Technical Report

C3S_425_Predictia - Web architecture to develop demonstrators based on the CDS Toolbox

Issued by: Predictia Intelligent Data Solutions SL

Date: 21/01/2019

Ref: C3S_425_201901_Technical_Report_v1

Contributors

Predictia Intelligent Data Solutions SL
Fernando Martín Marlasca fmartin@predictia.es
Daniel San Martín Segura daniel@predictia.es
Markel García Díez garciam@predictia.es

Table of contents

Introduction

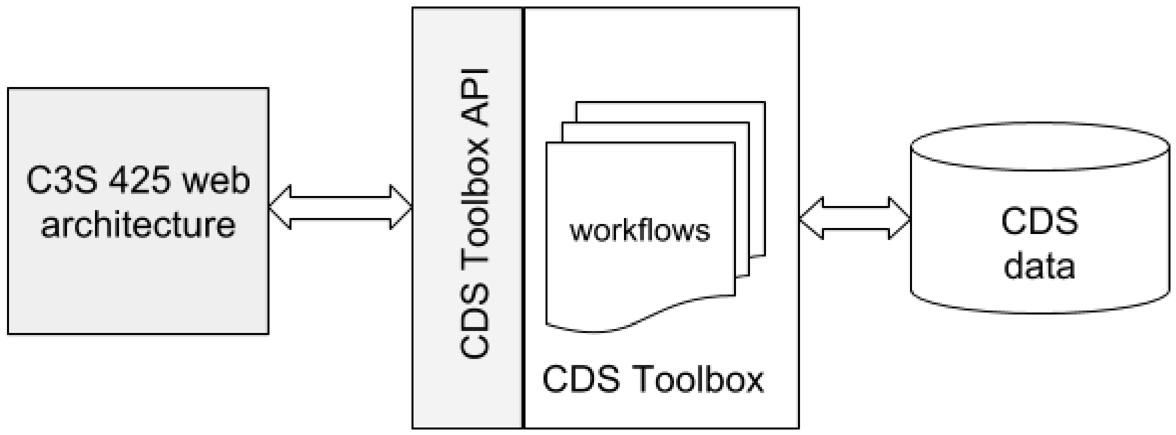
The scope of the tender “C3S 425: Migration of Sectoral Information System demonstrators onto the operational infrastructure” is to reimplement five demonstrators previously developed in the SIS by using the Climate Data Store and the associated Toolbox. During this reimplementation a redesign of the user interfaces will be performed to obtain an homogeneous graphical design and user experience.

The project started on December 2017 and is expected to end during the first semester of 2019.

The objective of this document is to describe the technical solution that has been built to achieve the objectives of the project. To avoid implementing five completely independent demonstrators, it was proposed to develop a common web architecture that could be adapted to their common and particular functionalities. The resulting software architecture is not a general framework, but can be used to ease the development of new demonstrators.

Architecture overview

The proposed architecture is described in the figure below:



The CDS Toolbox is used to generate graphical products (charts, maps…) and to gather data. This products and datasets are generated by a set of workflows that have to be defined for each demonstrator. The web architecture retrieve this information from the CDS Toolbox by making asynchronous HTTP requests. This document will focus on describing the web architecture and not the required workflows to generate graphical products.

The demonstrators cope with a diversity of data sources: observations, climate projections and seasonal forecasts for climate variables or sector-specific indices. Obviously, the nature of these different data sources determine very different parameters in order to fetch a concrete data result (a forecast or an observation). For instance, a specific scenario has to be defined for climate projections and a runtime value is needed to determine a seasonal forecast.

The selection procedure also includes the many different Essential Climate Variables (ECV) and sector-specific climate indices that have been identified in the demonstrators as relevant for end users. Therefore, each result is defined by an important number of parameters, and these parameters change between different types of results. This way, user interfaces have to provide an important number of selection elements (HTML form elements such as selects, check boxes, sliders or radio buttons) to enable this data selection.

Evidently, there are strong interdependencies between these selection elements. Some of these dependencies are hierarchical given by the nature of the different data sources (the selection of an scenario is only available for climate projections). However, some application-specific constraints produce non-hierarchical dependencies. For instance, in the ECEM demonstrator, seasonal forecasting is only available at a country level but not at a cluster level. This type of constraints force to use discovery services in order to know the availability of results.

These parameters are selected via a menu where all of these constraints and inter-dependencies are defined.

Since demonstrators have been designed as Geographic Information System (GIS) clients, maps constitute the primary data visualization support.

The proposed web architecture has been designed to ease the development of this kind of GIS user interfaces: a complex menu which enables the selection of a dataset to be displayed over a map/chart.

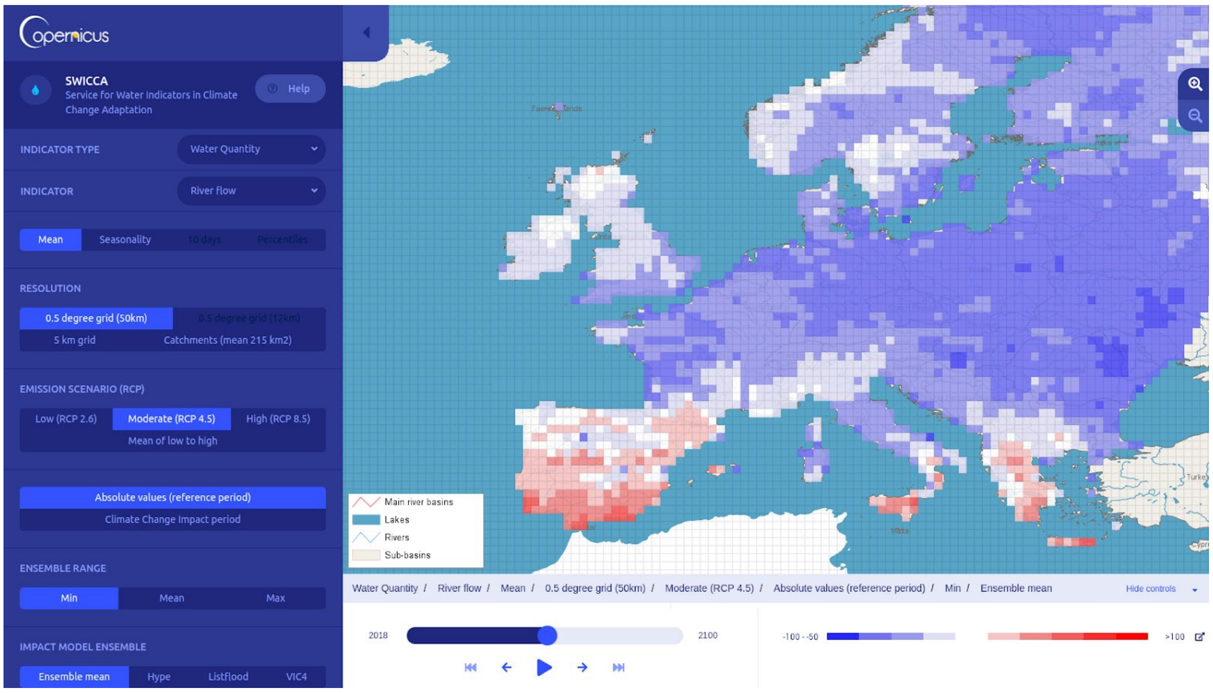
These are the main technical characteristics of the resulting architecture:

- A Single-Page Application (SPA) approach have been followed. SPA approach avoids interruption of the user experience between successive pages, making the application behave more like a desktop application. To achieve this objective, there is an intensive usage of asynchronous requests that enables the partial update of a web page.
- This application has been developed using the framework Angular v.5.
- The business logic of the application is based on a set configuration files to define: the menu elements and its inter-dependencies, the CDS Toolbox workflows to be executed, etc.
- It has been designed to be as flexible as possible and to have the possibility of defining new functionalities over it.
- New responsive user interface with an improved and homogeneous user experience.

It is important to note that the web architecture does not aim to automatically create new demonstrators. Each demonstrator has its own characteristics and features, so they shall have big differences in their implementation. This web architecture aims to ease the development of new demonstrators since it solves many common requirements. It is proposed to be the base project for the development of new demonstrators. Therefore, deep knowledge of the Angular framework is required to make use of this web architecture.

The goal of this document is to explain how to create a new demonstrator in this application, with its configuration files, always having in mind that each demonstrator will have the need to develop and implement its own ad-hoc features.

This is a preliminary version of the documentation and will be extended over the next months (including the base source code and the implementation of several demonstrators).



SWICCA
Service for Water Indicators in Climate Change Adaptation

INDICATOR TYPE: Water Quantity

INDICATOR: River flow

[✓ Mean] Seasonality 10 days Percentiles

RESOLUTION
[✓ 0.5 degree grid (50km)] 0.5 degree grid (12km) 5 km grid Catchments (mean 215 km2)

EMISSION SCENARIO (RCP)
Low (RCP 2.6) [✓ Moderate (RCP 4.5)] High (RCP 8.5) Mean of low to high

[✓ Absolute values (reference period)] Climate Change Impact period

ENSEMBLE RANGE
[✓ Min] Mean Max

IMPACT MODEL ENSEMBLE
[✓ Ensemble mean] Hype Listflood VIC4

Main river basins
Lakes
Rivers
Sub-basins

Water Quantity / River flow / Mean / 0.5 degree grid (50km) / Moderate (RCP 4.5) / Absolute values (reference period) / Min / Ensemble mean

