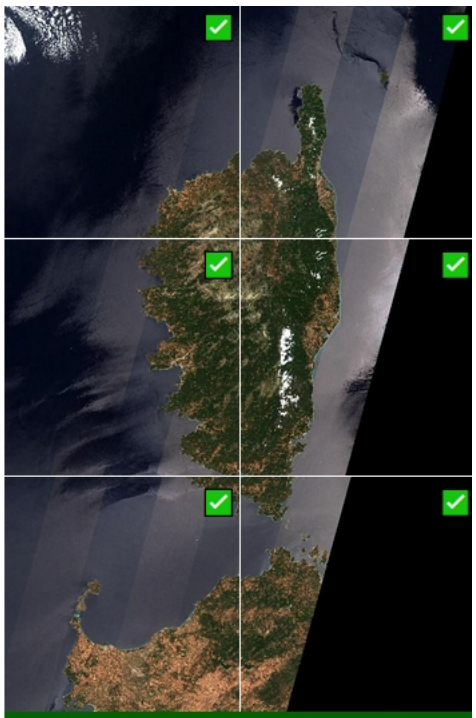




‹‹ < GIU ⊚ 2022 > ›› 文

| LUN | MAR | MER | GIO | VEN | SAB | DOM |
|---|---|---|---|---|---|---|
| ⛅30 | 31 | ☀1 | 2 | ☀3 | ☀4 | 5 |
| ⛅6 | 7 | ⛅8 | ☁9 | 10 | ☀11 | 12 |

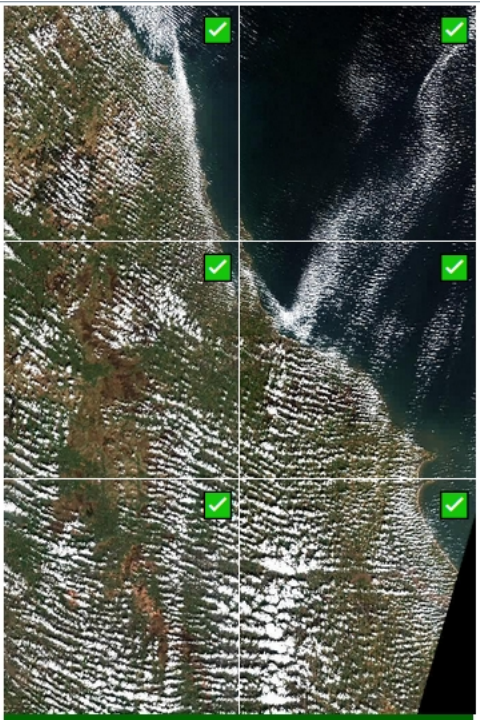
***Corsica***



❮❮ < JUIL. ⊚ 2022 > ❯❯ 文

| LUN. | MAR. | MER. | JEU. | VEN. | SAM. | DIM. |
|---|---|---|---|---|---|---|
| ⛅27 | 28 | 29 | ☀30 | 1 | ⛅2 | 3 |
| 4 | ☀5 | 6 | ⛅7 | 8 | 9 | ☀10 |
| 11 | ☀12 | 13 | 14 | ☀15 | 16 | ☀17 |

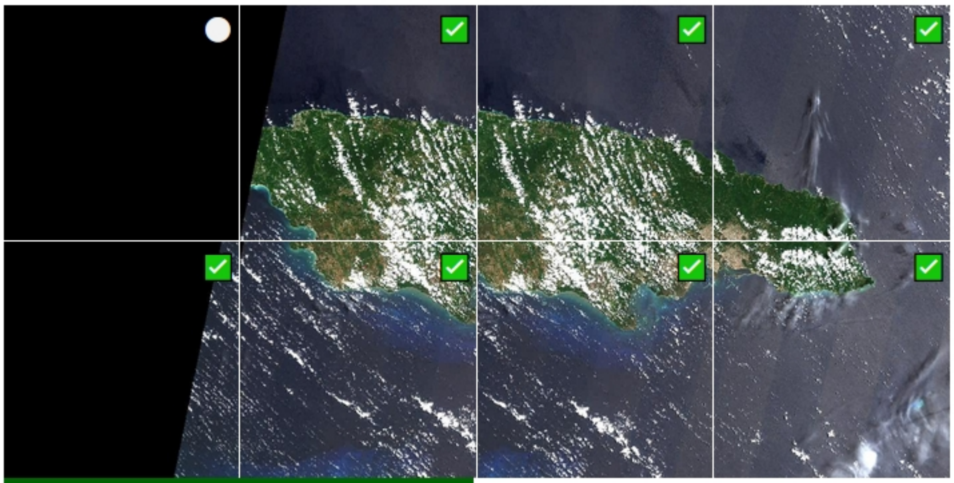
***Yorkshire UK***



❮❮ < MAR ⊚ 2022 > ❯❯ 文

| MON | TUE | WED | THU | FRI | SAT | SUN |
|---|---|---|---|---|---|---|
| ☁28 | 1 | ☁2 | 3 | 4 | ⛅5 | 6 |
| ⛅7 | 8 | 9 | ⛅10 | 11 | ⛅12 | 13 |
| 14 | ☁15 | 16 | ☀17 | 18 | 19 | ☀20 |

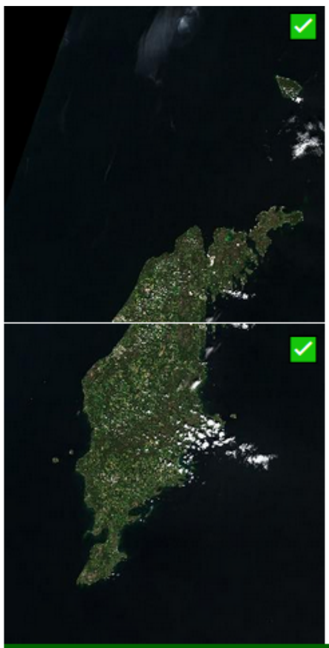
***Jamaica***



❮❮ < JUN ⊚ 2022 > ❯❯ 文

| MON | TUE | WED | THU | FRI | SAT | SUN |
|---|---|---|---|---|---|---|
| 30 | 31 | ☁1 | 2 | ⛅3 | 4 | 5 |

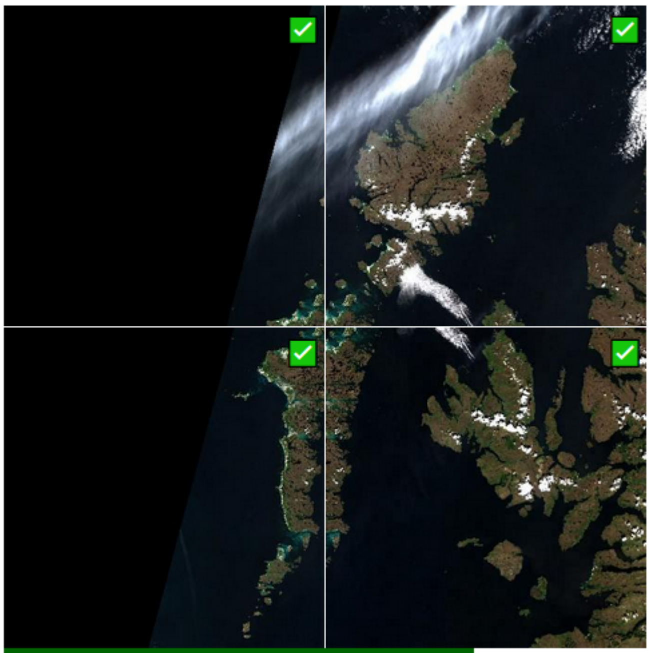
***Gotland Sweden***



❮❮ < JUNI ◎ 2022 > ❯❯ 文

| MÅN | TIS | ONS | TORS | FRE | LÖR | SÖN |
|---|---|---|---|---|---|---|
| ☁30 | 31 | 1 | 2 | 3 | ☀4 | 5 |

***Hebrides Scotland***



❮❮ < JUN ◎ 2022 > ❯❯ 文

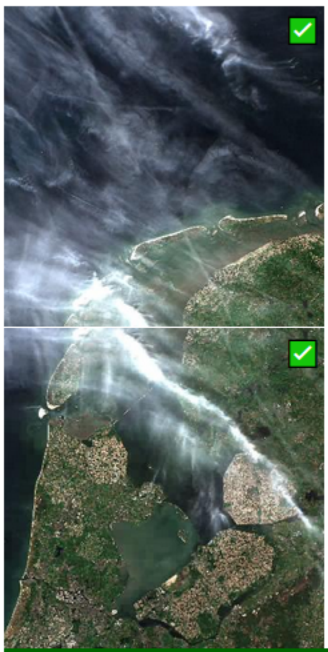
| MON | TUE | WED | THU | FRI | SAT | SUN |
|---|---|---|---|---|---|---|
| ⛅30 | 31 | 1 | ⛅2 | 3 | ⛅4 | 5 |

