FORMS

USER'S MANUAL

– rev 2.7 –

Porto, May 2022

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

PWA	Progressive Web Application
WYSIWYG	What you see is what you get

1 Introduction

The system is a full suite for the creation of electronic forms (fig. 1), their management and their use.

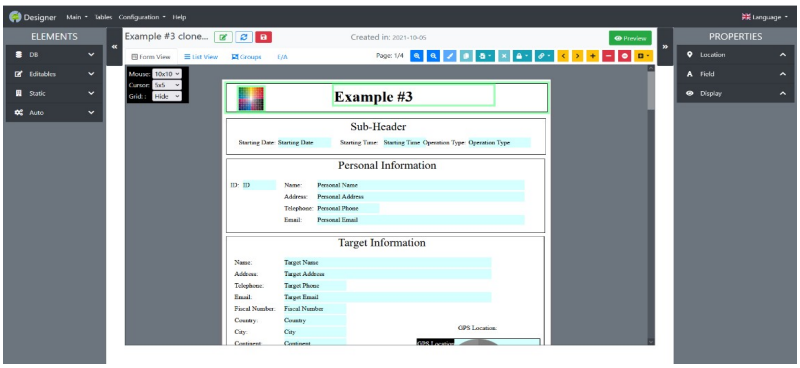

Fig. 1: The form's editor

When filling a form, the user is able to chose between two distinctive views: *Form View* and *List View* (fig. 2). The first is directed to medium / larger devices, such as pc and large tablets) and for those users who feel more comfortable with a printing friendly format. The *List View* is directed towards small / medium devices, such as small tablets and smartphones.

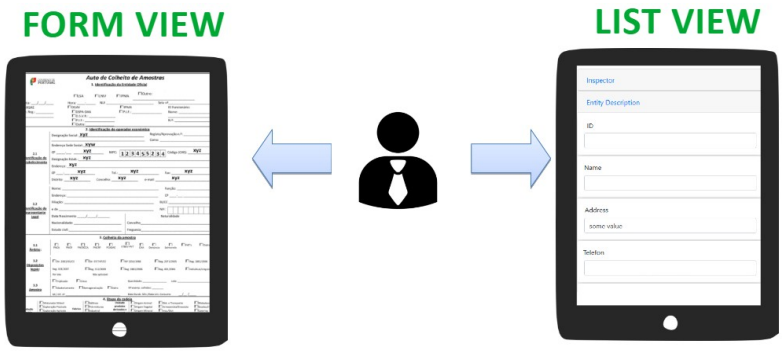

Fig. 2: The views

Apart from the traditional inputs, such as numbers and checkboxes, there are several special ones, like:

- Barcode reader
- GPS
- Drawing
- Signature
- …

The entire suite is compose by seven applications (fig. 3), which are listed and described in table 1.

Fig. 3: Full suite

What	Description
Designer	Editor of forms
Preview	Previews of any form
Forms Manager	Manages all forms, regardless of their state
Operations Manager	Manages all operations, regardless of their state
Operations Editor	Edits any non **COMPLETED** operation
Inspectors App	Creates and manages a users' Operations
Rest API	Available to managers and administrators to manage Operations

Tab. 1: Description of the different applications

2 Designer

The *Designer* or *Editor* is where one can build and edit forms. Fig. 4 presents the main view of the designer.

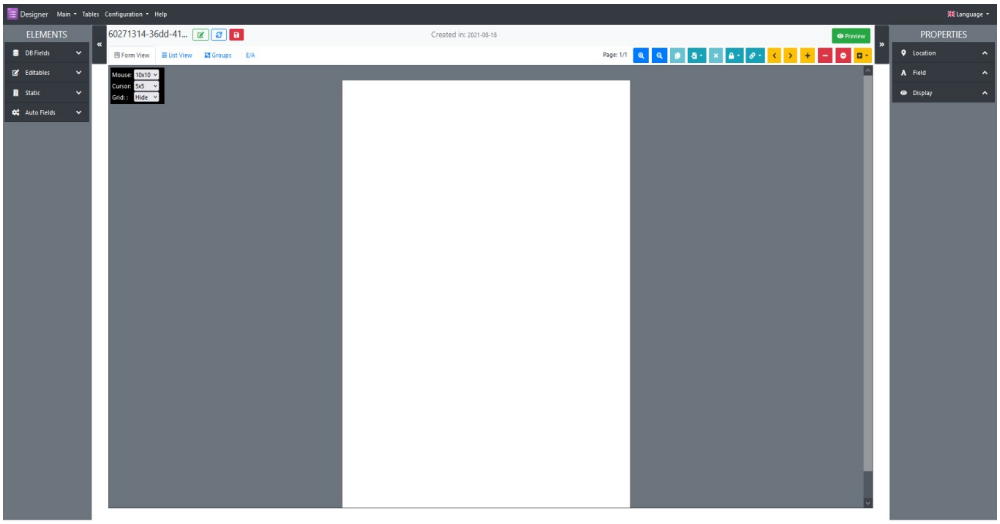

Fig. 4: The designer application

2.1 Main UI

The main window (fig. 5) is composed by:

- A *navbar*, containing the typical options, such as *Open* and *Save*, and also global options and operations, such as the *Auto-Save* dialogue;
- Two collapsible *sidebars*, one containing all elements the user can drag and drop into a page and the other containing the properties of each of those elements;
- A *topbar*, with the indication of the form's name, and buttons for opening the properties dialogue, another to reload the page and a quick-save button, which visibility depends on whether or not there're changes to be saved;
- The *working area*, whose content depends on the selected tab: form view, list view, groups and E/A;
- The *context menu*, whose content also depends on the selected tab.

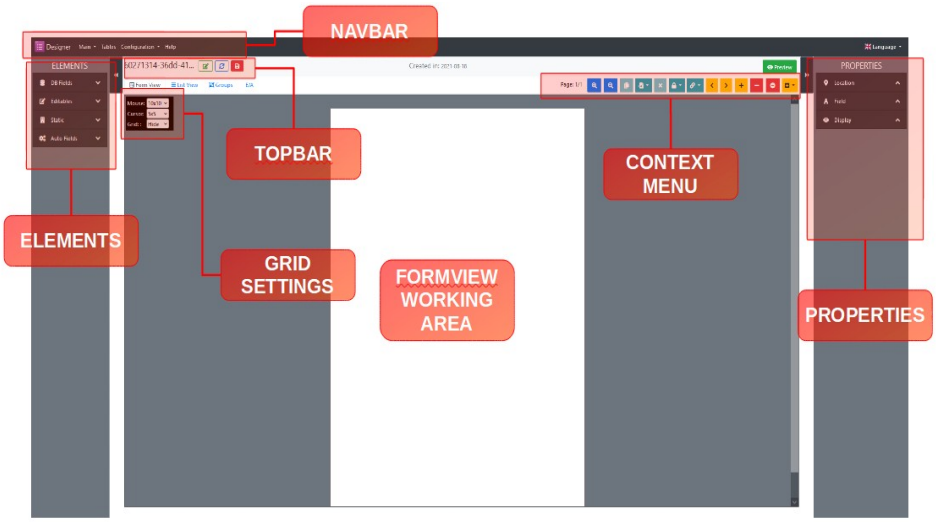

Fig. 5: Designer's main view

2.1.1 Navbar

In table 2 we find a resume of all options available in the page's top menu.