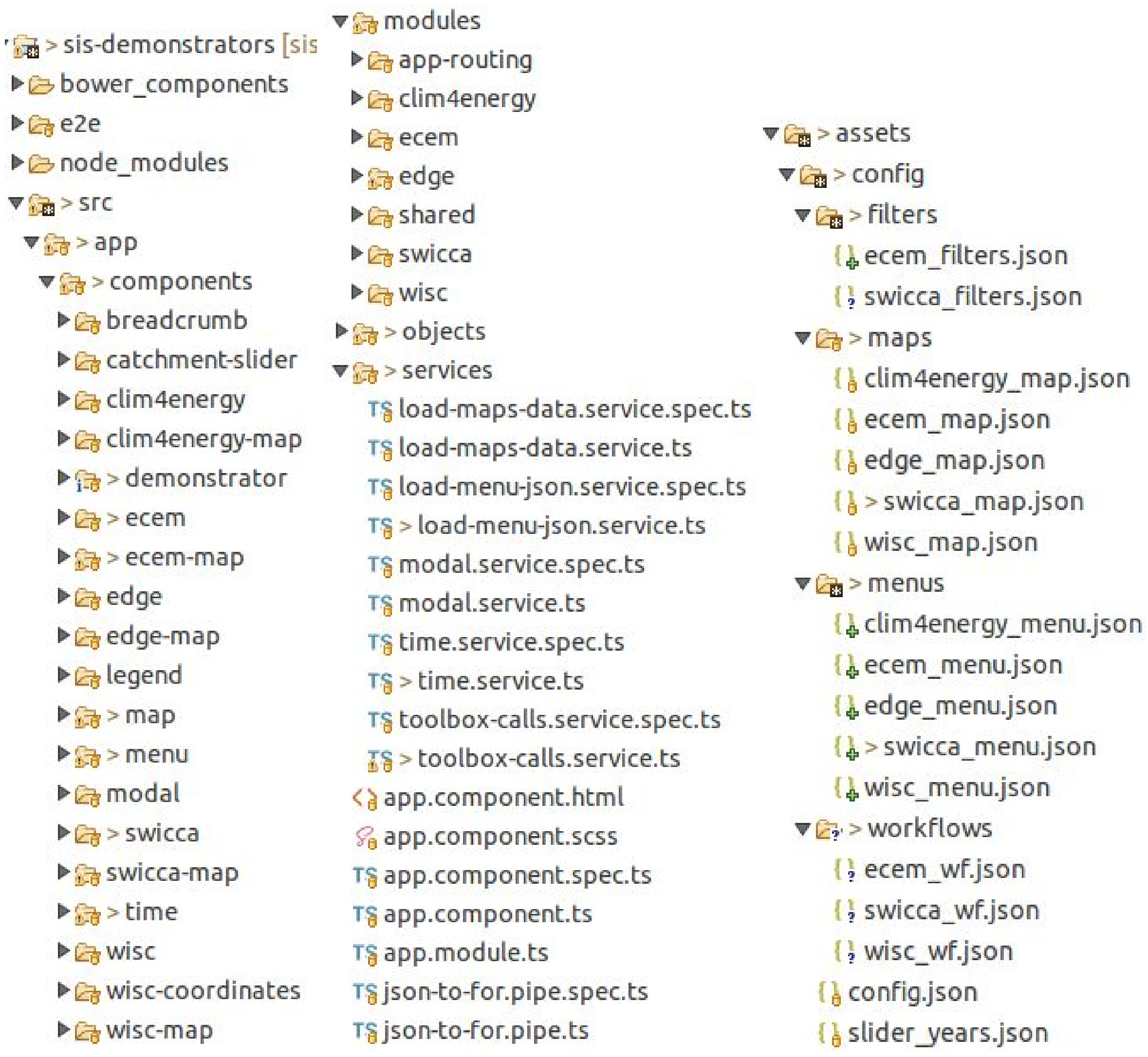
2018 2100

-100 - -50 >100

Screenshot of the new user interface

Application structure

The following diagram shows the file and folder structure containing the configuration files to define a set of demonstrators. This structure follows the guidelines of Angular.



```
▾ > sis-demonstrators [sis
  ▸ bower_components
  ▸ e2e
  ▸ node_modules
  ▾ > src
    ▾ > app
      ▾ > components
        ▸ breadcrumb
        ▸ catchment-slider
        ▸ clim4energy
        ▸ clim4energy-map
        ▸ > demonstrator
        ▸ > ecem
        ▸ > ecem-map
        ▸ edge
        ▸ edge-map
        ▸ legend
        ▸ > map
        ▸ > menu
        ▸ modal
        ▸ > swicca
        ▸ swicca-map
        ▸ > time
        ▸ wisc
        ▸ wisc-coordinates
        ▸ wisc-map
```

```
▾ modules
  ▸ app-routing
  ▸ clim4energy
  ▸ ecem
  ▸ edge
  ▸ shared
  ▸ swicca
  ▸ wisc
▸ > objects
▾ > services
    load-maps-data.service.spec.ts
    load-maps-data.service.ts
    load-menu-json.service.spec.ts
    > load-menu-json.service.ts
    modal.service.spec.ts
    modal.service.ts
    time.service.spec.ts
    > time.service.ts
    toolbox-calls.service.spec.ts
    > toolbox-calls.service.ts
  app.component.html
  app.component.scss
  app.component.spec.ts
  app.component.ts
  app.module.ts
  json-to-for.pipe.spec.ts
  json-to-for.pipe.ts
```

```
▾ > assets
  ▾ > config
    ▾ > filters
        ecem_filters.json
        swicca_filters.json
    ▾ > maps
        clim4energy_map.json
        ecem_map.json
        edge_map.json
        > swicca_map.json
        wisc_map.json
    ▾ > menus
        clim4energy_menu.json
        ecem_menu.json
        edge_menu.json
        > swicca_menu.json
        wisc_menu.json
    ▾ > workflows
        ecem_wf.json
        swicca_wf.json
        wisc_wf.json
      config.json
      slider_years.json
```

Creating new demonstrators

For the creation of a new demonstrator, the following tasks must be carried out:

- Create a new **Angular component** for the *demonstrator*.

  app > component > *demonstrator* > *demonstrator*-demonstrator.component

  For the **HTML** of this component, another demonstrator can be taken as an example, because some HTML elements must be placed in order to have a good visualization and behavior of the demonstrator.
  The **javascript** of the component is explained in the *demonstrator*-filters.json explanation.

- Create a new ***map* component** for this demonstrator that has to be instantiated in the previous component HTML.

  app > component > *demonstrator*-map > *demonstrator*.map.component

  The HTML is just a *div* to instantiate there the leaflet map (leaflet library is used to generate and manage maps).
  In the JavaScript code, functions can be added in order to add new layers, new styles for the layers… The functionality to add the base layer and to add the defined controls to the map is developed in the **map.component**.

- Create **two new modules** in the folder modules > *demonstrator*
  - *demonstrator*-routing.module > with a reference to the created *demonstrator*-demonstrator.component.
  - *demonstrator*.module > it loads the necessary modules, services and components.

- Create a new configuration file that will **define the menu** elements of the new demonstrator and add the reference to this json file in *load-menu-json.service.ts.*

  assets > config > menus > *demonstrator*_menu.json

  The explanation of how to create this JSON file can be found in **Menu configuration**

- Create a new configuration file in order to **choose the workflow** of the toolbox that will be executed (by sending parameters to it) to show the data selected in the menu in the demonstrator map, and add the reference to this JSON file in *load-menu-json.service.ts.*

  assets > config > filters > *demonstrator*_filters.json

The explanation of how to create this json file is in **Filter configuration**

- Create a new configuration file in order to define the data necessary to call the workflow, identifier, function name. WORK IN PROGRESS

  assets > config > workflows > *demonstrator*_wf.json

- Create a new configuration file in order to define some **parameters used in the map** of the new demonstrator.

  assets > config > maps > *demonstrator*_map.json

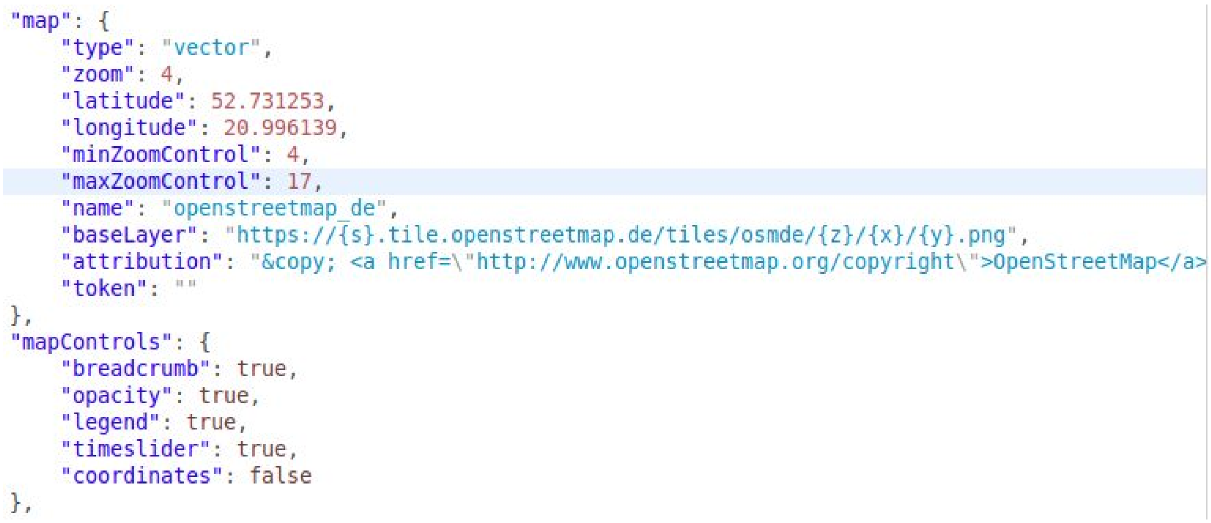
  In this file the following parameters can be configured (which belong to the element 'map'):

  - latitude: to define the center of the map
  - longitude: to define the center of the map
  - zoom: default zoom at the load
  - minZoomControl: minimum zoom to be controlled by the zoom control
  - maxZoomControl: maximum zoom to be controlled by the zoom control
  - type: of the map, 'vector' or 'raw'
  - baseLayer: URL of the baselayer that we want to use
  - attribution: of the baselayer
  - token: access token if needed to use the map

  In the same file, in the element "mapControls", we can configure the controls that we want to display in the map.