***Darwin Australia***

❮❮ < JUL ◎ 2022 > ❯❯ 文

| MON | TUE | WED | THU | FRI | SAT | SUN |
|---|---|---|---|---|---|---|
| 27 | ☁28 | 29 | ☁30 | 1 | 2 | ⛅3 |
| 4 | ☀5 | 6 | 7 | ⛅8 | 9 | ☀10 |
| 11 | 12 | ☁13 | 14 | ☀15 | 16 | 17 |
| ⛅18 | 19 | ☀20 | 21 | 22 | ☀23 | 24 |

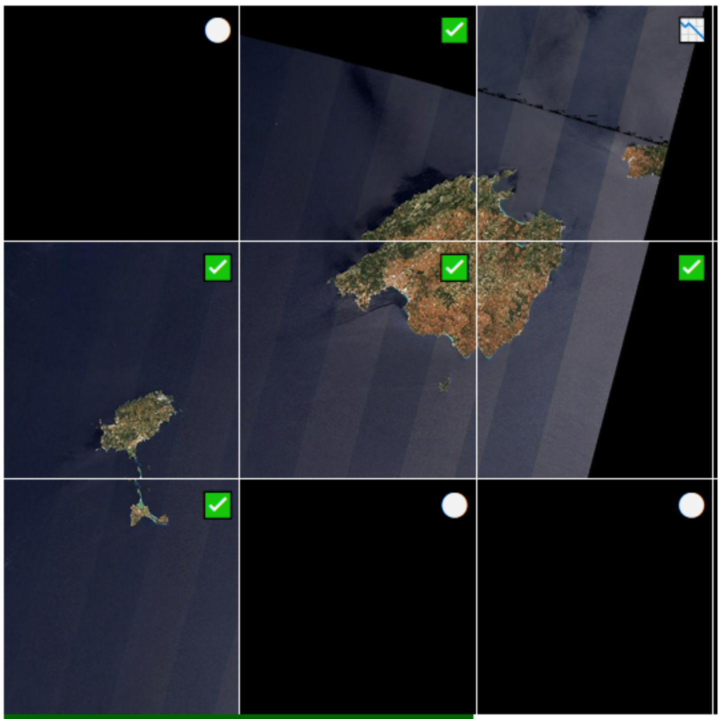
***Friesland***



❮❮ < MAY ◎ 2022 > ❯❯ 文

| MON | TUE | WED | THU | FRI | SAT | SUN |
|---|---|---|---|---|---|---|
| ⛅25 | 26 | 27 | 28 | 29 | ⛅30 | 1 |
| 2 | 3 | 4 | ☁5 | 6 | 7 | 8 |
| 9 | ☁10 | 11 | 12 | 13 | 14 | ⛅15 |

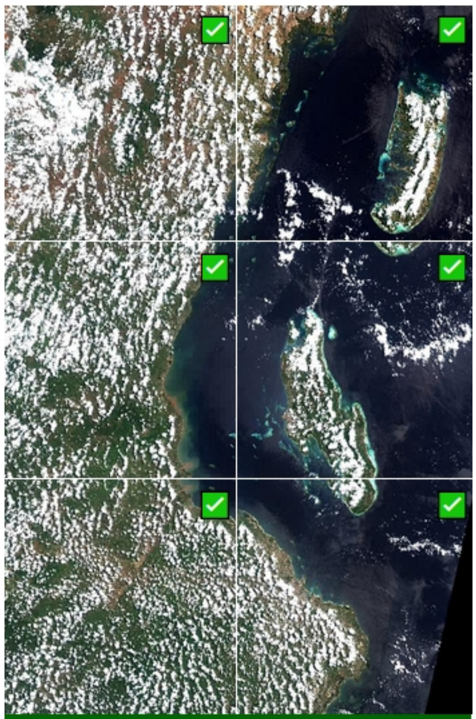
***Balearic Isles***



« < MAIG ⊚ 2022 > » 文

| DL. | DT. | DC. | DJ. | DV. | DS. | DG. |
|---|---|---|---|---|---|---|
| 25 | 26 | ☁27 | 28 | ⛅29 | 30 | 1 |
| ⛅2 | 3 | ☁4 | 5 | 6 | ⛅7 | 8 |
| ☀9 | 10 | 11 | ☀12 | 13 | ⛅14 | 15 |

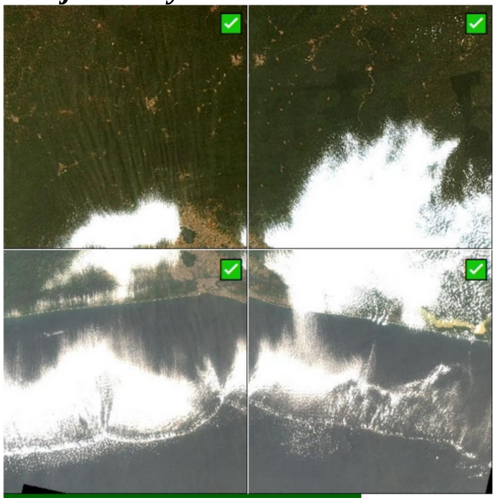
***Zanzibar Tanzania***



« < APR ⊚ 2022 > » 文

| MON | TUE | WED | THU | FRI | SAT | SUN |
|---|---|---|---|---|---|---|
| 28 | 29 | 30 | ⛅31 | 1 | 2 | 3 |
| 4 | ⛅5 | 6 | 7 | 8 | 9 | ⛅10 |
| 11 | 12 | 13 | 14 | ☁15 | 16 | 17 |

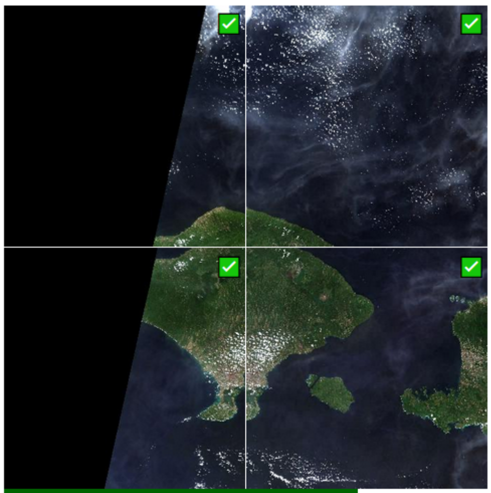
***Abidjan Ivory Coast***



« < JANV. ⊚ 2022 > » 文

| LUN. | MAR. | MER. | JEU. | VEN. | SAM. | DIM. |
|---|---|---|---|---|---|---|
| 27 | 28 | 29 | ☀30 | 31 | 1 | 2 |
| 3 | ⛅4 | 5 | 6 | 7 | 8 | ☁9 |
| 10 | 11 | 12 | 13 | ⛅14 | 15 | 16 |
| 17 | 18 | ☀19 | 20 | 21 | 22 | 23 |
| ☀24 | 25 | 26 | 27 | 28 | ⛅29 | 30 |
| 31 | 1 | 2 | ⛅3 | 4 | 5 | 6 |

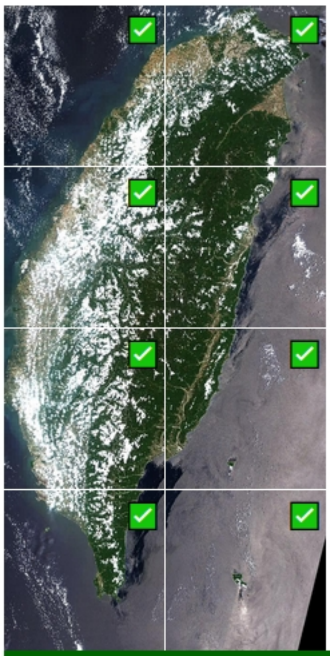
***Bali***

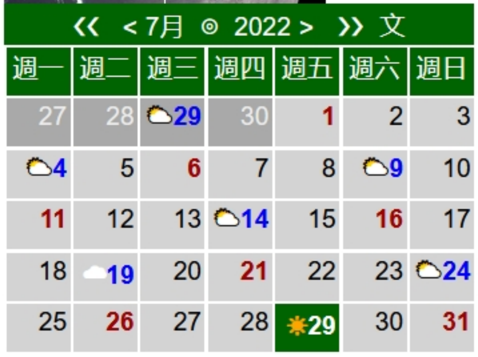


« < MAY ⊚ 2022 > » 文

| MON | TUE | WED | THU | FRI | SAT | SUN |
|---|---|---|---|---|---|---|
| ⛅25 | 26 | ☁27 | 28 | 29 | ☁30 | 1 |
| ⛅2 | 3 | 4 | ⛅5 | 6 | ⛅7 | 8 |
| 9 | ⛅10 | 11 | ☁12 | 13 | 14 | ⛅15 |
| 16 | ⛅17 | 18 | 19 | ☁20 | 21 | ☀22 |
| 23 | 24 | ☁25 | 26 | ☁27 | 28 | 29 |
| ☀30 | 31 | ☁1 | 2 | 3 | ⛅4 | 5 |