Menu	Option	Description
Main	**New**	Clears everything and start a new form
	Open	Selects and open a form
	Save	Saves the current form
	Properties	Opens the properties dialogue
	Forms Manager	Leaves the designer and opens the forms manager
	Main Page	Leaves the designer and opens the main page
Tables		Opens the tables dialogue, to manage tables
Configuration	**Create Elements from the Database**	Opens a dialogue that allows the selection of database fields in order to create new elements based on those fields
	Auto Save	Opens the auto save dialogue to activate/deactivate the auto-save
Help		Opens the designer's help page

Tab. 2: Main menu options

2.1.1.1 Properties Dialogue

The properties dialogue (fig. 6) allows the user not only to define the form's name and description, but also to get general information of the form. This includes: data of creation, original author, number of elements, and others. There are two ways to access to properties dialogue: the *Main* > *Properties* or the Properties button on the right of the forms's name (fig. 7).

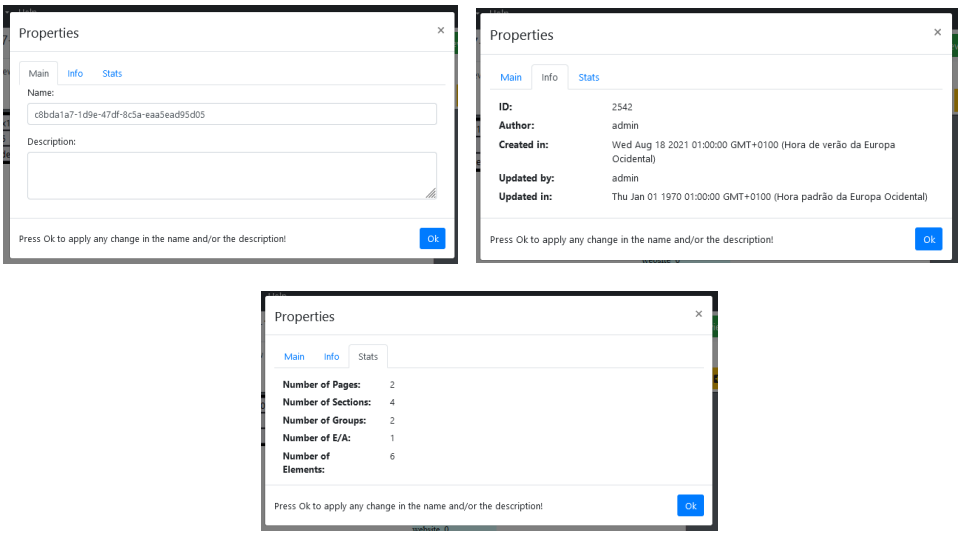

Fig. 6: The Properties dialogue

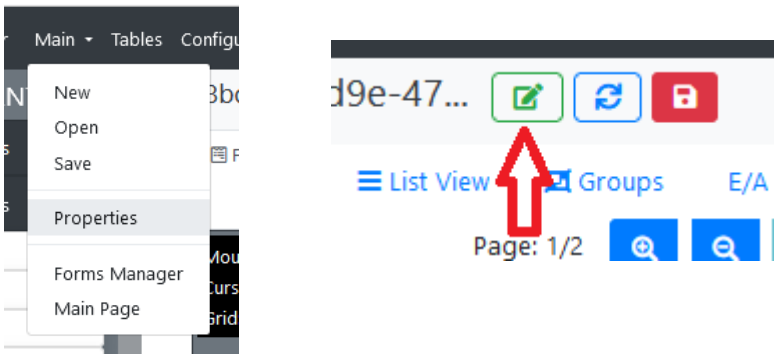

Fig. 7: Access to the Properties dialogue

2.1.1.2 Auto Save Dialogue

To access the auto-save dialogue, one goes to *Configuration* > *Auto Save* (fig. 8).

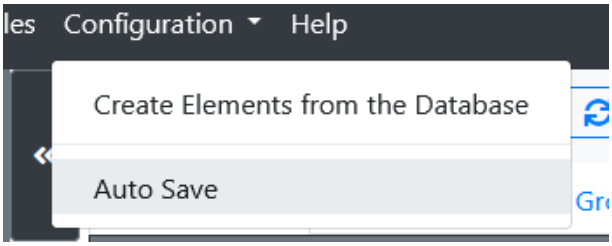

Fig. 8: Selecting the auto-save dialogue

By default, the auto-save feature is disabled. To activate this feature, just check the *Auto Save* box, and define the time interval between saves, as depicted in fig. 9.

Fig. 9: Setting up a new auto-save interval

2.1.1.3 Tables Dialogue

This dialogue (figs. 10 and 11) allows the user to manage tables, which will be used either as data sources for dropdowns / lists elements or for the E/A system to perform tables queries on them.

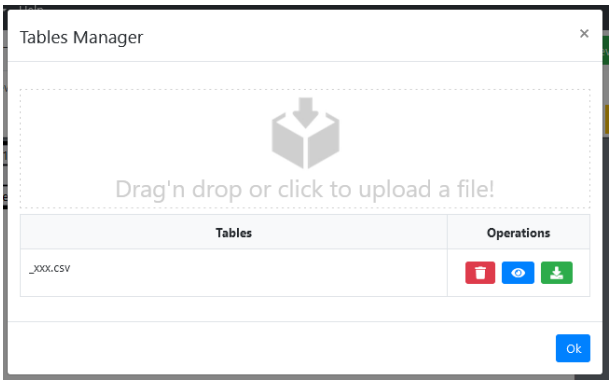

Fig. 10: The Tables Manager dialogue

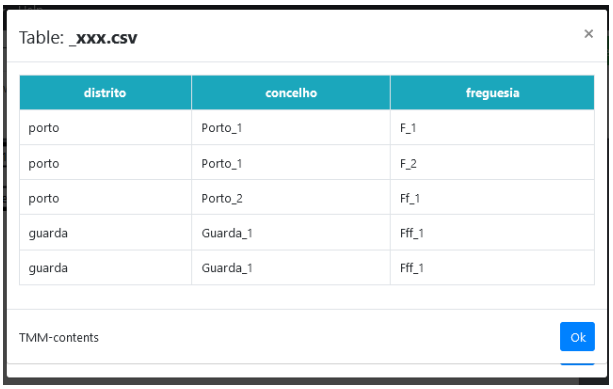

Fig. 11: Table contents preview

Be aware that saving the form right after adding a new table, it will delete that table, since it's not being used anywhere. Therefore it's recommended to add a table only before it's use.

2.1.2 Topbar

The topbar (fig. 12) consists of six elements, which are described in table 3.

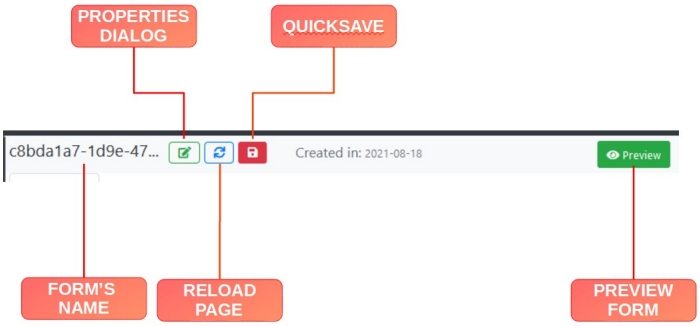

Fig. 12: Designer's top bar

What	Description
Form's Name	Current name of the form. If no name is specified, it will assume a random stream of characters as its name.
Properties Dialogue	Opens the properties dialogue.
Reload Page	Automatically saves any changes and reloads the page/ form.
Quicksave	Saves all current changes.
Preview	Opens a new tab to preview the form.