  - breadcrumb
  - opacity
  - legend
  - time-slider
  - coordinates

Example:

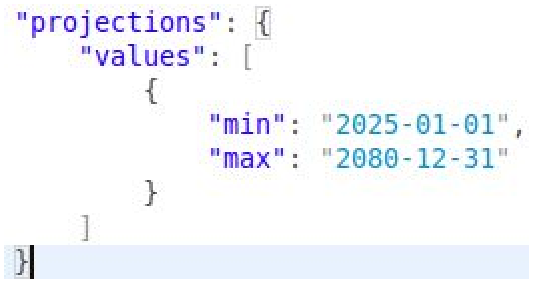


```
"map": {
    "type": "vector",
    "zoom": 4,
    "latitude": 52.731253,
    "longitude": 20.996139,
    "minZoomControl": 4,
    "maxZoomControl": 17,
    "name": "openstreetmap_de",
    "baseLayer": "https://{s}.tile.openstreetmap.de/tiles/osmde/{z}/{x}/{y}.png",
    "attribution": "&copy; <a href=\"http://www.openstreetmap.org/copyright\">OpenStreetMap</a>
    "token": ""
},
"mapControls": {
    "breadcrumb": true,
    "opacity": true,
    "legend": true,
    "timeslider": true,
    "coordinates": false
},
```

- If in the demonstrator there is going to be a time-slider control, the time values must be configured in the *slider_years.json* file, in this way:

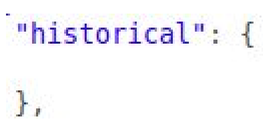
  A new node with the name of the demonstrator has to be added to the file. Inside of it, menu options must be added by adding subtrees, until having the combination of options where we want to add the control and its time values. Examples:

  1. If we want the time control in a demonstrator for all the combinations with the “projections” option selected:



```
"projections": {
    "values": [
        {
            "min": "2025-01-01",
            "max": "2080-12-31"
        }
    ]
}
```

  2. If we do not want the time control with the “historical” option selected:



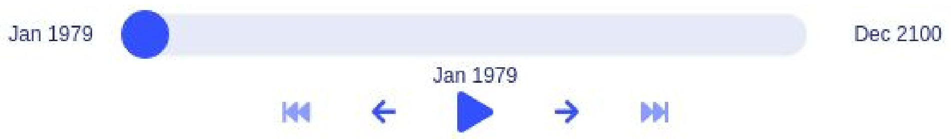
```
"historical": {

},
```

  3. If we want the control with a combination of options (projections, climate, climate_model_climate), one of its values with a time option, and the others with others:

```
"projections": {
    "climate": {
        "climate_model_climate": {
            "rcm1": {
                "values": [
                    {
                        "min": "1979-01-01",
                        "max": "2098-12-31"
                    }
                ]
            },
            "others": {
                "values": [
                    {
                        "min": "1979-01-01",
                        "max": "2100-12-31"
                    }
                ]
            }
        },
```

There are different ways to show the time values in the *time-slider*:

1. From an initial date to an end date:

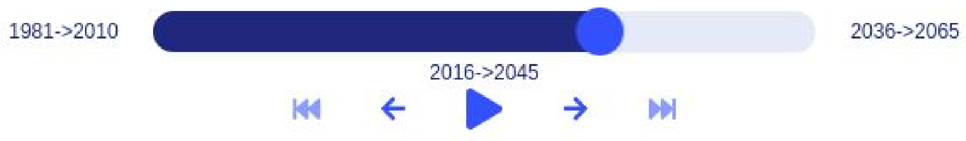


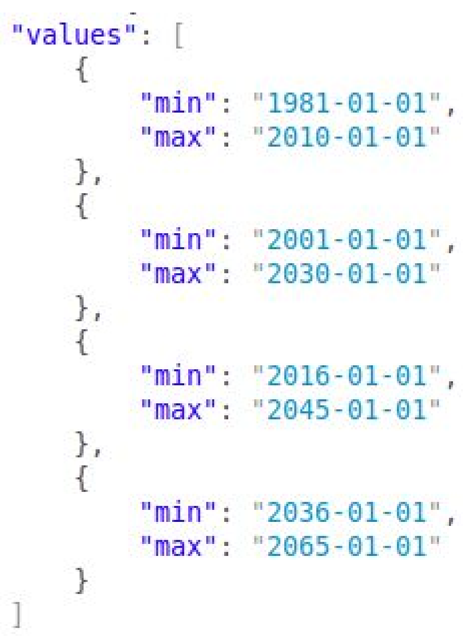
```
"values": [
    {
        "min": "2015-01-01",
        "max": "2065-12-31"
    }
]
```

2. Number of steps by month (for seasonal options):

```
"values": [
    {
        "unit": "month",
        "step": 1,
        "max": 6
    }
]
```

3. Exact steps:





```
"values": [
    {
        "min": "1981-01-01",
        "max": "2010-01-01"
    },
    {
        "min": "2001-01-01",
        "max": "2030-01-01"
    },
    {
        "min": "2016-01-01",
        "max": "2045-01-01"
    },
    {
        "min": "2036-01-01",
        "max": "2065-01-01"
    }
]
```

Menu configuration

In order to configure a menu for a demonstrator, a JSON file must be written, with an element for each item of the menu. If multiple demonstrators will be created for a single application, there will be a JSON file for each demonstrator.

Schema of the JSON file

```
{
   "$schema":"http://json-schema.org/draft-07/schema#",
   "$id":"http://example.com/root.json",
   "type":"array",
   "title":"Demonstrator Menu Configuration",
   "description":"The menu configuration for one demonstrator",
   "items":{
      "$id":"#/items",
      "type":"object",
      "required":[
         "id",
         "label",
         "description",
         "type",
         "value",
         "options",
         "breadcrumb"
      ],
      "properties":{
         "id":{
            "description":"Unique identifier",
            "type":"string",
            "examples":[
               "spatial_aggregation"
            ]
         },
         "label":{
            "description":"Text to show, reference to the i18n file",
            "type":"string",
            "examples":[
               "selector.spatial_aggregation.label"
            ]
         },
         "description":{
            "description":"Text of the tooltip, ref. to the i18n file",
            "type":"string",
            "examples":[
               "selector.spatial_aggregation.description"
            ]
         },
         "type":{
            "description":"Type of element generated",
            "type":"string",
            "examples":[
               "radio",
               "select",
               "checkbox",
               "input",
               "date",
               "draw"
            ]
         },
         "value":{
            "description":"Value of the default selected option",
            "type":"string",
            "examples":[
               "countries"
            ]
         },
         "options":{
            "description":"array of values of the element",
            "type":"array",
            "items":{
               "type":"object",
               "required":[
                  "value",
                  "text",
                  "description"
               ],
               "properties":{
                  "value":{
                     "description":"value of the option",
                     "type":"string",
                     "examples":[
                        "countries"
                     ]
                  },
                  "text":{
                     "description":"text of the option, ref. to the i18n file",
                     "type":"string",
                     "examples":[
                        "selector.spatial_aggregation.countries.label"
                     ]
                  },
                  "description":{
                     "description":"tooltip of the option,ref. to the i18n file",
                     "type":"string",
                     "examples":[
                           "selector.spatial_aggregation.countries.description"
                     ]
                  },
                  "enabled":{
                           "description":"array of conditions to enable this option. If this
parameter is not filled, the option will be always enabled. All the conditions must be true to
enable it.",
                     "type":"array",
                     "items":{
                        "type":"object",
                        "required":[
                           "selector",
                           "value"
                        ],
                        "properties":{
                           "selector":{
                              "description":"Id of an element of the current menu ",
                              "type":"string",
                              "examples":[
                                 "spatial_aggregation"
                              ]
                           },
                           "value":{
                              "description":"Array of possible values of that element that has
to be selected to consider “true” this condition",
                              "type":"array",
                              "items":{
                                 "type":"string",
                                 "examples":[
                                    "countries"
                                 ]
                              }
                           }
                        }
                     }
                  },
                  "visible":{
                        "description":"array of conditions to set visible this option. If this
parameter is not filled, the option will be always visible. All the conditions must be true to
set visible it.",
                     "type":"array",
                     "items":{
                        "type":"object",
                        "required":[
                           "selector",
                           "value"
                        ],
                        "properties":{
                           "selector":{
                              "description":"Id of an element of the current menu ",
                              "type":"string",
                              "examples":[
                                 "spatial_aggregation"
                              ]
                           },
                           "value":{
                              "description":"Array of possible values of that element that has
to be selected to consider “true” this condition",
                              "type":"array",
                              "items":{
                                 "type":"string",
                                 "examples":[
                                    "countries"
                                 ]
                              }
                           }
                        }
                     }
                  }
               }
            }
         },
         "visible":{
             "description":"Array of conditions to set visible this item. If this parameter is
not filled, the item will be always visible. All the conditions must be true to set visible
it.",
            "type":"array",
            "items":{
               "type":"object",
               "required":[
                  "selector",
                  "value"
               ],
               "properties":{
                  "selector":{
                     "description":"Id of an element of the current menu ",
                     "type":"string",
                     "examples":[
                        "spatial_aggregation"
                     ]
                  },
                  "value":{
                        "description":"Array of possible values of that element that has to be
selected to consider “true” this condition",
                     "type":"array",
                     "items":{
                        "type":"string",
                        "examples":[
                           "countries"
                        ]
                     }
                  }
               }
            }
         },
         "breadcrumb":{
             "description":"If there is a “breadcrumb control” on the map, this parameter sets
if this menu option is shown on the control or not.",
            "type":"boolean",
            "default":false,
            "examples":[
               false,
               true
            ]
         },
         "minChecked":{
              "description":"(only if type == checkbox) in a group of checkboxes, it sets the
minimum number of checkboxes that must be checked. If the user has checked that number of
checkboxes, they are disabled, so the user cannot uncheck anymore, but he can select the
unchecked ones.",
            "type":"integer",
            "examples":[
               1,
               2,
               3
            ]
         },
         "classCheck":{
              "description":"(only if type == checkbox)  possible values: check / switch. To
decide the kind of checkbox to draw",
            "type":"string",
            "examples":[
               "check",
               "switch"
            ]
         }
      }
   }
}
```

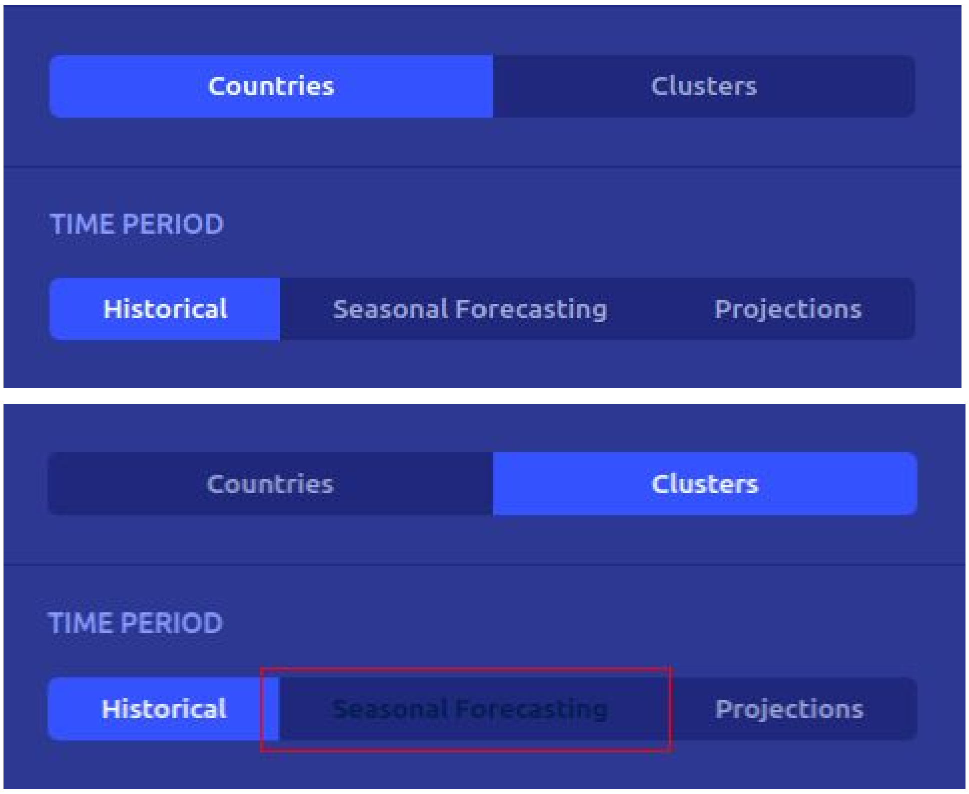
Examples of different types of menu items (selected by “type” parameter):