*Taiwan ROC*

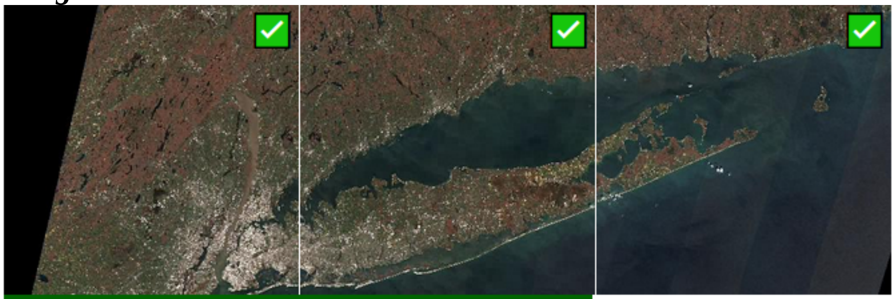


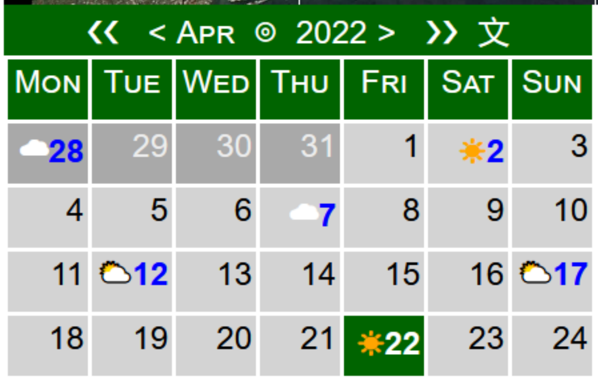


《 < 7月 ⊙ 2022 > 》 文

| 週一 | 週二 | 週三 | 週四 | 週五 | 週六 | 週日 |
|---|---|---|---|---|---|---|
| 27 | 28 | ⛅29 | 30 | 1 | 2 | 3 |
| ⛅4 | 5 | 6 | 7 | 8 | ⛅9 | 10 |
| 11 | 12 | 13 | ⛅14 | 15 | 16 | 17 |
| 18 | ☁19 | 20 | 21 | 22 | 23 | ⛅24 |
| 25 | 26 | 27 | 28 | ☀29 | 30 | 31 |

*Long Island New York*





《 < Apr ⊙ 2022 > 》 文

| Mon | Tue | Wed | Thu | Fri | Sat | Sun |
|---|---|---|---|---|---|---|
| ☁28 | 29 | 30 | 31 | 1 | ☀2 | 3 |
| 4 | 5 | 6 | ☁7 | 8 | 9 | 10 |
| 11 | ⛅12 | 13 | 14 | 15 | 16 | ⛅17 |
| 18 | 19 | 20 | 21 | ☀22 | 23 | 24 |

## 7. Implementation details

The following sections explain the implementation details instrumental in the creation of the web-app. These include:

- Declarative HTML & CSS grid – for customising the geographical parameters
- REST query and responses – to populate the calendar with acquisition data
- Fail-over image sourcing – to fill the tile-panel with thumbnail images
- Composite imaging – to render “split-tiles”
- Caching and Local Storage – to play fair with online live services
- Accessing metadata at Google Storage via CORS proxy in the Tile-Tab.

### *7.1 Declarative HTML & CSS grid*

The layout for a given place is declared in the HTML – e.g. for the island of Sardinia:

```
<div class="grid2x3" name="Sardinia" id="tiles"
     polygon="POLYGON((8.02+39,10.12+39,10.12+41,8.02+41,8.02+39))">
  <div id="T32TML" class="grid-item" orbits="R022 R065"></div>
  <div id="T32TNL" class="grid-item" orbits="R022 R065"></div>
  <div id="T32TMK" class="grid-item" orbits="R022 R065"></div>
  <div id="T32TNK" class="grid-item" orbits="R022 R065"></div>
  <div id="T32SMJ" class="grid-item" orbits="R022 R065"></div>
  <div id="T32SNJ" class="grid-item" orbits="R022"></div>
</div>
<div id="day" day="" lang="en it ca"></div>
<div class="grid7x7" id="calendar"></div>
```

Here, the **upper panel** is declared by the DIV element having the id attribute tiles – termed **#tiles**. It wraps the declaration of six tiles (also DIVs) that are to be laid out in the 2x3 grid pattern, as seen in Fig. 1. The bolded attributes of the tiles and the wrapper are discussed later in this section.
The **lower panel** is declared by two more DIV elements: **#day** for the calendar navigation bar, and **#calendar** for the month view 7x7 grid. Here the attribute **lang** indicates the chosen language locales: English, Italian, etc.

Grid layout [20] follows the CSS convention of specifying the class attribute grid*C*x*R* – where *C* and *R* are respectively the number of columns and rows in the grid. For example the above tile panel is laid out according to the following CSS declarations:

```
#tiles.grid2x3
 {
  display: grid;
  grid-template-columns: repeat(2,147px);  /* 2 cols, 343 total width */
  grid-template-rows:    repeat(3,147px);  /* 3 rows  */
  grid-gap: 1px 1px;                       /* visible tile borders */
}

#tiles div.grid-item { position: relative; }
```

The **polygon** attribute of #tiles wrapper DIV specifies an ordered list of longitude, latitude coordinate-pairs – in a format readily understood as a geo-located polygon by the CreoDIAS Finder web service. In our example **`POLYGON((8.02+39,10.12+39,10.12+41,8.02+41,8.02+39))`** there are five comma-separated pairs – with each pair coded as two numbers separated by a plus sign. By convention the last pair must repeat the first – so to close the polygon.

When designing the HTML for a given region it is convenient to select an initial rectangular polygon with the mouse interactively at the CreoDIAS Finder site – e.g. as a bounding box around the island of Sardinia. This is then to be successively reduced in width and height until the finder produces results only for the region of interest and not unwanted adjacent areas. Note, the polygon declared above corresponds to just a thin rectangle running down the central spine of Sardinia.
The designer/developer can then sift the Finder's search results into a list of all the tile **id** and **orbits** operational in the region. For example, the results for the tile with **`id="T32TML"`** coincide with just two distinct orbits: **`orbits`**`="`**`R022 R065`**`"`. The attributes can then be assigned to one of the tiles in the HTML grid – and so on until all the tiles are complete. Finally, the tiles in the grid must be sorted into traditional map order. This is easily achieved by following their N-S, E-W grid-order observed at the online map service at eAtlas [21] Note, our tile id is composed of the letter T followed by the code-name of the 100x100 km UTC tile that it is to represent.

## *7.2 RESTful Query and Responses*

The polygon once designed is then used by the web-app to compose a RESTful query at the CreoDIAS Finder for a chosen date range. The following URL does this by specifying our polygon for the calendar month-view: July 2022:

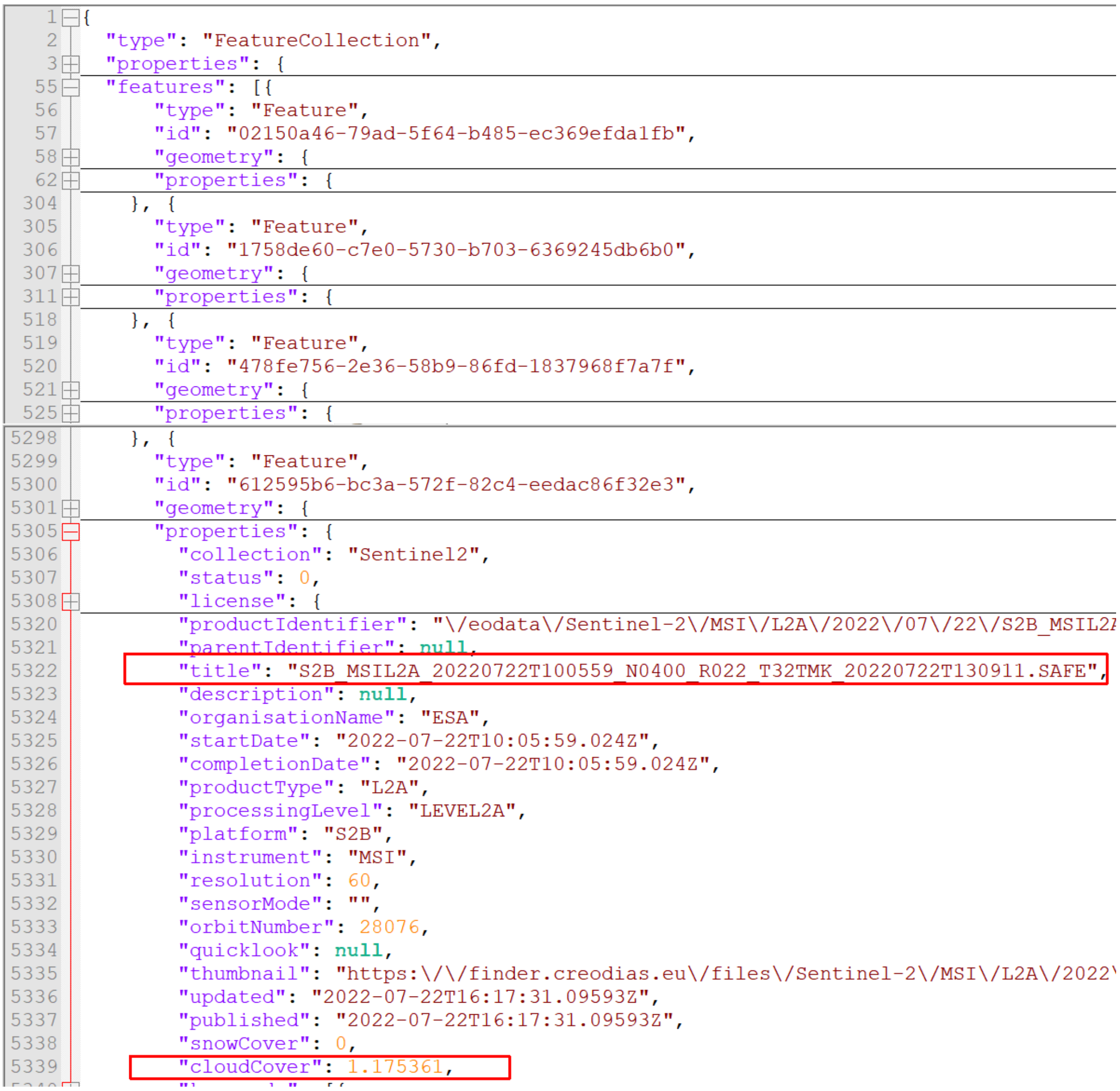
```
https://finder.creodias.eu/resto/api/collections/Sentinel2/search.json?
maxRecords=400&startDate=2022-06-27T00:00:00Z&completionDate=2022-08-
07T23:59:59Z&geometry=POLYGON((8.02+39,10.12+39,10.12+41,8.02+41,8.02+39))&so
rtParam=startDate&sortOrder=descending&status=all&dataset=ESA-DATASET
```