Tab. 3: Top bar options

2.1.3 Sidebars

There are two collapsible sidebars, one in each side. On the left we have the *Elements* panel, where one can find all elements that can be dragged and set in the forms (fig. 13).

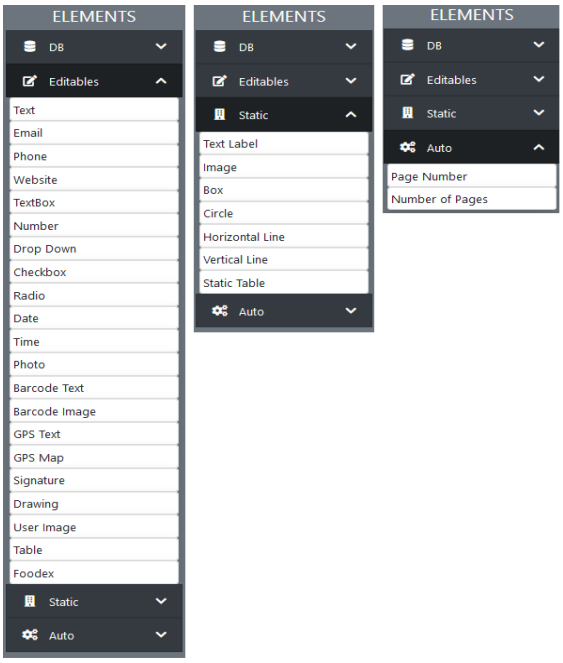

Fig. 13: The different elements in the Elements sidebar

On the right, we have the *Properties* panel, where one can change the individual or group properties of the selected elements. Note that the properties presented in this panel will depend on the selected elements (fig. 14).

Fig. 14: The Properties sidebar, depending on element's selection

In table 4, the visibility of each property according to the select element is presented.

	TEXT	EMAIL	PHONE	WEBSITE	TEXTBOX	NUMBER	DROPDOWN	CHECKBOX	RADIO	DATE	TIME	PHOTO	BARCODE TEXT	BARCODE IMAGE	GPS TEXT	GPS MAP	SIGNATURE	DRAWING	USER IMAGE	TABLE	TEXT LABEL	IMAGE	BOX	CIRCLE	LINE	HORI-ZONTAL LINE	VERTICAL LINE	PAGE NUMBER	NPAGES	TABLE	STATIC TABLE	FOODEX	MULTIPLE
URL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
PATTERN	1	1	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
GROUP	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
DATABASE	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
ITEMS	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CHECKED	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
REQUIRED	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
UPPERCASE	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
STEP	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MIN	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MAX	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MAX-LENGTH	1	1	1	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
PLACEHOLDER	1	1	1	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
CROSS	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	1
ENABLED	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1
DEFAULT	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
SHOW LABEL	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LABEL	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	0	1	0
NAME	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0
ID	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
ROTATION	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1
BORDER RADIUS	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1
BORDER WIDTH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1
BORDER	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1
BORDER BORDERS	1	1	1	1	1	1	0	0	0	1	1	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	1
BACKGROUND ALPHA	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	1
BACKGROUND COLOR	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	1
COLOR	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1
FONT ALIGNMENT	1	1	1	1	1	1	0	0	0	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1
FONT WEIGTH	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1
FONT DECORATION	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1
FONT STYLE	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1
FONT SIZE	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1
FONT ALIGNMENT	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1

HEIGHT	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	1	0
WIDTH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	0
LEFT	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
TOP	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Z-INDEX	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
VISIBLE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Tab. 4: Properties visibility vs selected Element(s) – 1 is visible, 0 is not visible

2.1.4 Tabs and Context Menus

The designer contains four main tabs, each one corresponding to an important step in the creation a form (fig. 15):

- FormView
- ListView
- Groups
- E/A

Fig. 15: Tabs

Each tab contains a context menu, consisting of a row of buttons. These are enabled/disabled depending on the situation, i.e., there are buttons that are only enabled when one or more elements are selected and there others that require at least one page to exist. As an example, fig. 16 presents the context menu for the *FormView* tab.

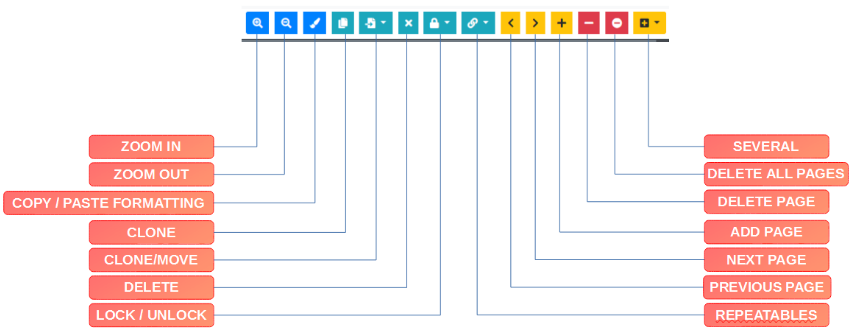

Fig. 16: The FormView context menu

2.1.5 Table Panel

This is a panel for the configuration of Table elements. For more information go to section 2.4.5.

2.2 Parts

The designer has four parts. These are listed and describe in table 5.

Part	Description
Form View	where the user creates the form, in a WYSIWYG fashion, i.e., just like in the paper form (section 2.6)
List View	where the user creates the different sections, their content and the order in which they will appear when the form's user selects the List View in their devices (section 2.7)
Groups	where the user puts the checkboxes and radios into groups (section 2.8)
E/A	where the user creates the automatization, based on events (section 2.9)

Tab. 5: The four parts of the Designer

2.3 Elements

2.3.1 DB

DB (database) elements are automatically created by selecting specific fields from a database. To create one of these elements, we go to Configuration > Create Elements from the Database (fig. 17). That will open a dialogue where we can select the database field and its type (fig. 18).

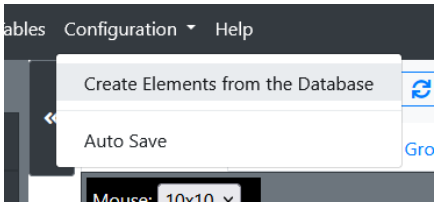

Fig. 17: Adding a new Element from a database

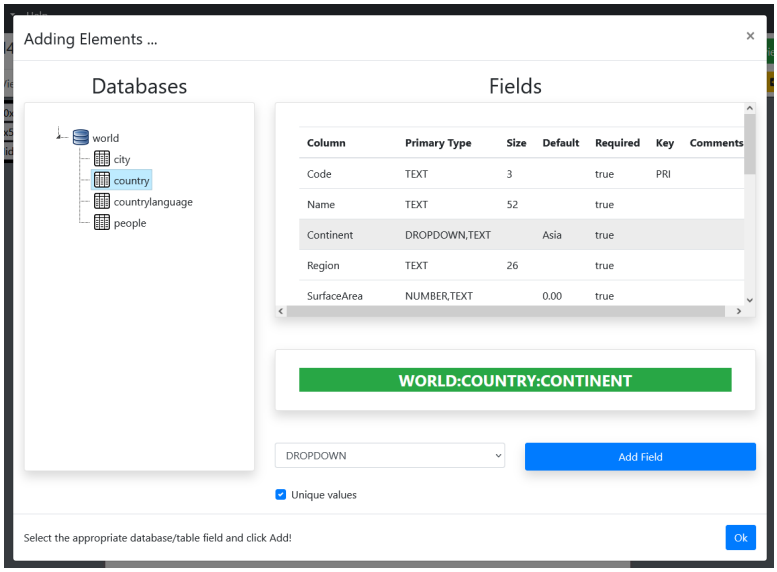

Fig. 18: Dialogue to create a database element

The editor tries to create an element that best matches the selected field. It also tries to obtain as many properties as possible, including, length, type, if it's required and so on.