Type **radio**

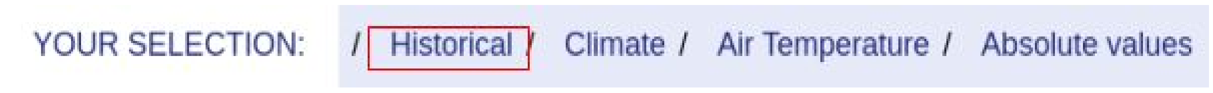
It creates a HTML radio group.

Example 1: 2 menu elements of type “radio”:

```
{
        "id": "spatial_aggregation",
        "label": "selector.spatial_aggregation.label",
        "description": "selector.spatial_aggregation.description",
        "type": "radio",
        "value": "countries",
        "options": [
                {
                        "value": "countries",
                        "text": "selector.spatial_aggregation.countries.label",
                        "description": "selector.spatial_aggregation.countries.description"
                },
                {
                        "value": "clusters",
                        "text": "selector.spatial_aggregation.clusters.label",
                        "description": "selector.spatial_aggregation.clusters.description"
                }
        ],
        "breadcrumb": false
},
{
        "id": "model_type",
        "label": "selector.model_type.label",
        "description": "selector.model_type.description",
        "type": "radio",
        "value": "historical",
        "options": [
                {
                        "value": "historical",
                        "text": "selector.model_type.historical.label",
                        "description": "selector.model_type.historical.description"
                },
                {
                        "value": "seasonal",
                        "text": "selector.model_type.seasonal.label",
                        "description": "selector.model_type.seasonal.description",
                        "enabled": [
                                {
                                        "selector": "spatial_aggregation",
                                        "value": [
                                                "countries"
                                                ]
                                }
                                ]
                        },
                {
                        "value": "projections",
                        "text": "selector.model_type.projections.label",
                        "description": "selector.model_type.projections.description"
                }
                ],
        "breadcrumb": true
}
```



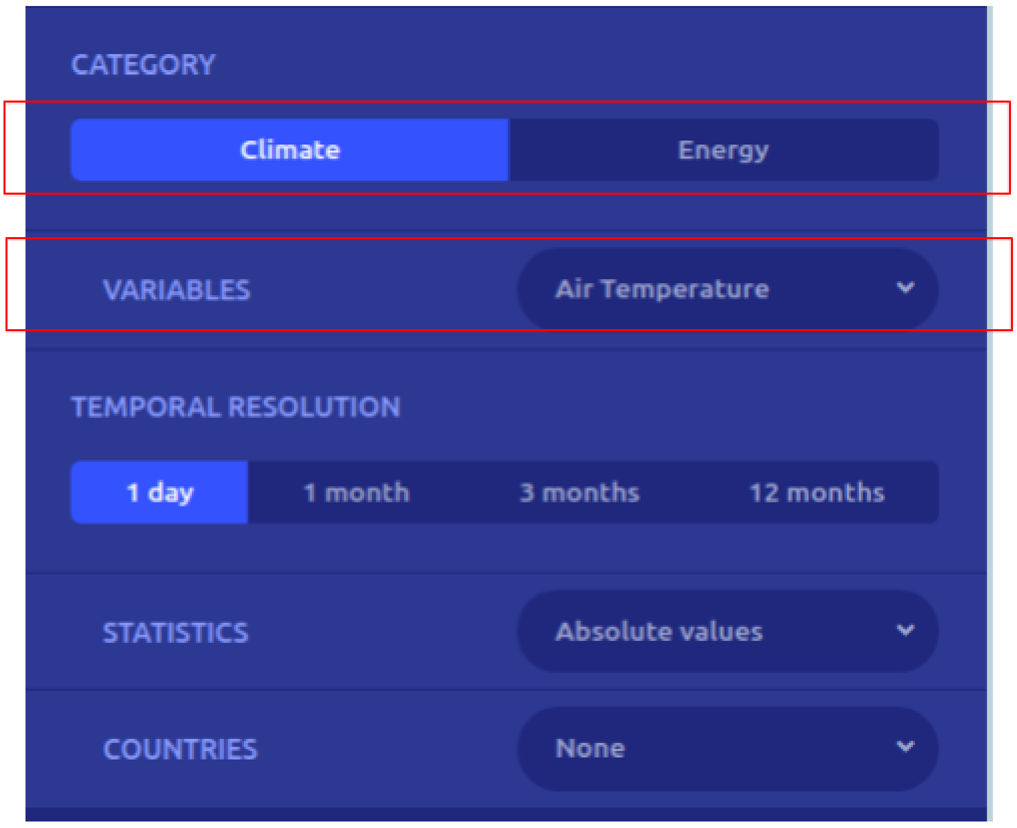
In the second capture, "Seasonal Forecasting" is disabled because "Countries" is not selected.



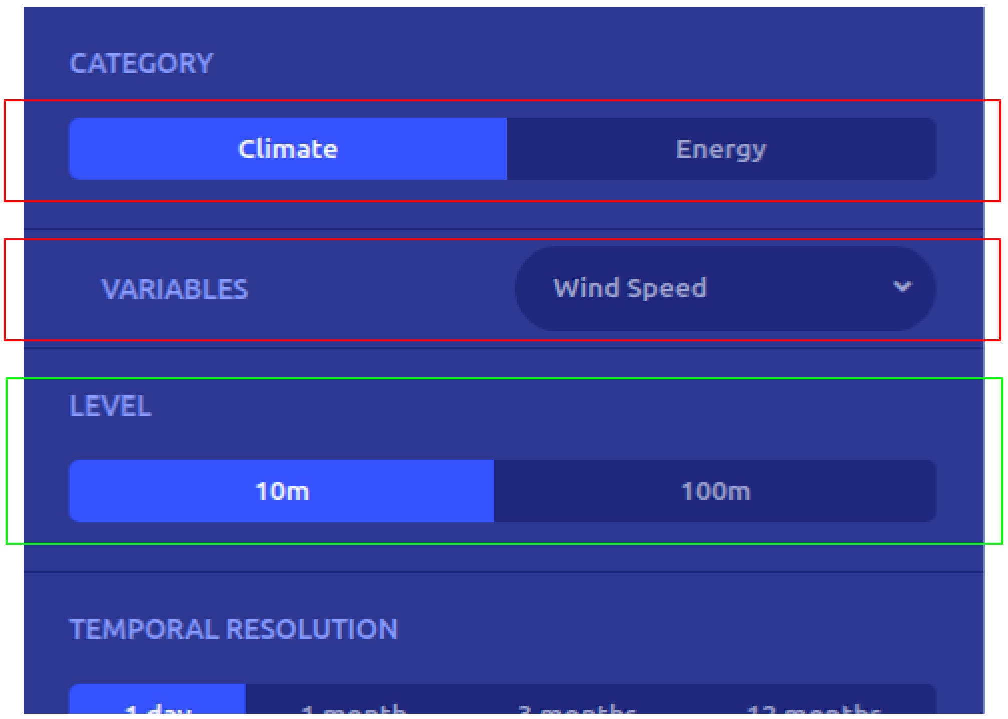
In the breadcrumb-control the value of the first group is not shown, but it is show the value of the second one; it is because the "breadcrumb" parameter.

Example 2: 1 menu element of type "radio" only visible with some options

```
{
        "id": "level",
        "label": "selector.level.label",
        "description": "selector.level.description",
        "type": "radio",
        "value": "ten",
        "visible": [
                {
                        "selector": "category",
                        "value": [
                                "climate"
                        ]
                },
                {
                        "selector": "climate_variable",
                        "value": [
                                "wind_speed"
                        ]
                }
        ],
        "options": [
                {
                        "value": "ten",
                        "text": "selector.level.ten.label",
                        "description": "selector.level.ten.description"
                },
                {
                        "value": "one_hundred",
                        "text": "selector.level.one_hundred.label",
                        "description": "selector.level.one_hundred.description"
                }
        ],
        "breadcrumb": true
}
```



In this capture “level menu item” is not visible because the option “wind_speed” is not selected.



In this capture “level menu item” is visible because the option “wind_speed” is selected.