The URL composition and submission is implemented in JavaScript – where day1 and day2 are the start and end dates, and polygon is the `POLYGON` string.

```
deactivate_ui();

fetch( get_creodias_url(polygon, day1, day2), {headers: {'Origin':origin }} )
      .then((response) => {
          if(response.ok) {
             return response.json();
         }
         throw new Error("fetching creodias failed ****");
      })
      .then((jsonResponse) => {
        process_features(jsonResponse.features); // now do the business
      })
      .catch((error) => {
          console.log( error + "\n origin is: " + origin );
          activate_ui();
      });
```

Here the URL is composed by the `get_creodias_url` function then passed to the `fetch` function in order to submit the request using asynchronous promises. Note, the interactivity of the page gets deactivated until the request returns – which thus avoids any accidental double-clicking – which would otherwise hit the CreoDIAS server in unduly rapid succession (satisfying **R3**).
The returned response then can be parsed as a JSON object and stored as a collection of *Features.*



```
{
  "type": "FeatureCollection",
  "properties": {
  "features": [{
      "type": "Feature",
      "id": "02150a46-79ad-5f64-b485-ec369efda1fb",
      "geometry": {
      "properties": {
    }, {
      "type": "Feature",
      "id": "1758de60-c7e0-5730-b703-6369245db6b0",
      "geometry": {
      "properties": {
    }, {
      "type": "Feature",
      "id": "478fe756-2e36-58b9-86fd-1837968f7a7f",
      "geometry": {
      "properties": {
    }, {
      "type": "Feature",
      "id": "612595b6-bc3a-572f-82c4-eedac86f32e3",
      "geometry": {
      "properties": {
        "collection": "Sentinel2",
        "status": 0,
        "license": {
        "productIdentifier": "\/eodata\/Sentinel-2\/MSI\/L2A\/2022\/07\/22\/S2B_MSIL2
        "parentIdentifier": null,
        "title": "S2B_MSIL2A_20220722T100559_N0400_R022_T32TMK_20220722T130911.SAFE",
        "description": null,
        "organisationName": "ESA",
        "startDate": "2022-07-22T10:05:59.024Z",
        "completionDate": "2022-07-22T10:05:59.024Z",
        "productType": "L2A",
        "processingLevel": "LEVEL2A",
        "platform": "S2B",
        "instrument": "MSI",
        "resolution": 60,
        "sensorMode": "",
        "orbitNumber": 28076,
        "quicklook": null,
        "thumbnail": "https:\/\/finder.creodias.eu\/files\/Sentinel-2\/MSI\/L2A\/2022
        "updated": "2022-07-22T16:17:31.09593Z",
        "published": "2022-07-22T16:17:31.09593Z",
        "snowCover": 0,
        "cloudCover": 1.175361,
```

**Fig 2.** *Example response from CreaDIAS Finder in JSON format (extracts)*

Each Feature corresponds to a Sentinel-2 data product found within the polygon between the start and end dates. Each contains the relevant metadata for that Feature – as illustrated above where the two fields outlined in red are instrumental for the web-app:

- title – e.g. `S2B_MSIL2A_20220722T100559_N0400_R022_T32TMK_20220722T130911.SAFE`
- cloudCover – e.g. `1.175362` (as percentage)

The title field is later used to compose URLs for thumbnail images that fill the tiles in the web-app's upper panel – but first the field is parsed to obtain several sub-items:

- S2B – mission S2B, rather than S2A
- L2A – processing level L2A, rather than L1C
- 20220722 – i.e. 22/07/2022 the calendar date of the acquisition
- `R022` – the orbit number
- T32TMK – the tile id.

To populate the calendar month-view the Features are grouped by acquisition date. Date cells corresponding to days with one or more acquisition then receive several new attributes – e.g. our date-cell for 22/07/2022 is then written as follows:

```
<div class="grid-item dom ..." date="2022-07-22" cloud="☀"
L1C="S2B_MSIL1C_20220722T100559_N0400_R022_T_20220722T121536 32SMJ 32SNJ
32TMK 32TML 32TNK 32TNL"
L2A="S2B_MSIL2A_20220722T100559_N0400_R022_T_20220722T130911 32SMJ 32SNJ
32TMK 32TML 32TNK 32TNL">22</div>
```

These attributes include:

- `cloud` – that is to allow the styling of the cell based on the *cloudCover* values averaged over all available tiles
- `L1C` – that lists all level-1 tile products found for that acquisition date.
- `L2A` – that lists all level-2 tile products found for that acquisition date.

The latter two are space-separated lists for the two distinct product types. Each lists the UTC tile-names (e.g. `32SMJ 32SNJ 32TMK 32TML 32TNK 32TNL`) preceded by the common stem of their product name (e.g. `S2B_MSIL2A_20220722T100559_N0400_R022_T_20220722T130911`) – i.e. that name with tile-names excised. That way it is possible to recover the full product name for each listed tile when needed. Indeed, each populated date-cell is then assigned an event-handler which when clicked updates the upper-panel with thumbnail images from that day – i.e. by recovering the relevant product names, as discussed below.

Dates that are in the future are styled in red if they match planned acquisition dates according to ESA's 10 day orbital repeat schedule – this makes use of the aforementioned `orbits` attributes and is implemented by the `is_transit_day` algorithm (not shown here).

## *7.3 Fail-over image sourcing*

The upper-panel – see Fig 1. – of the web-app presents a map comprising thumbnail Sentinel-2 images – one per UTC tile – each being represented by a DIV element ready to be filled with a tempo-spatially mapped thumbnail image. Three cases arise, in order of preference:

✅ : using an L2A thumbnail – when the tile’s id is listed in the date-cell’s L2A attribute,
▨ : using an L1C thumbnail – when the tile’s id is listed in the date-cell’s L1C attribute,
◯ : leaving the tile completely black – when no data is available.

The latter can also happen when the orbital swath does not actually intersect the tile’s geographical extent. Also, just part of a tile can be black when that part lies outside of the orbital swath – e.g. at the top left corner in Fig.1. That tile has some content and thus counts as a regular acquisition.

Though not advertised as such, live online services often permit Sentinel-2 thumbnail images to be loaded at ones own site using the regular IMG element – as long as one knows the correct URL. However, this does not include ESA’s own Open Access Hub since it enforces registration and times-out logins on a strict frequency and stipulates content-disposition response headers. Neither does it include the Google Storage repository which it only provides thumbnails as JP2 files – since they are not a browser-native format. Live online services that do allow unencumbered access to thumbnails in browser-native JPG format include:

**CreoDIAS** – that has URLs of the type:
https://finder.creodias.eu/files/Sentinel-2/MSI/L2A/2022/07/22/
S2B_MSIL2A_20220722T100559_N0400_R022_T32TNK_20220722T130911.SAFE/
S2B_MSIL2A_20220722T100559_N0400_R022_T32TNK_20220722T130911-ql.jpg

**Peps-CNES** – that has URLs of the type:
https://peps.cnes.fr/quicklook/2022/07/22/S2B/
S2B_MSIL2A_20220722T100559_N0400_R022_T32TNK_20220722T130911_quicklook.jpg

**Sentinel-Hub.com** – that has URLs, only for L1C thumbnails, of the type:
https://roda.sentinel-hub.com/sentinel-s2-l1c/tiles/32/T/NK/2022/7/22/0/preview.jpg
Note: Sentinel-Hub.com thumbnails are exceptional in three ways:

a) their thumbnails are nearly always available across the globe.
b) their URLs can be composed with only the tile-id and the acquisition date (without product name) – so they are a “poor man's substitute” when all L2A sources fail.
c) Their thumbnails have white rather than black background pixels.