Once the field and its type are selected, we press Add Field, which will add the newly element to the Elements sidebar, with an indication of its origin table and type (fig. 19).

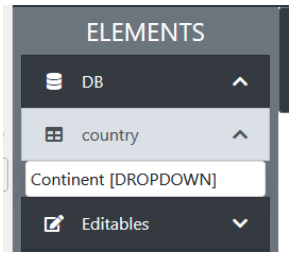

Fig. 19: Newly created element

If we drag the example above to the form and inspect the items, we notice that everything was automatically set (fig. 20).

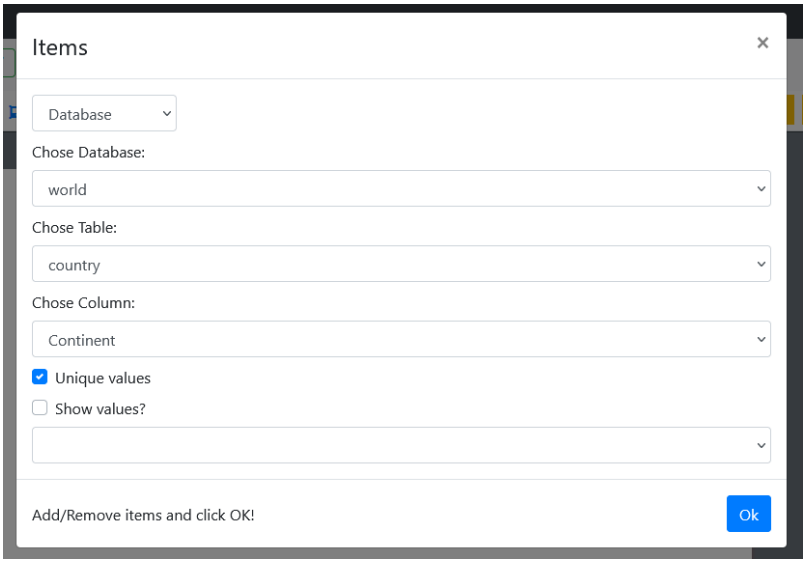

Fig. 20: All was automatically set in Define Items dialogue

If instead of a list we chose a radio element, it automatically creates all options, including automatically creating and setting up a group (fig. 21).

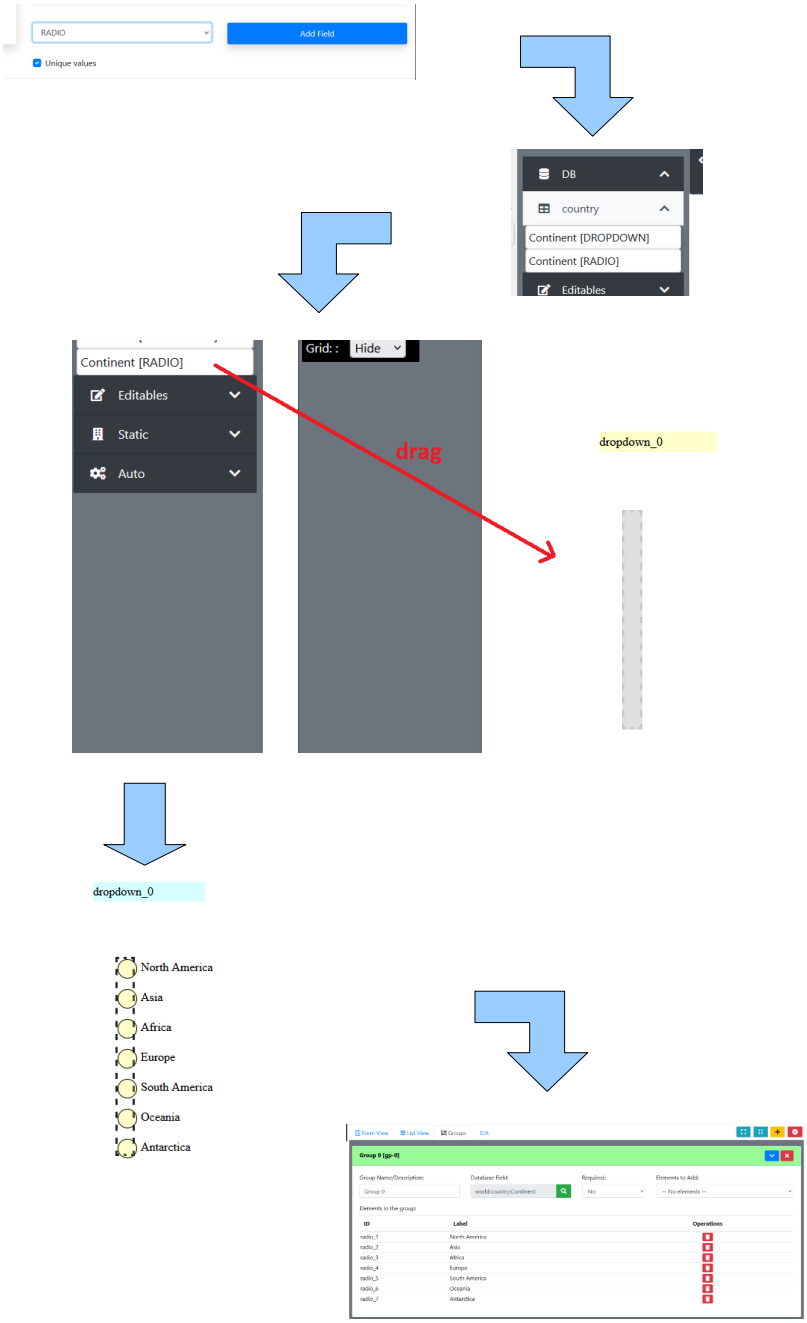

Fig. 21: A radio element from the database

Case the number of options surpasses 24 items, a confirmation dialogue pops up, requesting the user's permission to continue, see fig. 22. A high number of options can cause issues in the editor, from slowdowns to crashes.

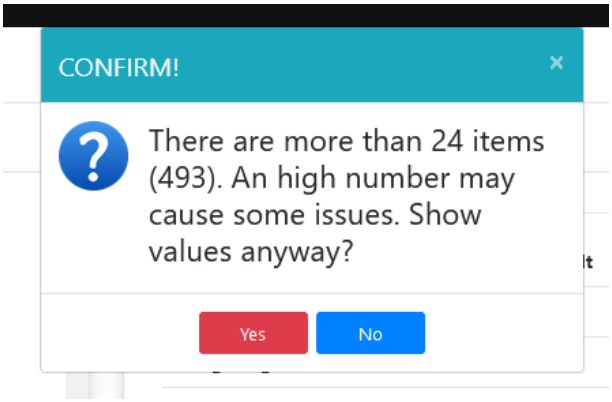

Fig. 22: Warning message on the number of items

Creating elements with this process is particularly useful for quickly creating checkboxes, radios and lists.