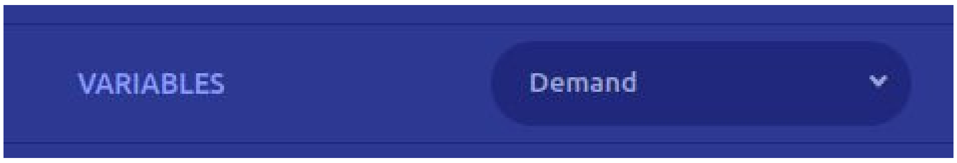
Type **select**

It creates a dropdown of options using [ngx-select-ex.]()

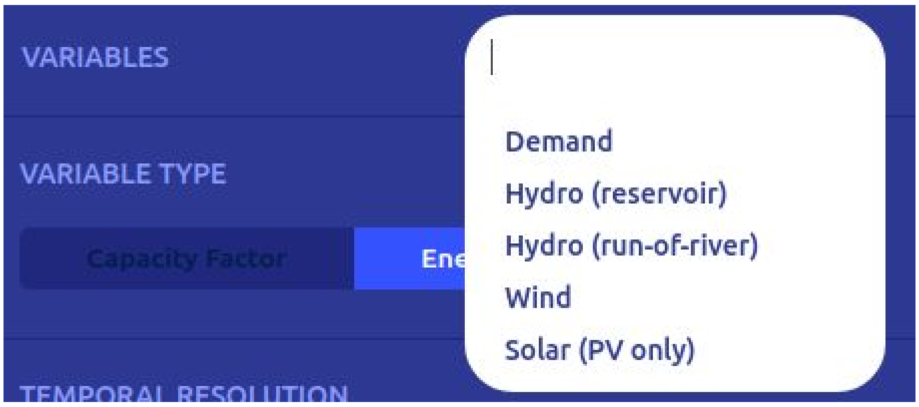
Example 1: Menu element of type “select”

```
{
        "id": "energy_variable",
        "label": "selector.energy_variable.label",
        "description": "selector.energy_variable.description",
        "type": "select",
        "value": "demand",
        "visible": [
                        {
                                "selector": "category",
                                "value": [
                                        "energy"
                                ]
                        }
                ],
        "options": [
                {
                        "value": "demand",
                        "text": "selector.energy_variable.demand.label",
                        "description": "selector.energy_variable.demand.description"
                },
                {
                        "value": "hydro_reservoir",
                        "text": "selector.energy_variable.hydro_reservoir.label",
                        "description": "selector.energy_variable.hydro_reservoir.description"
                },
                {
```

```
                "value": "hydro_run_of_river",
                "text": "selector.energy_variable.hydro_run_of_river.label",
                "description": "selector.energy_variable.hydro_run_of_river.description
            },
            {
                "value": "wind",
                "text": "selector.energy_variable.wind.label",
                "description": "selector.energy_variable.wind.description"
            },
            {
                "value": "solar",
                "text": "selector.energy_variable.solar.label",
                "description": "selector.energy_variable.solar.description"
            }
        ],
        "breadcrumb": true
}
```



Select menu item collapsed

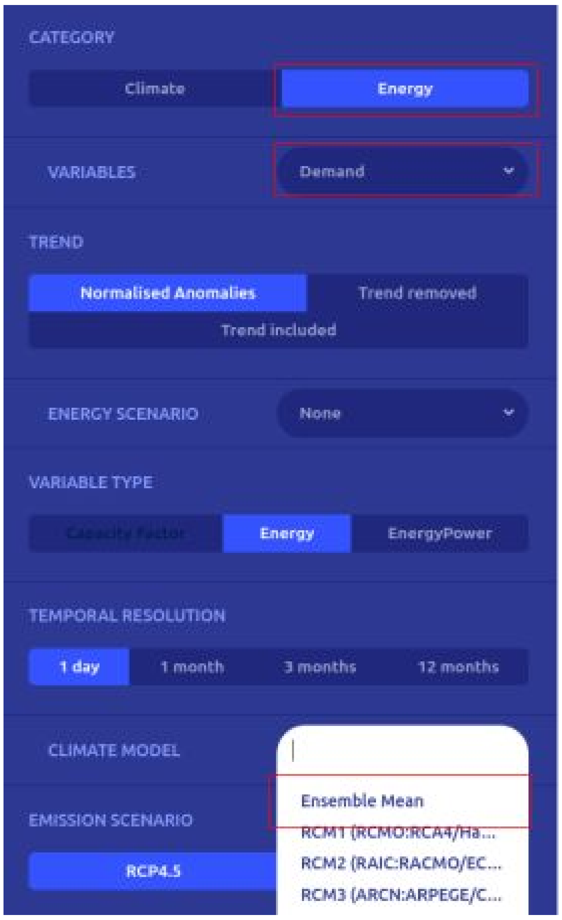


Select menu item expanded

Example 2: Menu element of type select with an option not always visible.

```
{
        "id": "climate_model_energy",
        "label": "selector.climate_model.label",
        "description": "selector.climate_model.description",
        "type": "select",
        "value": "rcm1",
        "visible": [
            {
                "selector": "model_type",
                "value": [
                    "projections"
                ]
            },
            {
                "selector": "category",
                "value": [
                    "energy"
```

```
                        ]
                }
        ],
        "options": [
                {
                        "value": "ensemble_mean",
                        "text": "selector.climate_model.ensemble_mean.label",
                        "description": "selector.climate_model.ensemble_mean.description",
                        "visible": [
                                {
                                        "selector": "energy_variable",
                                        "value": [
                                                "demand",
                                                "wind",
                                                "solar"
                                        ]
                                }
                        ],
                        "enabled": [
                                {
                                        "selector": "temporal_resolution",
                                        "value": [
                                                "yearly"
                                        ]
                                }
                        ]
                },
…
}
```


CATEGORY: Climate [✓ Energy]

VARIABLES: Demand

TREND: [✓ Normalised Anomalies] Trend removed Trend included

ENERGY SCENARIO: None

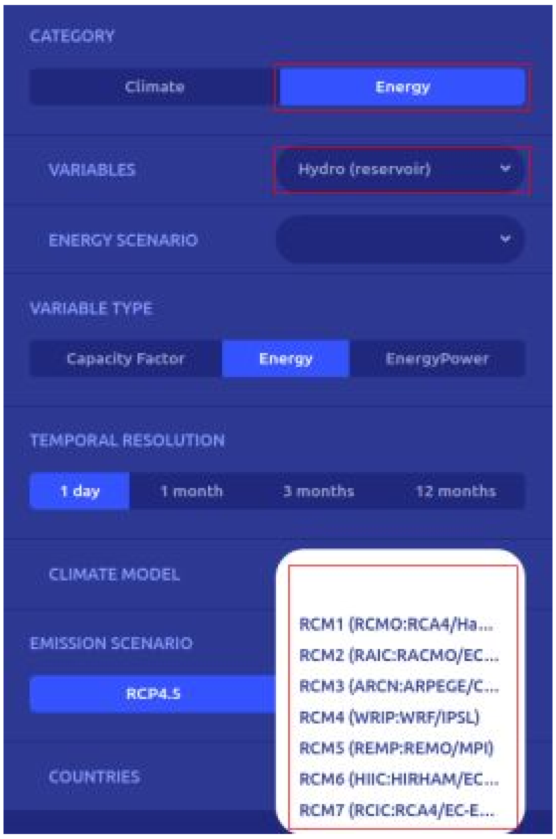
VARIABLE TYPE: Capacity Factor [✓ Energy] EnergyPower

TEMPORAL RESOLUTION: [✓ 1 day] 1 month 3 months 12 months

CLIMATE MODEL:
- Ensemble Mean
- RCM1 (RCMO:RCA4/Ha...
- RCM2 (RAIC:RACMO/EC...
- RCM3 (ARCN:ARPEGE/C...

EMISSION SCENARIO: [✓ RCP4.5]

The “demand” is selected, so the “ensemble mean” option is visible



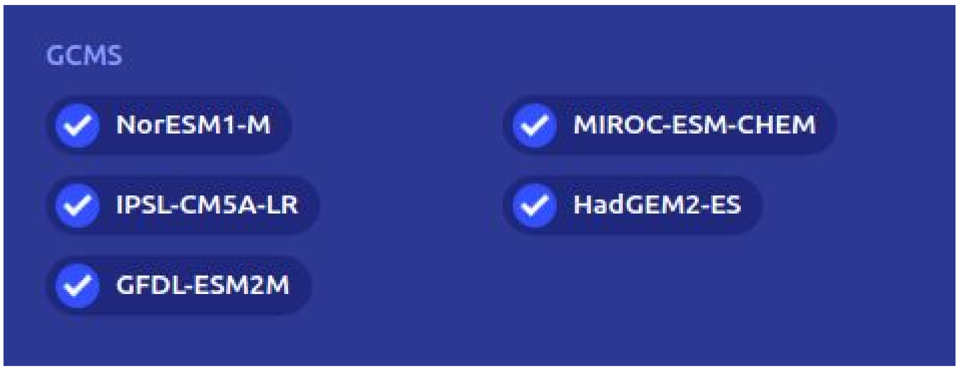
“Demand”, “wind”, “solar” are not selected, so the “ensemble mean” option is not visible

Type **checkbox**

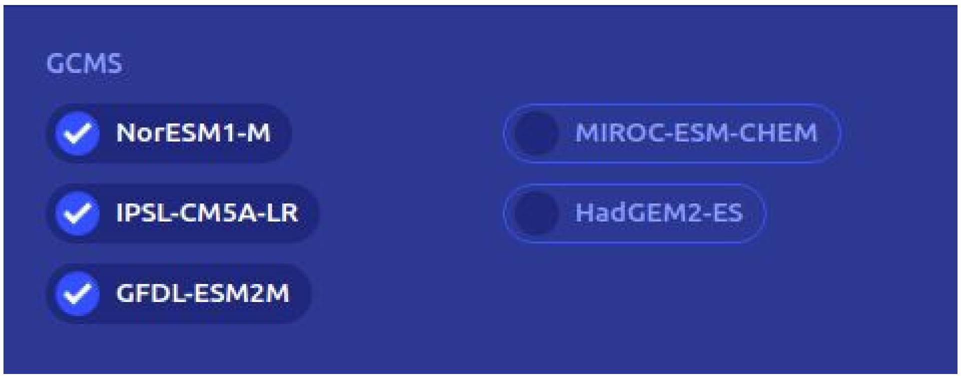
It creates a group of HTML checkboxes. There are 2 types of checkboxes: “check” and “switch”; the type is set with the parameter “classCheck”.