It is possible to try one thumbnail-image source after another until one successfully loads into the tile’s IMG element. This has a dual role:

- **Fail-over** image sourcing – when one live source goes down another cuts in.
- **Acquisition day feedback** – until the L2A image shows up the L1C image is shown, else a black tile may indicate nothing available yet.,

We implement thumbnail image-sourcing from four alternatives – in the following order

1. CreoDIAS *L2A*
2. PEPS-CNES *L2A*
3. PEPS-CNES *L1C*
4. Sentinel-Hub.com *L1C*.

This is achieved using a tail-recursive JavaScript function `loadAlternative` [22] in conjunction with the IMG element.

```
function loadAlternative( img_el, list )
{
  img_el.onload = fix_roda; // see later
  var tmp_img = new Image();
  tmp_img.onload  = function() { img_el.src = this.src; }
  tmp_img.onerror = function() {
       if( list.length ) loadAlternative( img_el, list );
    }
  tmp_img.src = list.shift(); //  pick off the first url in the list
}  // now the onload or onerror fires
```

This function depends on the IMG element's own `onload` and `onerror` event-handlers – whereby if the current thumbnail does not load the error is handled by passing on to the next alternative in the list.  For our example, the HTML of IMG element reads like this:

```
<IMG
src="https://finder.creodias.eu/files/Sentinel-2/MSI/L2A/2022/07/22/
S2B_MSIL2A_20220722T100559_N0400_R022_T32TNK_20220722T130911.SAFE/
S2B_MSIL2A_20220722T100559_N0400_R022_T32TNK_20220722T130911-ql.jpg"
data-alternative="https://peps.cnes.fr/quicklook/2022/07/22/S2B/
S2B_MSIL2A_20220722T100559_N0400_R022_T32TNK_20220722T130911_quicklook.jpg,
https://peps.cnes.fr/quicklook/2022/07/22/S2B/
S2B_MSIL1C_20220722T100559_N0400_R022_T32TNK_20220722T121536_quicklook.jpg,
https://roda.sentinel-hub.com/sentinel-s2-l1c/tiles/32/T/NK/2022/7/22/0/preview.jpg"
onerror="loadAlternative(this,this.getAttribute('data-alternative' ).split(','))"
>
```

Clearly this is not hand-coded. It is generated automatically for each tile – via JavaScript – whenever a new date is selected in the calendar panel.  As a result the best available thumbnail loads into each tile – and fills up the map.  Finally any tile marked with a green-tick gets assigned an event-handler so that a click on that tile open up the Tile-Tab (see Fig. 1) – as covered later.

The white background of a Sentinel-Hub.com thumbnail can disrupt the visual interpretation of the upper-panel – especially when mixed with other tiles having black backgrounds. Recent browser innovation now means that white image pixels can be efficiently made black within the browser – using the new standard JavaScript routines `getImageData` and `putImageData` [23] which allow read/write access to the pixels inside browser-loaded images – as seen in our example code below. There the function `fix_roda` is largely declarative – and as such the structure of the code provides little indication of the sequence of events that it implements.  In the interests of clarity, we outline some of that sequence below.

- An *Image Object* gets first instantiated as `img`;
- `img` then declares (but does not trigger) its own `onerror` and `onload` event-handlers: `onerror` caters for final fail-over;  `onload` contains all the image processing.
- Program control then arrives at the final statement of the function – where `img` is assigned a `src` attribute which instructs the browser to immediately try to load the image – and trigger either the `onerror` or `onload` code.

- Once triggered the onload handler uses `getImageData` to gain access to the pixel map of the now loaded image, and then passes that map to the `set_white_black` subroutine – so that all white pixels from the edge inwards become black.  The new pixel map is then rendered as an image in the tile's DIV – using `putImageData`.

```
function fix_roda( ev )
{
    var myImg = ev.target;
    var myDiv = myImg.parentNode;
    if( ! myImg.src.includes('roda.sentinel-hub.com') ) return; // not roda so do nothing
    myDiv.style.position = 'relative';
    var img = new Image();
    img.alt = "Loading...";
    img.crossOrigin =  ": *";
    img.onerror= function( e ) { console.log('img.onerror: ' + this.src ); };
    img.onload = function( ) {
        console.log('img.onload');
        var canvas = document.createElement('canvas');
        style_canvas( canvas, img.width, img.height, 1 );
        myDiv.appendChild(canvas);
        var ctx = canvas.getContext('2d');
        ctx.drawImage(img, 0, 0);
        const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
        const data = imageData.data;
        set_white_black( data, canvas.width, canvas.height );
        imageData.data = data;
        ctx.putImageData(imageData, 0, 0); // update into canvas
    }
    img.src = myImg.src; // triggers img.onload()
}

function is_white(data, i)
{ const TOP=252; // fuzz is 3
  return( (data[i++] > TOP) && (data[i++] > TOP) && (data[i] > TOP) ) ;
}

function set_black(data, i) { data[i++] = 0; data[i++] = 0; data[i] = 0; }

function set_white_black( data, width, height )
{
    for( var row=0; row<height; row++ )
    {   // Left-right direction
       var i = 4*row*width;
       for( var col=0; col<width; col++, i+=4 )
       {
           if( ! is_white(data, i) ) break; // out of this row
           set_black(data, i) ;
       }
       // Right-left direction
       i = 4*(row+1)*width  - 4;
       for( var col=0; col<width; col++, i-=4 )
       {
           if( ! is_white(data, i) ) break; // out of this row
           set_black(data, i) ;
       }
    }
}

function style_canvas( canvas, width, height, zIndex )
{
  canvas.width=width;    canvas.height=height; canvas.style.position = 'absolute';
  canvas.style.left = 0; canvas.style.top = 0; canvas.style.zIndex = zIndex;
}
```

## *7.4 Composite imaging – to render "split-tiles"*

An edge case arises whenever ESA decides to cut the data-take of the orbital swath directly above the given tile. ESA then delivers two L1C/L2A products for that tile – with the first being filled only in the upper-diagonal and the second only in the lower-diagonal – as shown below.

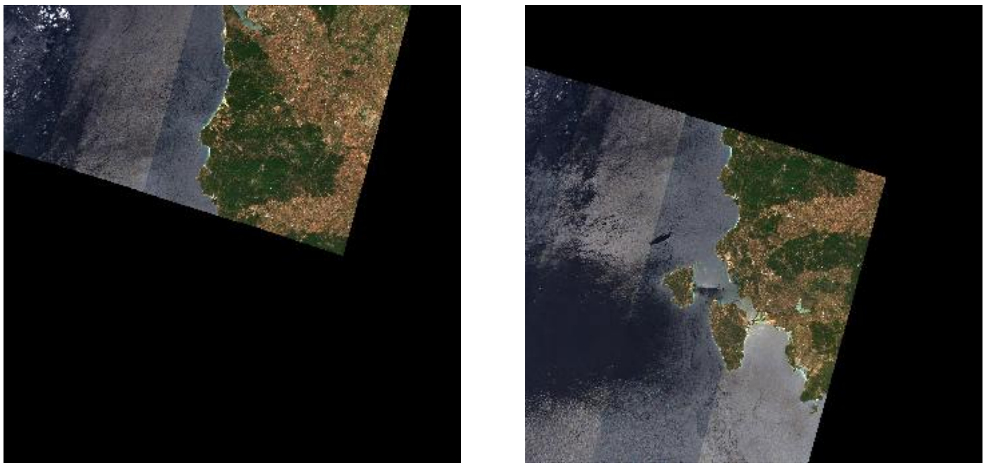


**Fig 3**. Example of the L2A "split-tile" left) upper-diagonal, right) lower-diagonal
Date: 2022-05-31: Tile T32SMJ

ESA leaves the task of combining the two "split-tiles" into one composite tile to the consumer. Indeed the task of combining two JPG thumbnail images, used to necessitate a back-end service violating **R1**. This can now occur efficiently within the browser – using the same JavaScript routines as before – e.g., as shown below