2.3.2 Editables

Editables elements (table 6) are elements that interact either with the user or with the E/A system. It allows the user, for example, to:

- input or select values;
- select or take pictures;
- draw;
- automatically get current coordinates;
- read barcodes.

Element	Description
Text	-
Email	-
Phone	-
Website	-
TextBox	Input extensive text with multiple lines
Number	-
Drop Down / List	Select one from a group of items
Checkbox	Select one or more from a group of options
Radio	Select just one from a group of options
Date	Input date with format: DD-MM-YY
Time	Input time with format HH:MM
Photo	Take a photograph (mobile) or select an image (pc)
Barcode Text	Input or take the barcode (manual or automatic)
Barcode Image	Input or take the barcode and its picture (automatic)
GPS Text	Input coordinates in text format (manual or automatic)
GPS Map	Input coordinates in map format (automatic)
Signature	Draw a signature
Drawing	-
User Image	Select a picture from local storage
Table	Dynamic table (the user can add/remove rows in real time), where each cell is an input (textbox)
Foodex	Name or foodex[1] code

Tab. 6: Editable elements

2.3.3 Static

Static elements (table 7) are elements that can't be changed by the user of the form. They can however interact with the E/A system, particularly their visibility.

[1] https://www.efsa.europa.eu/en/data/data-standardisation

Element	Description
Text Label	-
Image	-
Box	-
Circle	Basically a box or rectangle with rounded corners
Horizontal Line	-
Vertical Line	-
Static Table	Simple non-dynamic table without any input, other than the ones the designer manually places in each cell.

Tab. 7: Static elements

2.3.4 Auto

Automatic elements (table 8) are elements that are not directly controlled by the user, and whose values are automatically set depending on the situation. They can however interact with the E/A system, particularly their visibility.

Element	Description
Page Number	Current page number
Number of Pages	Total number of pages

Tab. 8: Auto elements

A classical example on the usage of these elements is presented in fig. 23.

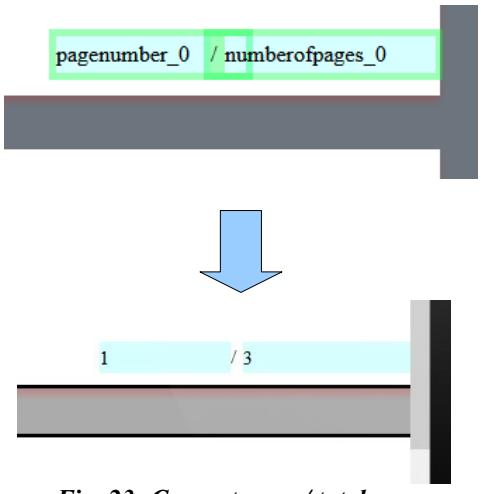

Fig. 23: Current page / total pages

2.4 Special Elements

2.4.1 DropDowns/Lists

Dropdowns are basically lists of values that the user can chose from. The designer provides three sources of data for its values:

- **manual** – manually define all items;
- **table** – the items come from a column of a *csv* table;
- **database** - the items come from a column of a specific database table.

Its selection and setup is done by clicking the "Define items" button on the properties panel, see fig. 24 and 25.

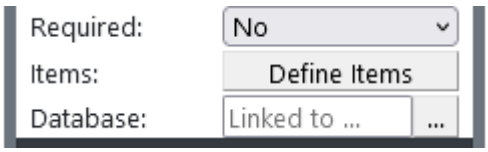

Fig. 24: The Define Items button in the properties sidebar

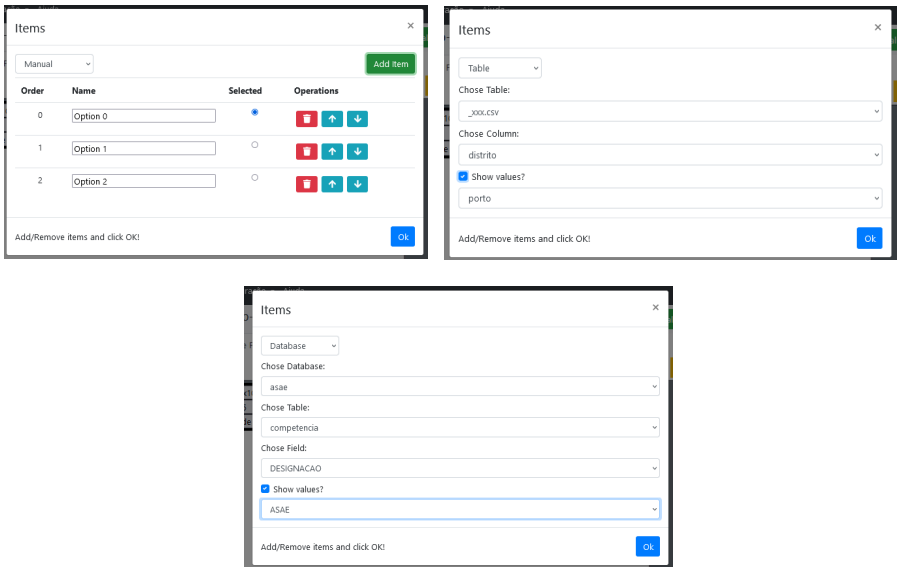

Fig. 25: The different data sources in the Define Items Dialogue

DO NOT FORGET TO PRESS OK!

A *DB Element* will have its items automatically set (see section 2.3.1).

2.4.2 Checkboxes and Radios

Checkboxes are elements that allows the user to select one or more values within a group. On the other hand, *Radios*, only allow the user to select a single value within a group (fig. 26).

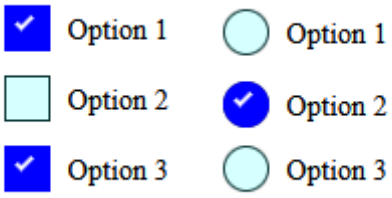

Fig. 26: Checkboxes and Radios

In of a *DB Element*, an array of these elements is automatically created and set, including their labels, names and a group containing all possible values.

2.4.3 GPS Text / Map

The designer provides two types of Map elements: text only and map + text (fig. 27). To automatically fetch the coordinates, requires the user to be in a secure context and the browser to support geolocation. If both these requirements are not fulfilled these elements will be unable to automatically fetch the current coordinates, and the user will have to manually input the coordinates.

To be considered secured, two criteria must be met:

- the form must be served over *https://* or *wss://* URLs;
- the security properties of the network channel used to deliver the resource must not be considered deprecated.

Fig. 27: Map elements

2.4.4 Barcode

As with the GPS elements, the designer also provides two types of barcode elements: only text or image + text. See Section 6.4 for more information.