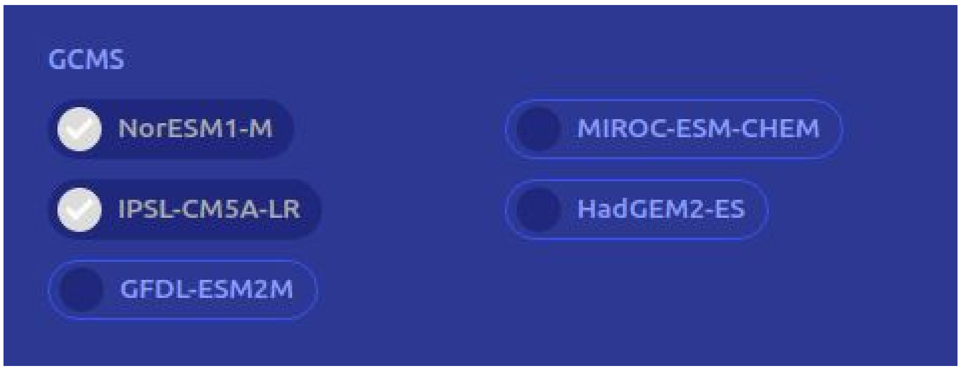
Example 1: Menu element of type “checkbox”, classCheck = “check”

```
{
                "id": "gcm",
                "label": "edge.selector.gcm.label",
                "description": "edge.selector.gcm.description",
                "type": "checkbox",
                "minchecked": 2,
                "classCheck":"check",
                "value": "noresm1",
                "visible": [
                        {
                                "selector": "model_type",
                                "value": [
                                        "projections"
                                ]
                        }
                ],
                "options": [
                        {
                                "value": "noresm1",
                                "checked": true,
                                "text": "edge.selector.gcm.noresm1.label",
                                "description": "edge.selector.noresm1.description"
                        },
                        {
                                "value": "miroc",
                                "checked": true,
                                "text": "edge.selector.gcm.miroc.label",
                                "description": "edge.selector.miroc.description"
                        },
                        {
                                "value": "ipsl",
                                "checked": true,
                                "text": "edge.selector.gcm.ipsl.label",
                                "description": "edge.selector.ipsl.description"
                        },
                        {
                                "value": "hadgem2",
                                "checked": true,
                                "text": "edge.selector.gcm.hadgem2.label",
                                "description": "edge.selector.hadgem2.description"
                        },
                        {
                                "value": "gfdl",
                                "checked": true,
                                "text": "edge.selector.gcm.gfdl.label",
                                "description": "edge.selector.gfdl.description"
                        }
                ],
                "breadcrumb": true
        }
```



All of the options are selected because of the parameters “checked” are set to ‘true’

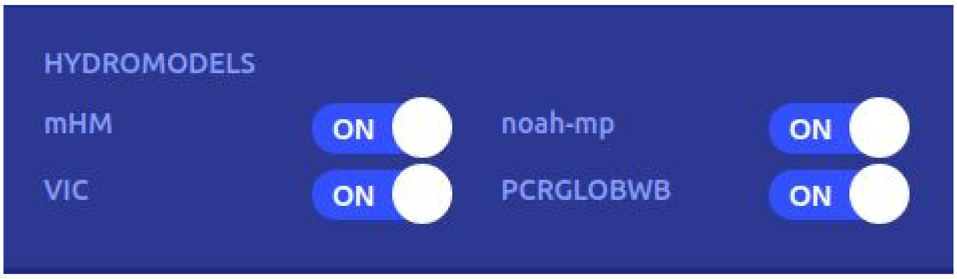




In the 3rd capture, the 2 checkboxes are disabled because the minchecked parameter.

Example 2: Menu element of type “checkbox”, classCheck = “switch”

```
{
        "id": "hydromodels",
        "label": "edge.selector.hydromodels.label",
        "description": "edge.selector.hydromodels.description",
        "type": "checkbox",
        "minchecked": 1,
        "classCheck":"switch",
        "value": "mhm",
        "visible": [
                {
                        "selector": "model_type",
                        "value": [
                                "projections",
                                "seasonal"
                        ]
                }
        ],
        "options": [
                {
                        "value": "mhm",
                        "checked": true,
                        "text": "edge.selector.hydromodels.mhm.label",
                        "description": "edge.selector.hydromodels.mhm.description",
                        "enabled": []
                },
                {
                        "value": "noahmp",
                        "checked": true,
				"text": "edge.selector.hydromodels.noahmp.label",
				"description": "edge.selector.hydromodels.noahmp.description",
				"enabled": []
			},
			{
				"value": "vic",
				"checked": true,
				"text": "edge.selector.hydromodels.vic.label",
				"description": "edge.selector.hydromodels.vic.description",
				"enabled": []
			},
			{
				"value": "pcrglobwb",
				"checked": true,
				"text": "edge.selector.hydromodels.pcrglobwb.label",
				"description": "edge.selector.hydromodels.pcrglobwb.description",
				"enabled": []
			}
		],
		"breadcrumb": true
}
```



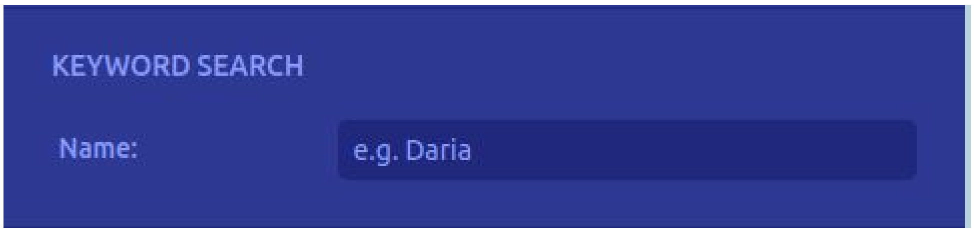
Type **input**

It creates a HTML text input with a label.

- The “label” text is set in the parameter options>text
- The “placeholder” text is set in the parameter options>description

Example: Menu element of type “input”

```
{
		"id": "keyword_search",
		"label": "wisc.selector.keyword_search.label",
		"description": "wisc.selector.keyword_search.description",
		"type": "input",
		"value": "",
		"visible": [
			{
				"selector": "dataset",
				"value": [
					"storm_track",
					"storm_footprint"
				]
			}
		],
		"options": [
			{
				"value": "name",
                        "text": "wisc.selector.keyword_search.name.label",
                        "description": "wisc.selector.keyword_search.name.description"
                }
        ],
        "breadcrumb": false
}
```

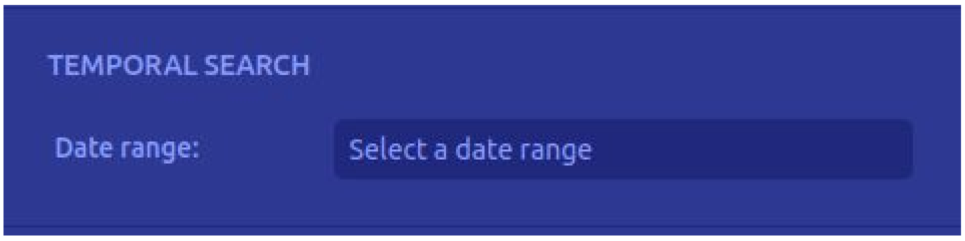


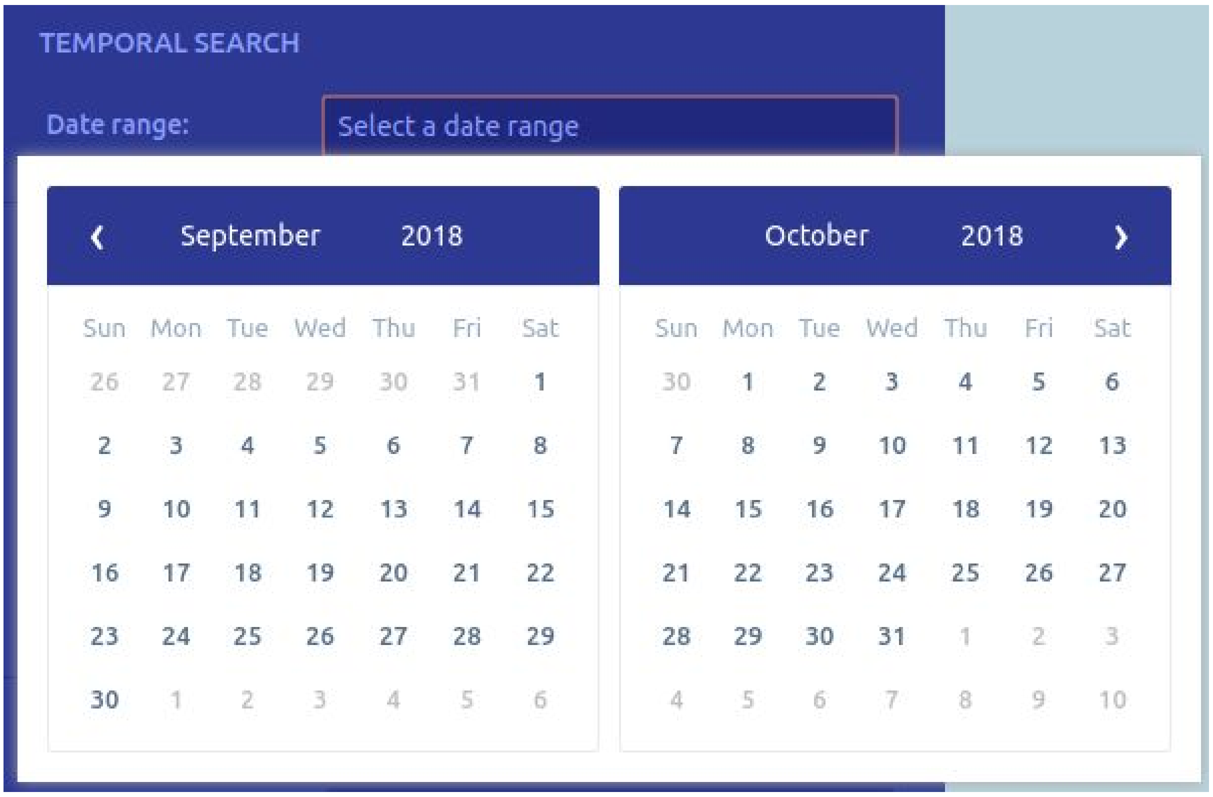
Type **date**

It creates a date menu item (range) using ngx-bootstrap Datepicker.