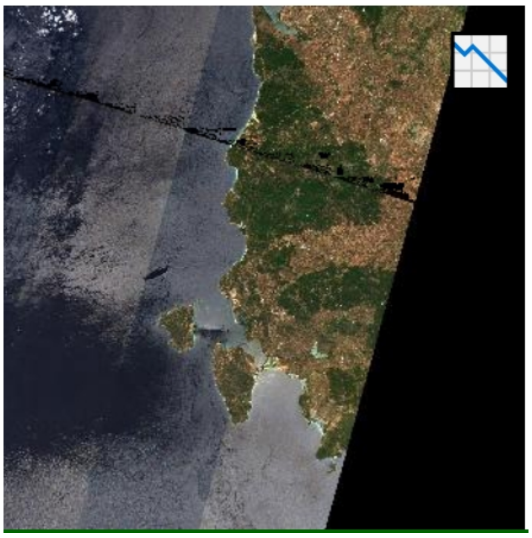


**Fig 4**. Composite thumbnail the L2A "split-tile" Date: 2022-05-31: Tile T32SMJ

Since it is useful to alert the user that a tile is split we leave a black line in the composite and mark the tile with a special ⧅ symbol. The black line indicates the orientation of the split. The composite thumbnail is implemented by function `load_composite` – where the thickness of that line. is determined, the input parameter `fuzz`. – see below.

```
function load_composite( id_str, src1, src1_bak, src2, src2_bak )
{
  var myDiv = document.getElementById( id_str );
  myDiv.style.position = 'relative';

  var img1 = new Image();
  img1.setAttribute('parent_id', id_str );
  img1.crossOrigin = 'anonymous';
  img1.onerror= function(ev) { ev.target.onerror=null; if(src1_bak) ev.target.src = src1_bak; }
  img1.onload = function( ) {
     if( img1.width + img1.height == 0 ) { img1.onerror(); return; }
     img_to_canvas( img1 );
     var img2 = new Image();
     img2.setAttribute('parent_id', id_str );
     img2.crossOrigin = 'anonymous';
     img2.onerror= function(ev) { ev.target.onerror=null; if(src2_bak) ev.target.src = src2_bak; }
     img2.onload = function( ) {
         if( img2.width + img2.height == 0 ) { img2.onerror(); return; }
         img_to_canvas(img2)
     }
     img2.src = src2; // triggers img2.onload()
  }
  img1.src = src1; // triggers img1.onload()
}

function img_to_canvas( img ) // img is a node/element <- this
{
    var canvas = document.createElement('canvas');
    style_canvas( canvas, img.width, img.height, 1 );

    const id_str = img.getAttribute('parent_id' );
    var myDiv = document.getElementById( id_str );
    myDiv.appendChild(canvas);
    var ctx = canvas.getContext('2d');

    ctx.drawImage(img, 0, 0);
    const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
    const data = imageData.data;
    set_black_transparent( data, 3 ); // fuzz = 3
    imageData.data = data;
    ctx.putImageData(imageData, 0, 0); // update into canvas
}

function style_canvas( canvas, width, height, zIndex )
{
  canvas.width=width;    canvas.height=height; canvas.style.position = 'absolute';
  canvas.style.left = 0; canvas.style.top = 0; canvas.style.zIndex = zIndex;
}

function set_black_transparent( data, fuzz )
{
  for( var i=0; i<data.length; i+=4 ) {
    if( (data[i] < fuzz) && (data[i + 1] < fuzz) && (data[i + 2] < fuzz) ) {
      data[i + 3] = 0; // set alpha = transparent where image is black
    }
  }
}
```

This code involves nested declarations one for each part of the split tile, as follows:

- An *Image Object* gets first instantiated as `img1`;
- `img1` then declares (but does not trigger) its own `onerror` and `onload` event-handlers: `onerror` caters for a fail-over; `onload` contains all the image processing.

- Program control then arrives at the final statement of the function – where `img1` is assigned a `src` attribute which instructs the browser to immediately try to load the image – and trigger either the `onerror` or `onload` code.
- Once triggered the onload handler calls `image_to_canvas` that uses `getImageData` to gain access to the pixel map of the now loaded image, and then passes that map to the `set_black_transparent` subroutine – so that all black pixels in it get marked as transparent. The new pixel map is then rendered as an image in the tile's DIV – using `putImageData`.
- Directly afterwards, a second *Image Object* is instantiated as `img2`
- `img2` then declares its own `onerror and onload` event-handlers.
- Program control then arrives at the penultimate statement of the function – where `img2` is assigned a `src` attribute which now instructs the browser to try to load that image – and trigger either its `onerror` or `onload` code.
- Its `onerror` and `onload` event-handlers are similarly designed to mark all black pixels in `img2` as transparent, and then overlay its new pixel map on to the tile's DIV.

## *7.5 Caching and Local Storage*

Any thumbnail image once loaded from a given online service typically gets cached by the browser – where it can be automatically retrieved for repeat viewing, without further hitting the service, and so play fair with that service (satisfy requirement **R3**).

However, the browser does not cache the search results served as JSON by the CreoDIAS Finder. Indeed it would be inappropriate to cache results from the current calendar month, as they are apt to update with new content arriving by the hour or by the day. The results for earlier months, however, are not expected to change so we take explicit measures to protect CreoDIAS from their undue repeat requests. To achieve this we deploy the browser's local-storage API [24] – which provides, per web domain, a few megabytes to store and retrieve items of text.

In particular, we store an item for each previously visited calendar month-view. Each item is made addressable by assigning it a unique key. In our case, we use keys like `Sardinia-2022-06` – where Sardinia is the name attribute of the tile-panel, 2022 is the viewed year and 06 the viewed month. The contents of each item is simply the HTML code that rendered the previously visited calendar month-view. The logic is as follows.

- Before making a new request to the CreoDIAS Finder the key is formulated for the {tile-panel name, year, month} combination.
- That key is looked up in local-storage:
- If **found** the item is retrieved and used to replace the HTML of the calendar month-view
- If **not found** the request to CreoDIAS Finder goes ahead and a new calendar month-view is built from the search results.
- If the latter case corresponds to a past month then the HTML of the new calendar month-view is written to local-storage using the new key formulated for that month-view.
- Eventually the local-storage fills up and older items get selectively retired by the browser.

## *7.6 Accessing meta-data at Google Storage via CORS proxy in the Tile-Tab*

The Tile-Tab resides in a separate browser tab – as indicated in the Mini User Guide and shown in Fig.1. It is launched from the web-app by a click on a chosen thumbnail in its upper-panel – as a natural gesture to zoom-in on the L2A product-data signalled as available by the green-tick. As such the Tile-Tab is an independent web page (under the same domain) which configures itself in response to the data it receives. These data are:

- Tile id and its product stems – there are 2 stems in the case of a split-tile. These parameters are passed via the URL's 'search-string' – e.g. as `&tile=T32SMJ&stems=S2A_MSIL2A_20220531T101611_N0400_R065_T_20220531T163910+S2A_MSIL2A_20220531T101611_N0400_R065_T_20220531T180718`
- Thumbnail data – originally from CreoDIAS or CNES-PEPS – obtained using the above parameters – as detailed earlier.
- Meta-data requested from Google Storage – see below

The Tile-Tab is built out of a simple HTML template that lays out a larger version of the thumbnail – above which we write the acquisition date, the tile id and allocate a row for some download links. This template gets duplicated for split-tiles so that the two individual L2A products will then be displayed side-by-side. Below the thumbnails there is a row of geolocation options.

***Downloads***

The task of generating the URLs for the download links is made non-trivial because the ESA format they follow cannot be wholly predicted from the band-name, tile-id and product stem. The following example is for the band B02 10m resolution JP2 image:

```
https://storage.googleapis.com/gcp-public-data-sentinel-2/L2/tiles/32/T/NK/
S2B_MSIL2A_20220722T100559_N0400_R022_T32TNK_20220722T130911.SAFE/GRANULE/
L2A_T32TNK_A028076_20220722T101518/IMG_DATA/R10m/
T32TNK_20220722T100559_B02_10m.jp2
```

It contains a string `A028076_20220722T101518` that cannot be predicted. Instead the URL – including that string – can be recovered from an axillary meta-data file that accompanies the product. The Tile-Tab does this by fetching a file called `MTD_MSIL2A.xml` from the given Google Storage folder, i.e.:

```
https://storage.googleapis.com/gcp-public-data-sentinel-2/L2/tiles/32/T/NK/
S2B_MSIL2A_20220722T100559_N0400_R022_T32TNK_20220722T130911.SAFE/
MTD_MSIL2A.xml
```

Ideally, this task would be as simple as fetching the JSON files from CreoDIAS Finder. In modern browsers the Cross-Origin Resource Sharing (CORS) protocol [25] defaults to preventing the browser accessing JSON or XML files served from other sites. However, the browser may include in its HTTP request the header – `{'Origin': document.location.origin}` and the server may participate in the protocol and signal that the files are to be accessible within the browser. CreoDIAS chooses to participate in CORS – but Google Storage does not. Nevertheless, a file is sent to the requester in both cases. Since Google declines to satisfy the CORS protocol the browser duly denies access.