2.4.5 Tables

2.4.5.1 Dynamic Tables

A *table* element, is as the name suggests a simple table where each cell corresponds to an input. By default, a table has 2 columns and 2 rows, apart from the header (fig. 28).

header 0	header 1
row 1	row 1
row 2	row 2

Fig. 28: Default Table element

This element has only a few properties available in the Properties sidebar (fig. 29). In the *ListView*, the text that will appear on the box of this element will be the *Label/ Text* or the *Name,* if not *Label* is specified.

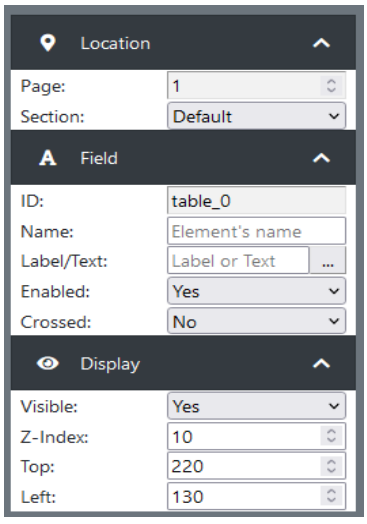

Fig. 29: Available properties for a Table Element

To edit the table itself, a movable panel is opened once a table is selected (fig. 30). In this panel, it's possible to change all the relevant aspects of the table, like the number of rows and columns, the header titles and colours, and so on. The name of the table is indicated on top of the panel. The name is set in the *Name* property.

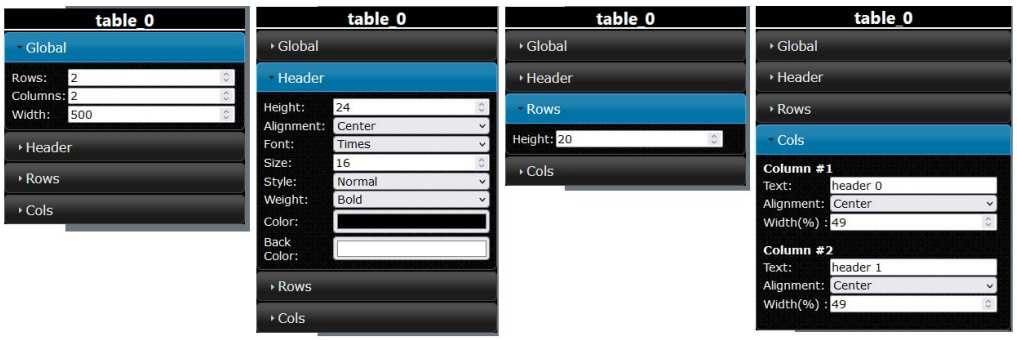

Fig. 30: The Table Element panel

We can easily turn the default table into something like depicted in fig. 31, which will show up in the preview as showed in fig. 32, ready for inputs to be introduced. Note the existence of one button, on each side of the table. These allows the user to add or remove rows from the table.

NAME	EMAIL	ID
row 1	row 1	row 1
row 2	row 2	row 2
row 3	row 3	row 3
row 4	row 4	row 4

Fig. 31: Example of a configured table element

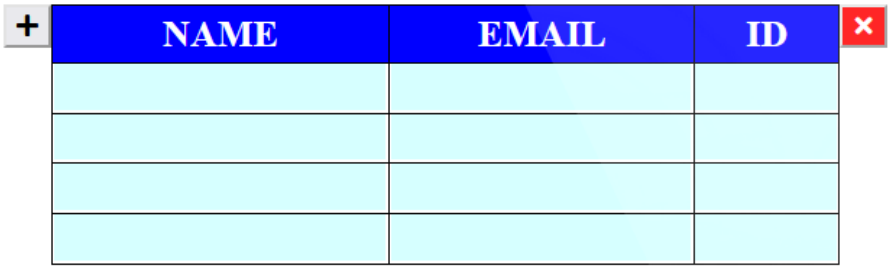

Fig. 32: Preview of the previous table element

This element has several limitations. These are:

- The row's font and size are limited to the default and they cannot be changed;
- E/As can only operate on the table level and not on the individual cell. For example, it's not possible to disable the second row of a table. It's only possible to disable all rows of the table;
- No validations are available;
- Only *TextBox* elements in each cell;
- Adding and removing rows will not affect any other element (see fig. 33 for an example);
- If when adding rows, these exceed the page limit, they will not trespass to the next page.

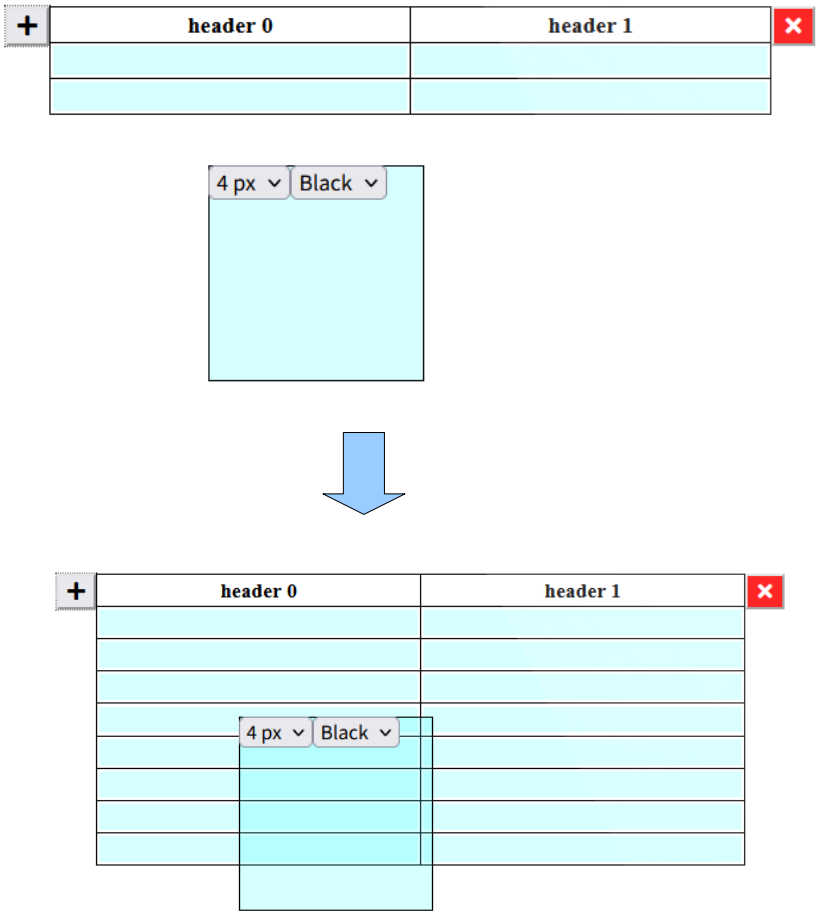

Fig. 33: Adding rows will not displace the elements below the table

2.4.5.2 Static Tables

Unlike the dynamic tables above, the properties of theses tables cannot be changed. This means that any inputs in them are the ones the form's designer placed in their cells. This element is useful to organize e create simple tables, that otherwise, would require extensive usage of lines and static text. See figure 34 for an example.

Question	Answer
Question 1	Yes No
Question 2	Yes No

Fig. 34: Application example of a static table

2.5 Load / Save Operations

There are three ways to save a form.

1. In the *Main*, select Save (fig. 35);
2. The quick save button (fig. 36);
3. Automatically, by turning on Auto Save (see section 2.1.1.2).