Example: Menu element of type "date"

```
        {
                "id": "temporal_search",
                "label": "wisc.selector.temporal_search.label",
                "description": "wisc.selector.temporal_search.description",
                "type": "date",
                "value": "",
                "visible": [
                        {
                                "selector": "dataset",
                                "value": [
                                        "storm_track",
                                        "storm_footprint"
                                ]
                        }
                ],
                "options": [],
                "breadcrumb": false
        }
```



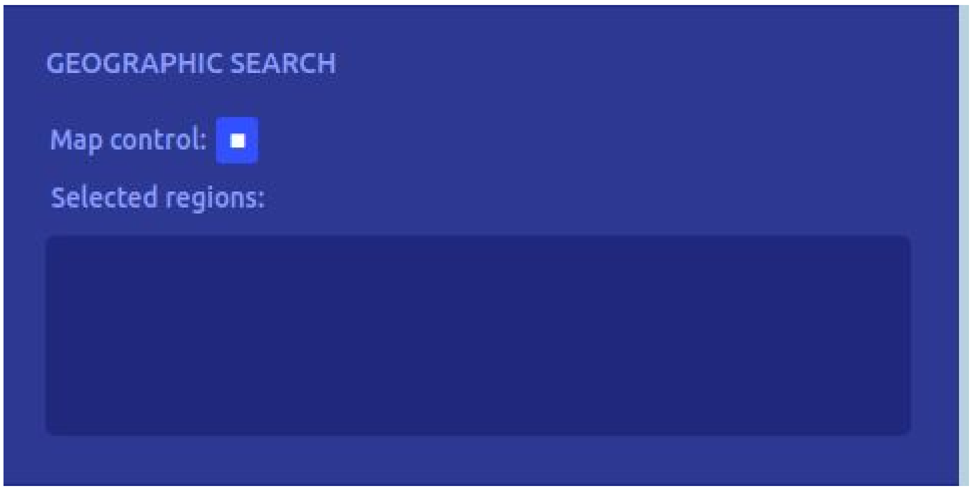


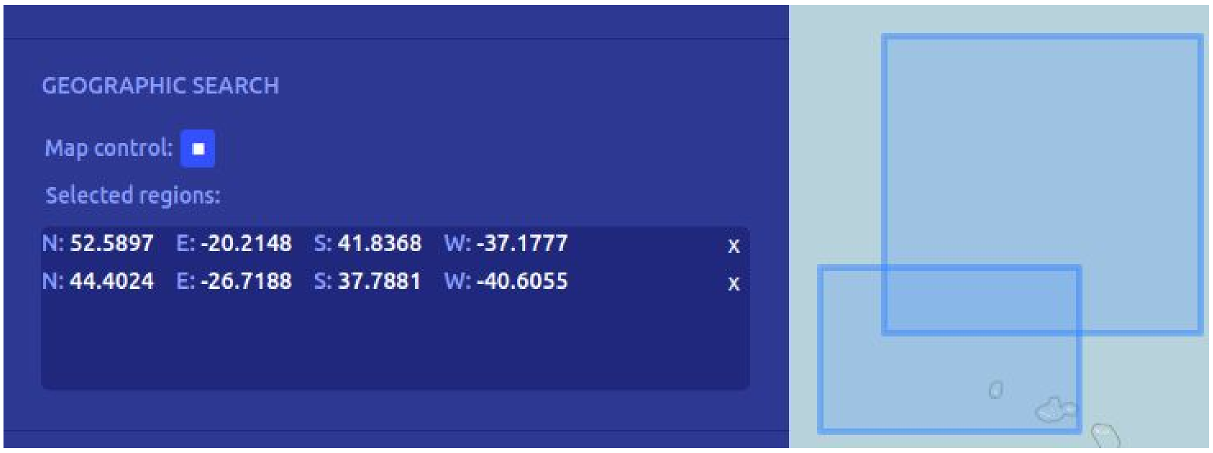
Type **draw**

It creates a menu item used to draw areas on the map. The coordinates of the areas are shown on a container. The areas can be deleted clicking on the X of the row. (This menu type is so specific because it was created originally for WISC demonstrator).

Example: Menu element of type “draw”

```
{
        "id": "geographical_search",
        "label": "wisc.selector.geographical_search.label",
        "description": "wisc.selector.geographical_search.description",
        "type": "draw",
        "value": "",
        "visible": [
                {
                        "selector": "dataset",
                        "value": [
                                "storm_track",
                                "storm_footprint"
                        ]
                }
        ],
        "options": [
                {
                        "value": "control",
                        "text":"wisc.selector.geographical_search.control.label",
                        "description": "wisc.selector.geographical_search.control.description"
                }
        ],
        "breadcrumb": false
}
```





```
N: 52.5897   E: -20.2148   S: 41.8368   W: -37.1777   x
N: 44.4024   E: -26.7188   S: 37.7881   W: -40.6055   x
```

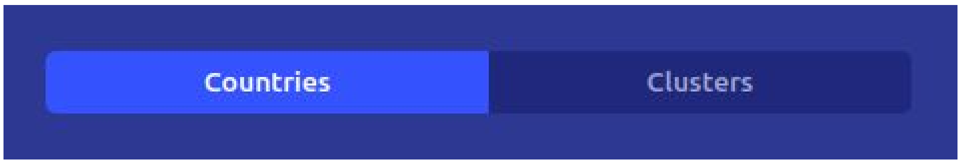
Translations of the menu items

To set the "translations" of the menu items, the library ngx-translate for Angular is used, so another JSON file must be created (ex: i18n/en.json). This file will be used for all the demonstrators of the application (in case of multiple demonstrators).

Each literal of the configuration file can be referenced in the translation file.

The translation is done with the pipe "translate".

Example 1: radio group without title.



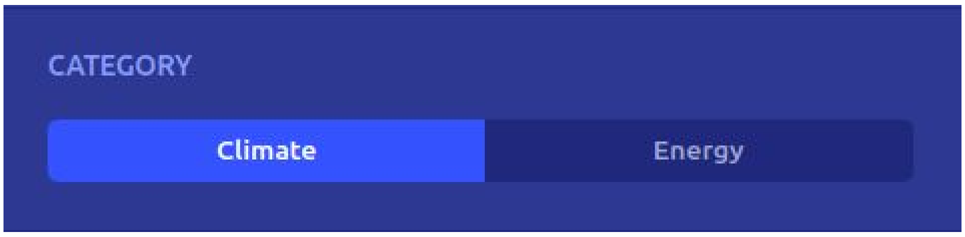
Menu configuration file:

```
{
        "id": "spatial_aggregation",
        "label": "selector.spatial_aggregation.label",
        "description": "selector.spatial_aggregation.description",
        "type": "radio",
        "value": "countries",
        "options": [
                {
                        "value": "countries",
                        "text": "selector.spatial_aggregation.countries.label",
                        "description": "selector.spatial_aggregation.countries.description"
                },
                {
                        "value": "clusters",
                        "text": "selector.spatial_aggregation.clusters.label",
                        "description": "selector.spatial_aggregation.clusters.description"
                }
        ],
        "breadcrumb": false
}
```

Translations file:

```
{
        "selector": {
                "spatial_aggregation": {
                        "label": "",
                        "description": "",
                        "clusters": {
                                "label": "Clusters",
                                "description": "96 e-Highway 2050 clusters"
                        },
                        "countries": {
                                "label": "Countries",
                                "description": "33 European countries"
                        }
                },
…
}
```

Example 2: radio group with title.



Menu configuration file:
```
{
        "id": "category",
        "label": "selector.category.label",
        "description": "selector.category.description",
        "type": "radio",
        "value": "climate",
        "options": [
                ...
        ],
        "breadcrumb": true
}
```

Translations file:
```
{
        "selector": {
                ...
                "category": {
                        "label": "Category",
                        "description": "",
                        "climate": {
                                "label": "Climate",
                                "description": ""
                        },
                        "energy": {
                                "label": "Energy",
                                "description": ""
                        }
                }
        …
}
```

Initialize the menu

To get the menu initialized in the new demonstrator, a *div* element must be placed in the file *demonstrator*-demonstrator.component.html:

```
<div  *ngIf="menu  else  loading;"  submenu-demons  [menus]="menu"  [menusToPaint]="[menu[0]]"
[firstTime]="true" [parameters]="parameters"></div>
```

Filter configuration

In order to configure which workflow of the toolbox has to be executed, a JSON file must be written. This file will be configured depending on how the workflows in the toolbox are defined. There will be a JSON file for each demonstrator.

Schema of the JSON file

```
  {
    "definitions": {},
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "http://example.com/root.json",
    "type": "object",
    "title": "The Root Schema",
    "required": [
      "filterGroups",
      "filters"
   ],
   "properties": {
     "filterGroups": {
       "$id": "#/properties/filterGroups",
       "type": "object",
       "title": "The Filtergroups Schema",
       "required": [
         "precipitation"
       ],
       "type": {
           "properties": {
             "filters": {
               "$id": "#/properties/filterGroups/properties/precipitation/properties/filters",
               "type": "object",
               "title": "The Filters Schema",
               "required": []
               }
             },
             "params": {
               "$id": "#/properties/filterGroups/properties/precipitation/properties/params",
               "type": "array",
               "title": "The Params Schema",
               "items": {
                                                                                          "$id":
"#/properties/filterGroups/properties/precipitation/properties/params/items",
                 "type": "string",
                 "title": "The Items Schema",
                 "default": "",
                 "examples": [
                   "indicator_precipitation",
                   "time_period",
                   "resolution",
                   "emission_scenario",
                   "model_type"
                 ],
                 "pattern": "^(.*)$"
               }
             }
           }
       }
```

```
},
"filters": {
  "$id": "#/properties/filters",
  "type": "array",
  "title": "The Filters Schema",
  "items": {
    "$id": "#/properties/filters/items",
    "type": "object",
    "title": "The Items Schema",
    "required": [
      "all",
      "any",
      "optional",
      "workflow"
    ],
    "properties": {
      "all": {
        "$id": "#/properties/filters/items/properties/all",
        "type": "array",
        "title": "The All Schema",
        "items": {
          "$id": "#/properties/filters/items/properties/all/items",
          "type": "string",
          "title": "The Items Schema",
          "default": "",
          "examples": [
            "precipitation"
          ],
          "pattern": "^(.*)$"
       }
     },
     "any": {
       "$id": "#/properties/filters/items/properties/any",
       "type": "array",
       "title": "The Any Schema",
       "items": {
         "$id": "#/properties/filters/items/properties/any/items",
         "type": "array",
         "title": "The Items Schema"
       }
     },
     "optional": {
       "$id": "#/properties/filters/items/properties/optional",
       "type": "array",
       "title": "The Optional Schema",
       "items": {
         "$id":"#/properties/filters/items/properties/optional/items",
         "type": "string",
         "title": "The Items Schema",
         "default": "",
         "examples": [
           "ensemble_range"
         ],
         "pattern": "^(.*)$"
       }
     },
     "workflow": {
       "$id": "#/properties/filters/items/properties/workflow",
       "type": "string",
       "title": "The Workflow Schema",
       "default": "",
       "examples": [
              "swicca-precipitation-linechart"
            ],
            "pattern": "^(.*)$"
          }
        }
      }
    }
  }
}
```

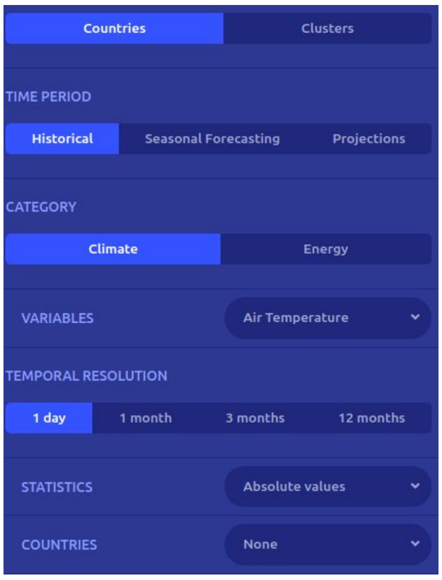
Explanation

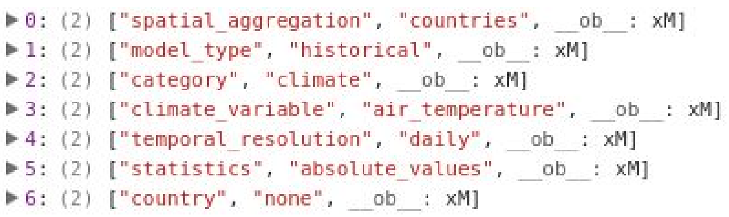
The following example shows how this functionality works, in order to give the developer the information to create a new configuration file for a new demonstrator.