Yet outside of the browser context access can easily be obtained – e.g. a Node_js program known as **CORSflare_js** is capable of reading meta-data files from Google Storage. CORSflare_js [26] is also capable of acting as a reverse proxy that:

1. receives request from the browser for a file housed at Google Storage
2. forwards the request to Google Storage – maintaining all headers, etc.
3. receives the response from Google Storage – including the file
4. attaches the permissive CORS protocol header
5. forwards the reply back to the browser

The entire sequence is transparent to the user. For this purpose we run CORSflare_js as our own live service on the Internet – without incurring significant back-end responsibilities – as follows:

1. by creating a non-billing account at *Cloudflare*
2. by deploying CORSflare_js as a *Cloudflare Worker* [27]
3. by routing all request for meta-data at Google Storage through that *Cloudflare Worker*.

That then allows our Tile-Tab JavaScript to read file `MTD_MSIL2A.xml`, extract the correct URLs and assign them for the 10m MSI images for the bands B02, B03, B04, B08 and TCI. Of course, it would be a simple matter to configure access to the other Sentinel-2 L2A bands.

***Geolocation***

The white cross shown in Fig. 1 indicates that the user has clicked on the Tile-Tab image to establish a point of interest. That point is then mapped onto a latitude, longitude coordinate so to allow the user to launch the *EO Browser* at that chosen geo-location. However, that mapping needs additional meta-data: in particular, the UTM latitude and longitude coordinates of the upper left corner of the tile and the WGS84/UTM zone label.
That data can be obtained from a second file – **MTD_TL.xml** – housed at Google Storage, e.g. at:

```
https://storage.googleapis.com/gcp-public-data-sentinel-2/L2/tiles/32/T/NK/
S2B_MSIL2A_20220722T100559_N0400_R022_T32TNK_20220722T130911.SAFE/GRANULE/
L2A_T32TNK_A028076_20220722T101518/MTD_TL.xml
```

Again the same *Cloudflare Worker* is needed to gain access to that file in the browser. In particular, we extract the following attributes from the file: `ULX`, `ULY`, `HORIZONTAL_CS_NAME`. These are then used to derive the latitude, longitude coordinates beneath the white cross, to display the coordinate in the row beneath the Tile-Tab image, and to formulate the direct URL into the EO Browser – e.g.:

```
https://apps.sentinel-hub.com/eo-browser/?lat=40.09&lng=9.57&zoom=16&time=2022-
07-22&preset=1_TRUE_COLOR&datasource=Sentinel-2%20L2A
```

For small regions of interest it may be desirable to skip the Tile-Tab page and instead navigate directly from the Web-app to the chosen location using an URL such as that above.

## 8. Extracting NDVI using EO Browser launched from the Tile-Tab

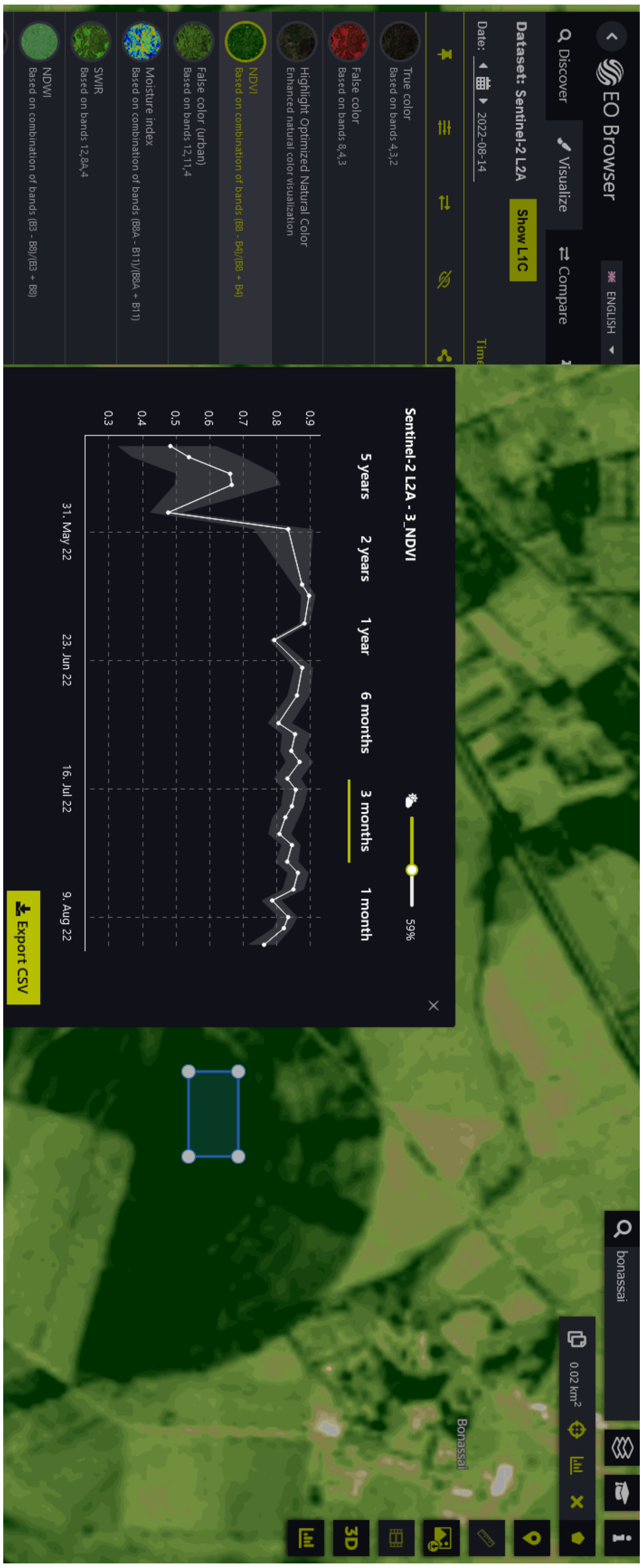
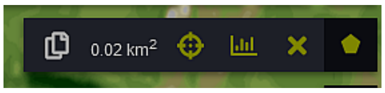


***Fig 5. Screenshot of the EO Browser:*** *Making a graph of NDVI for a polygon at Bonassai in Sardinia – © Sinergise, Ljubljana, Slovenia.*

Once launched for the given time and location from the Tile-Tab (see Fig. 1) the EO Browser can then be used to display the Normalized Difference Vegetation Index (NDVI) rather than the usual colour image. In particular, the tool at the top right corner of the EO Browser



can be used to draw a small polygon (in blue) onto the NDVI image and then made to display a graph of the variation over several months, or years, of the NDVI – in order to assess state of the vegetation in the targeted zone. **Fig 5** show a screenshot of the process.

More experienced users might choose instead to download MSI images from the Tile-Tab and carry out their own analysis on a local workstation. Although such analysis is beyond the scope of this report it can be achieved using ESA's SNAP software, a GIS, or by using custom Gdal based scripts.

## 9. Discussion

As demonstrated above in the Gallery section the *IsolaS2* web-app is well suited for a variety of islands and regions around the globe – especially where:

- The range of longitudes is such as to not to signal more than one acquisition per calendar date-cell.

- Adjacent UTM tiles do not unduly overlap.

In the latter case, when the tile's thumbnails are assembled into grids the repeat of the overlapping areas between adjacent tiles will have only minor visual salience – e.g. see the cloud above the Isle of Skye in the grid of the *Hebrides*.
However, the inhomogeneous infrastructure of the UTM tessellation means that, in certain key geographical loci, tile-overlaps can be large enough to distort the coherence of our web-apps depiction of an island or region. For example, the island of Crete – see Fig. 7 – is particularly unsatisfactory for two reasons:

- the large longitudinal extent of the island means only roughly half of it is visible on any given acquisition day – in Fig 6. we show just the Western part.

- the large tile-overlap occurring for Western Crete results in a caricature of the shape of the place – rather than its more recognisable form.

The established remedy to overlapping tiles is to compute a mosaic from the tiles and present it instead. Our web-app does not attempt that. It might be technically possible via pixel-manipulation in the browser – but that is beyond the scope of the current work. Mosaicking is already done well in the EO Browser which can be called directly from the Tile-Tab, or in ESA's SNAP operating upon downloaded JP2 band images.