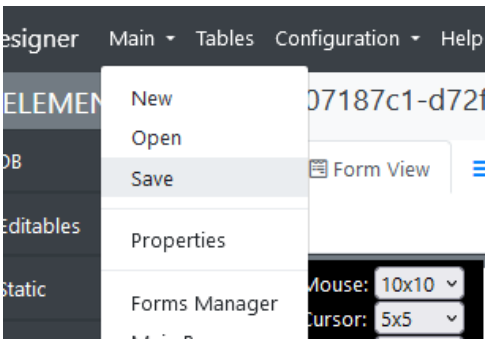

Fig. 35: Saving a form

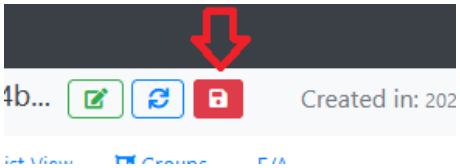

Fig. 36: The quick save button

Note that the quick save button will only be available after any change. After any save operation, the button automatically disappears.

EVERYTIME A FORM IS SAVED, ALL UNUSED ASSETS, LIKE IMAGES AND TABLES ARE DELETED!

2.6 FormView

2.6.1 UI

The context menu for the *FormView* is the most complex one of all (fig. 37). Not only contains several buttons but also several submenus (fig. 38).

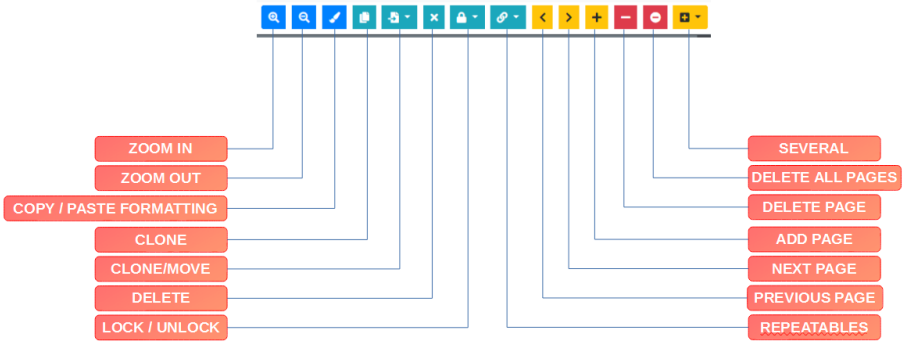

Fig. 37: The FormView context menu

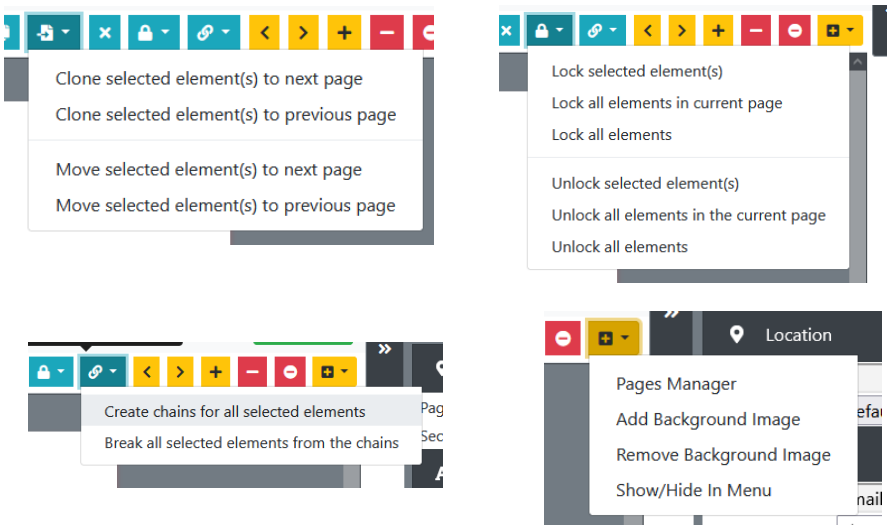

Fig. 38: The different submenus of the FormView context menu

2.6.2 Selection

Elements can either be individually selected by clicking on them. Fig. 39 present an example of clicking an element.

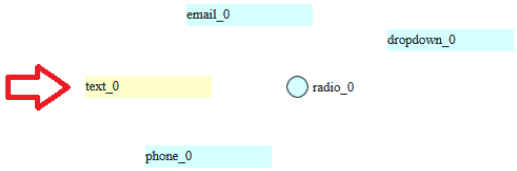

Fig. 39: Click selection

Multiple elements can be selected through a rectangular selection area (fig. 40).

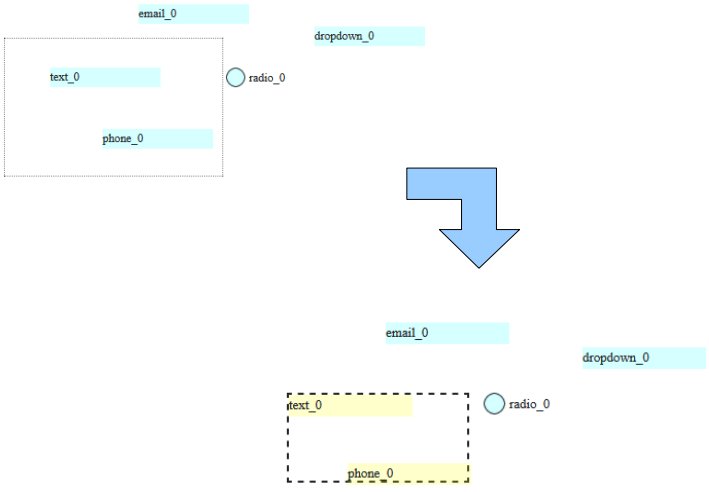

Fig. 40: Rectangular marquee selection

Shift select / unselect is also allowed.

Selected elements are highlight in yellow. Properties tab, will change not only according to the type of element selected but also if one or more elements are selected. Single selected elements have all their properties available, including the capacity to resize them directly. However, when multiple elements are selected, only some properties are available (fig. 41). Any change of these properties will be applied to all selected elements.

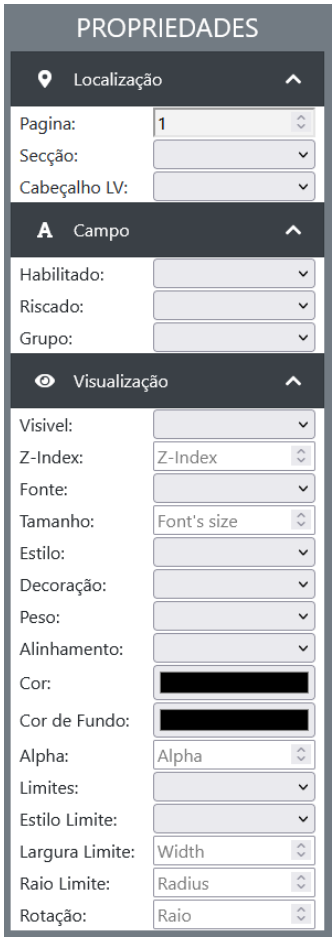

Fig. 41: The Properties sidebar for multiple selected elements

2.6.3 Repeatable Elements - Headers / Footers

Headers and footers can be created using *repeatable* elements. *Repeatable* elements are elements that are cloned and in-sync with each other through all pages. By “in-sync” we refer to synchronization, i.e., changing any property of field X at page 1, all fields X at every other page will also change accordingly. This also applies to operations like moving and deleting. So by deleting a repeatable element, all connected elements will also be delete.