The JSON file has 2 elements:

- **filters**: each element of the array defines one workflow, and has the next 4 elements:
  - **all**: array of elements, each one of them will fit with the identifier of a subelement of the filterGroups element.
  - **any**: array of array of elements, each one of them will fit with the identifier of a subelement of the filterGroups element.
  - **optional**: array of elements, each one of them will fit with the identifier of a subelement of the filterGroups element.
  - **workflow**: code of the workflow
- **filterGroups**: each element will have an identifier (the elements of *filters*), and as value, 2 elements:
  - **filters**: a tuple key-value that must fit with a selected option of the menu.
  - **params**: an array of keys that must fit with a selected option of the menu.

When we talk about the options of the menu, we refer to an array generated when the menu of a demonstrator is used. For example, for the ECEM demonstrator, a possible value for this array is:

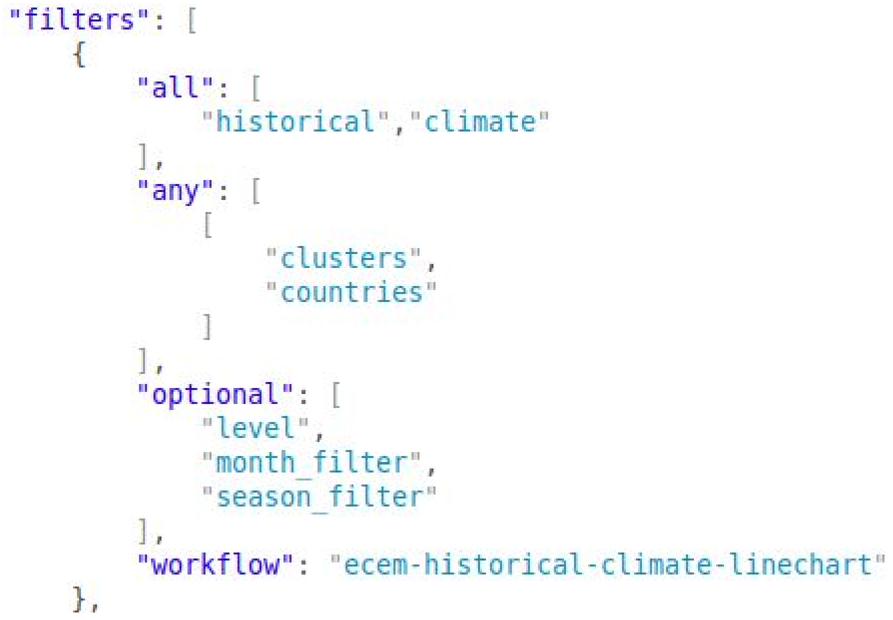




```
▶0: (2) ["spatial_aggregation", "countries", __ob__: xM]
▶1: (2) ["model_type", "historical", __ob__: xM]
▶2: (2) ["category", "climate", __ob__: xM]
▶3: (2) ["climate_variable", "air_temperature", __ob__: xM]
▶4: (2) ["temporal_resolution", "daily", __ob__: xM]
▶5: (2) ["statistics", "absolute_values", __ob__: xM]
▶6: (2) ["country", "none", __ob__: xM]
```

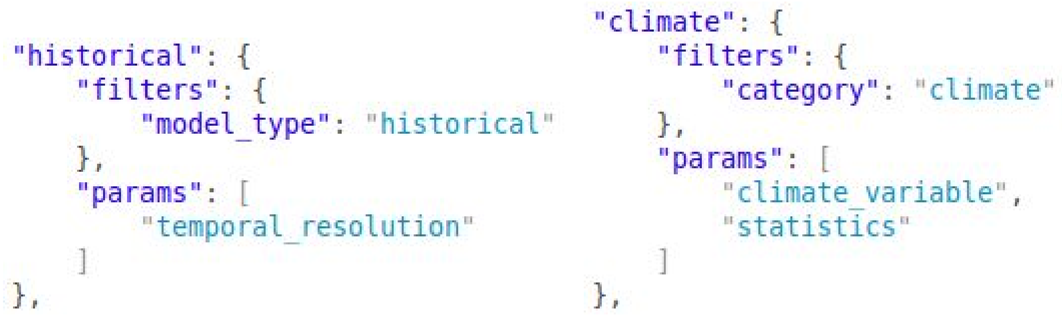
The goal here is to determine, depending on these array values, which **workflow** will be called and which **parameters** will be sent to it to generate / get the data from the Toolbox. We will see it with an example, with the ECEM demonstrator:

For this demonstrator a workflow "*ecem-historical-climate-linechart*" has been defined. This workflow will be called for the *historical* and *climate* selections in the menu, and will be sent the rest of the selections as parameters. In order to do this, we have defined the following *filter* element in the JSON file:



```
"filters": [
    {
        "all": [
            "historical","climate"
        ],
        "any": [
            [
                "clusters",
                "countries"
            ]
        ],
        "optional": [
            "level",
            "month_filter",
            "season_filter"
        ],
        "workflow": "ecem-historical-climate-linechart"
    },
```

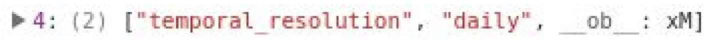
When a user modifies the options of the menu, the logic will search which of these *filter* elements fits with the options. It will loop through the **ALL** elements and search them in the *filterGroups* elements, it will find, in our example, these:



```
"historical": {
    "filters": {
        "model_type": "historical"
    },
    "params": [
        "temporal_resolution"
    ]
},
```

```
"climate": {
    "filters": {
        "category": "climate"
    },
    "params": [
        "climate_variable",
        "statistics"
    ]
},
```

In the first element ("historical"), if "model_type": "historical" exist in the menu options (it exists in our example), the application will save temporarily the options that fits with the "params" elements ("temporal_resolution").



```
▶ 4: (2) ["temporal_resolution", "daily", __ob__: xM]
```

The same with the "climate" element, **all** of the elements must be found, if not, this is not our workflow, and the logic will continue searching. So we can call these parameters (temporal_resolution, climate_variable, statistics), the **mandatory** parameters to be sent to the workflow.

If all of them have been found, we jump to the **ANY** element, and for each array, we search for its elements in *filterGroups,* in our example, we would find for "clusters","countries":

```
"countries": {
    "filters": {
        "spatial_aggregation": "countries"
    },
    "params": [
        "country"
    ]
},
```

```
"clusters": {
    "filters": {
        "spatial_aggregation": "clusters"
    },
    "params": [
        "cluster"
    ]
},
```

But only the “filters” element of the “countries” node will be found in the options:

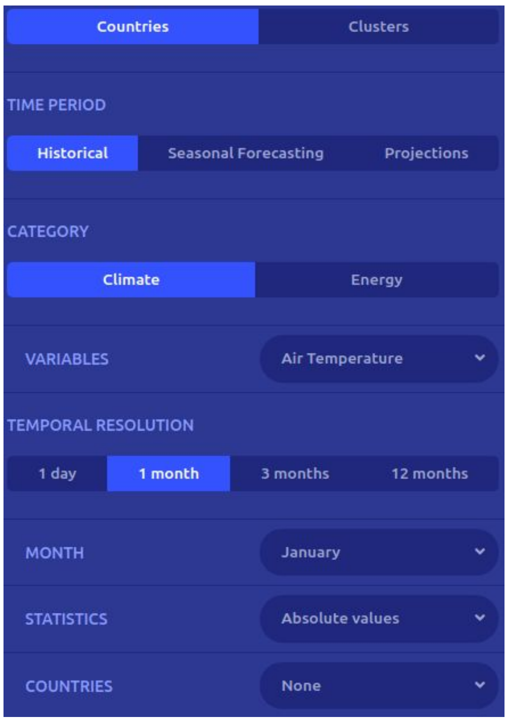
```
▶0: (2) ["spatial_aggregation", "countries", __ob__: xM]
```

And that is the goal of the **any** elements, only **one** must be found, and its “params” elements will be saved temporarily, in this case:

```
▶6: (2) ["country", "none", __ob__: xM]
```

If one of them of each array is found, this means that we have found the workflow.

Then, we jump to the **OPTIONAL** element, this means that none, one or more of these elements can be sent, searching them directly in the options of the menu, in our example, none of them is found. These kind of options are for the menu options that are only visible in some cases, for example “month_filter” is shown when we select “monthly” in “temporal_resolution” but not in “daily”.



```
▶0: (2) ["spatial_aggregation", "countries", __ob__: xM]
▶1: (2) ["model_type", "historical", __ob__: xM]
▶2: (2) ["category", "climate", __ob__: xM]
▶3: (2) ["climate_variable", "air_temperature", __ob__: xM]
▶4: (2) ["temporal_resolution", "monthly", __ob__: xM]
▶5: (2) ["month_filter", "january", __ob__: xM]
▶6: (2) ["statistics", "absolute_values", __ob__: xM]
▶7: (2) ["country", "none", __ob__: xM]
```

So, we already have the **workflow** to be called with its needed **parameters**.

```
▶4: (2) ["temporal_resolution", "daily", __ob__: xM]

▶0: (2) ["spatial_aggregation", "countries", __ob__: xM]

▶3: (2) ["climate_variable", "air_temperature", __ob__: xM]

▶5: (2) ["statistics", "absolute_values", __ob__: xM]

▶6: (2) ["country", "none", __ob__: xM]
```

All this functionality is in the code of each demonstrator and must be copied. This is because there could be demonstrators that have variations in their behaviours.