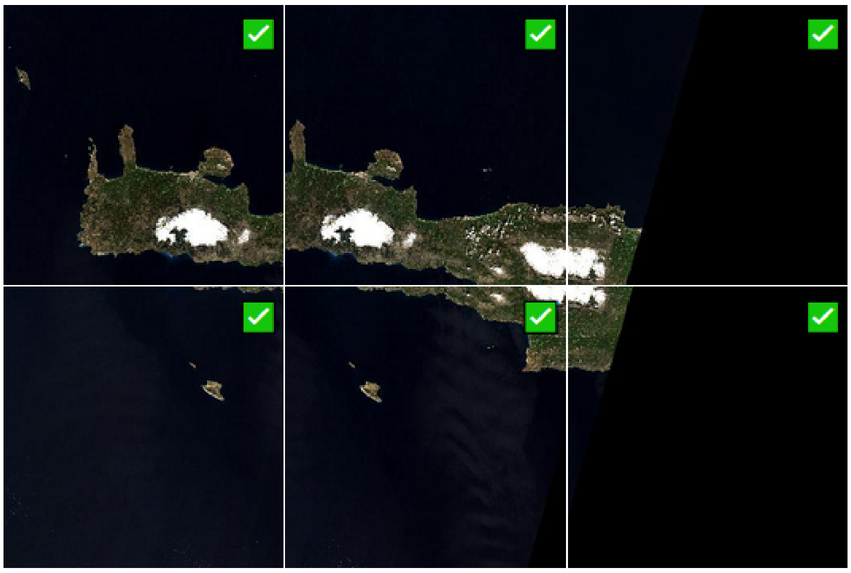


***Fig 6. Western Crete with visual disruptive repetition***

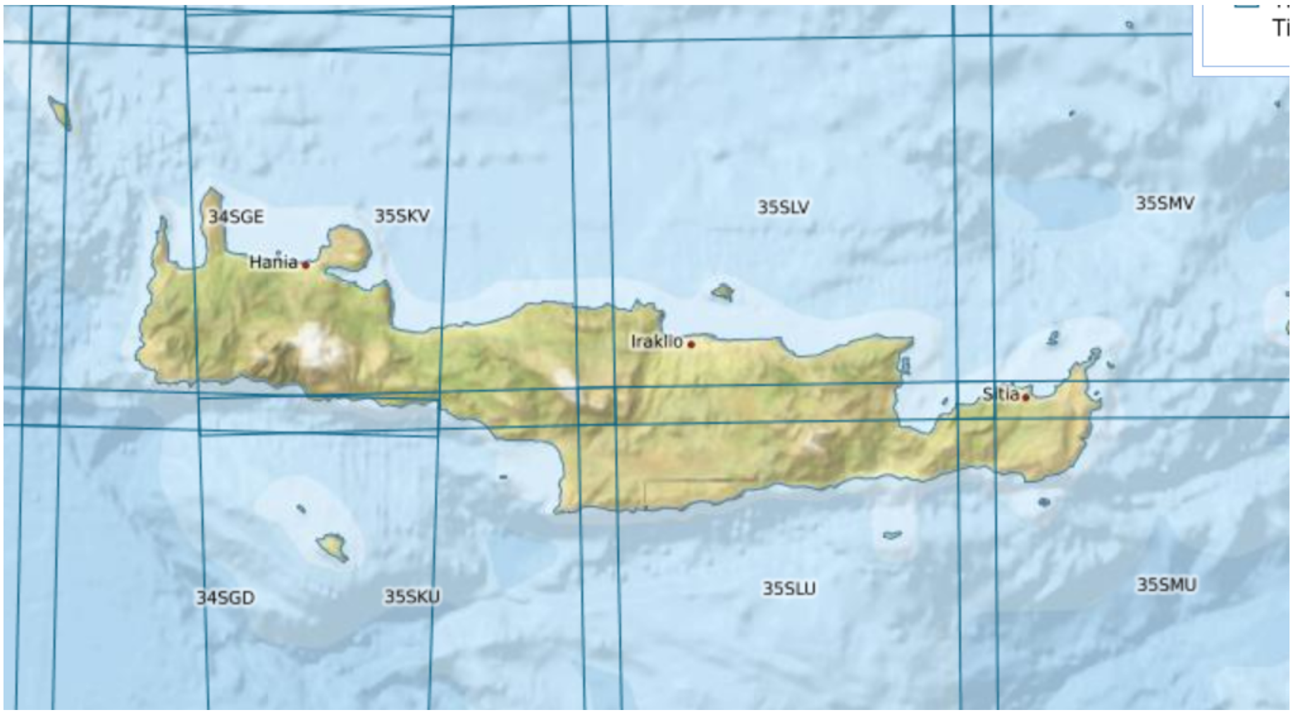


***Fig 7. Overlapping UTM Tessellation for the island of Crete***

Although, the web-app successfully deploys a CORS proxy via a CloudFlare Worker it would nevertheless be preferable that Google participate in the CORS protocol. To that end on 13 July 2021 the feedback was sent to the providers of the Bucket gcp-public-data-sentinel-2. No reply or solution has arrived at the time of writing. So the danger remains that the CloudFlare Worker may in time become a bottleneck as the number of concurrent users of the Tile-Tab module increases. A work-around would be to dedicate one CloudFlare Worker per geographical configuration – yet that would run counter to our Minimal Maintenance requirement (**R1**).

## 10. Conclusions

All seven of the list of must-have requirements are satisfied by the CRS4 web-app *IsolaS2* deploying several state-of-the-art innovations – with the following minor provisos:

**R1** middleware – Although *IsolaS2* has no back-end stack. We have had to resort to a degree of Middleware whenever live services fail to participate in the CORS protocol. This is currently the case for Google Storage – where we deploy the CORS proxy, as explained above – but which incurs no costs and, once up and running, requires zero maintenance. Occasionally, CreoDIAS Finder becomes misconfigured and also fails to participate in the CORS protocol. To date we have observed three periods of failure – often lasting a day or so. For that reason we have now programmed our fetch code to catch such failure and then automatically retry via a similar CORSflare proxy.

**R5** UTM tile-overlap – Although *IsolaS2* is generally configurable to other geo-locations of similar size and shape a few locations incur a UTM tessellation that disrupts the ready visual interpretation of the tiles in the upper panel – as was illustrated above for Western Crete.

The web-app *IsolaS2* – at the time of writing – runs live at the following URL:

https://crs4.github.io/IsolaS2/src

with access to each of the of the islands and regions shown in the Gallery Section above.

As such, the web-app *IsolaS2* helps lower the architectonic barriers experienced by members of the public, citizen scientists, nascent micro-enterprises and Regional government departments wishing to quickly assess the convenience of using of Sentinel-2 MSI data for land-use analysis across their particular region or island. Finally, it does so in an intuitive and readily discoverable manner.

## Acknowledgments

This project was inspired by Gianluigi Zanetti and sustained by Luca Pireddu at CRS4.

## References

1. Brelstaff G, Moehrs S, Anneda P, Tuveri. M, Zanetti G (2000). Electronic Patient Records on the Java InfoBus. In: Proceedings of the Second International Conference on the Practical Applications of Java (PAJava 2000). p. 45-58, Blackpool, UK:The Practical Application Company, ISBN: 1-902426-09-6
2. Brelstaff G, Chessa F (2011). Interactive alignment of Parallel Texts a cross browser experience. In: W3C Workshop: Content on the Multilingual Web. Pisa Italy, 4-5 April 2011, W3C Consortium
3. Chessa F, Brelstaff G (2011). Going Beyond Google Translate. In: Proceeding *Chitaly* Proceedings Of The 9th ACM SIGCHI Italian Chapter International Conference on Computer-Human Interaction: Facing Complexity . Alghero SS, p. 108-113, New York:ACM, ISBN: 978-1-4503-0876-2, doi: 10.1145/2037296.2037324
4. Brelstaff G, Chessa F (2014, April) Precision phrase linking and pulling – *Ispantu* in Footnotes, comments, bookmarks, and marginalia on the Web - A W3C Workshop on Annotations W3C Workshop on Annotations num. 1 Ivan Herman, Doug Schepers W3C
5. Brelstaff, G., & Chessa, F. (2015, May). Exceptional texts on the multilingual web. In *Proceedings of the 24th International Conference on World Wide Web* (pp. 847-851).
6. Brelstaff G. & Zanetti G. (2020) Earth Observation Satellite Multi-Spectral Imaging Data for Sardinian Land-Use Application: Sentinel-2 and Landsat-7/8. CRS4 Report.
7. Brelstaff G. (2021) Fostering inclusive regional access to Sentinel-2 data. CRS4 Report.
8. ESA Sentinel-2 https://sentinel.esa.int/web/sentinel/missions/sentinel-2
9. USGS Landsat 7 https://www.usgs.gov/landsat-missions/landsat-7
10. USGS Landsat 7 https://www.usgs.gov/landsat-missions/landsat-8
11. Sci-Hub, ESA Copernicus Open Access Hub https://scihub.copernicus.eu/dhus/#/home
12. EO Browser , Sinergise, Ljubljana, Slovenia https://apps.sentinel-hub.com/eo-browser
13. CreoDIAS Finder, CloudFerro, Warsaw, https://finder.creodias.eu
14. GoogleStorage Sentinel-2 https://storage.googleapis.com/gcp-public-data-sentinel-2
15. ESA's SNAP software https://step.esa.int/main/download/snap-download/
16. Gdal based scripts https://gdal.org/
17. IrfanView graphic viewer. https://www.irfanview.com/plugins.htm
18. Firebase hosting service https://firebase.google.com
19. GitHub Pages https://pages.github.com
20. CSS Grid layout https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout
21. eAtlas web mapping system https://maps.eatlas.org.au
22. loadAlternative Javascript function from stackoverflow.com
23. The ImageData object JavaScript API https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas
24. Window.localStorage JavaScript API https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
25. Cross-Origin Resource Sharing an HTTP-header based mechanism https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
26. CORSflare_js a lightweight JavaScript CORS Reverse Proxy designed to run in a *Cloudflare Worker*. https://github.com/Darkseal/CORSflare
27. Cloudflare Worker, a serverless execution environment https://workers.cloudflare.com/