Only *Static* and *AutoFields* can be set as *repeatable*. *Repeatable* elements are not allowed in E/A. Once an element are set as *repeatable*, a green outline will appear on them, as depicted in fig. 42.

Fig. 42: Repeatable Elements as page’s header

To set an element as Repeatable, select the desired elements and select *Create chains for all selected elements* option (fig. 43).

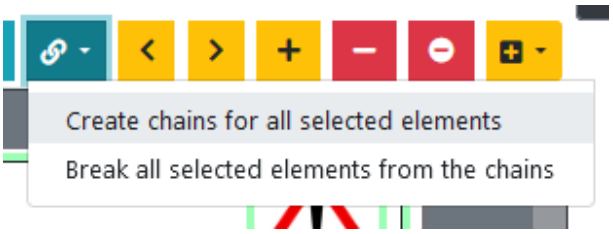

Fig. 43: Repeatables submenu

Consider the above case, where we have a header composed with a box, a title and an image. If we want to set an image in page n and not having the others automatically change as well, we can break the connection of this image element with the others, by selecting it and choosing "*Break all selected elements from the chains"*.

Creating a header in the *Form View* does not translates into the *List View*. To create a header for the *List View* check section 2.10.2.

2.6.4 Locked Elements

By locking elements we disable their selection and change. There are several available options to lock elements in the respective submenu (fig. 44). We can:

- select the element to lock and then "Lock selected element(s)"
- lock all element in the current page, or
- lock all elements in the form

On the other hand, to unlock elements, we can:

- individually select an element and then "Unlocks selected element(s)";
- unlock all element in the current page, or
- unlock all elements in the form

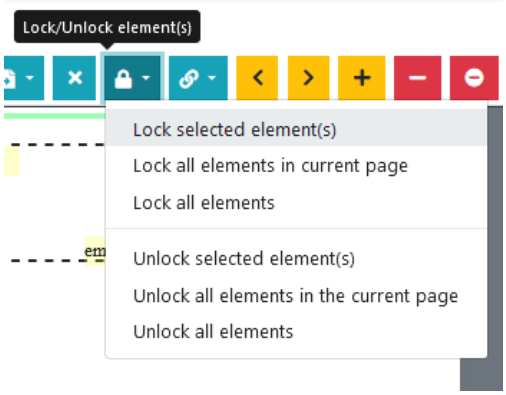

Fig. 44: Lock submenu

Locking elements are identified with a reddish outline, as depicted in fig. 45.

Fig. 45: A locked element (right) vs a non locked element (left)

2.6.5 Repeatable + Locked Elements

An element set both to be repeatable and locked, it presents a blue outline, as depicted in fig. 46.

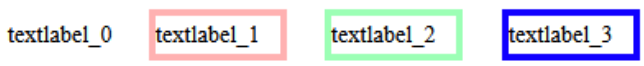

Fig. 46: A normal, a locked, a repeatable and a locked + repeatable elements

2.6.6 Pages Manager

This Manager can be found in the rightmost button of the *FormView* context buttons, see fig. 47.

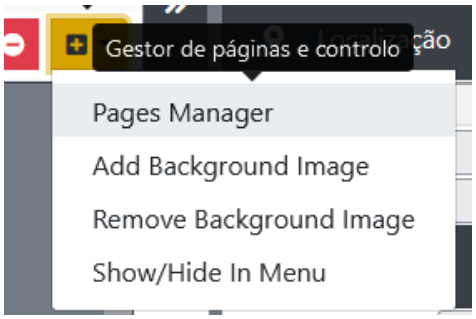

Fig. 47: The pages manager option

By selecting it, the following dialogue will appear. Here the user can create / delete / reorder pages, add and remove background images (fig. 48).

Fig. 48: The pages manager dialogue

2.6.7 Zoom

The context menu contains two buttons: one for zooming in and another zooming out. The zooming scale goes from ***0,7*** to ***1,5***. For more or less, the user can use the browser's zoom. Also if more space is required, it's possible to collapse the sidebars to get a larger working area. Actually, for now, it's recommended to zoom the browser window instead of using the editor's own zoom, since some rare minor issues with moving/resizing elements might occur.

2.6.8 Grid

The *grid setting* menu (fig. 49), allows to user to select the snap interval when dragging elements, either with the cursor or by mouse, and also the option to show or hide a grid layer over the page (fig. 50). By default the grid is not displayed.

Fig. 49: The Grid menu

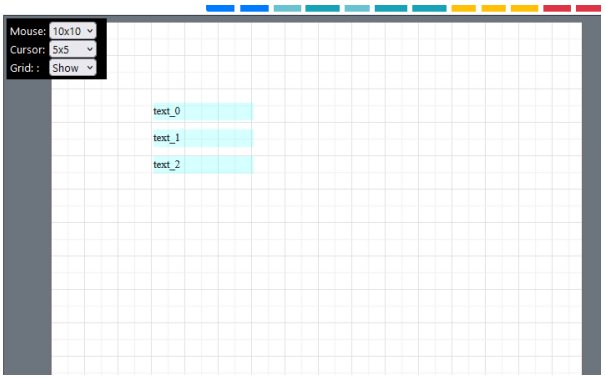

Fig. 50: Displaying the page's grid

2.6.9 Cloning

A *Cloning* operations refers to the act of duplicating one or more elements and all their properties (fig. 51).

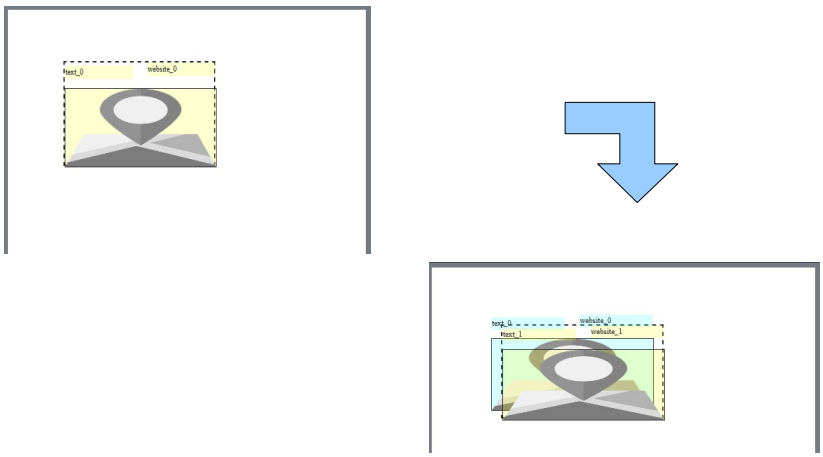

Fig. 51: Cloning selected elements

To clone or duplicate an element or a group of elements, the user just needs to select the element(s) and:

- click the cloning button (fig. 52) which will clone all selected elements in the same page, but in a distance of 20 units down and right, or

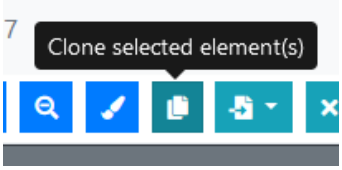

Fig. 52: Cloning button

- use the cloning submenu (fig. 53) to send the cloned elements to another page.

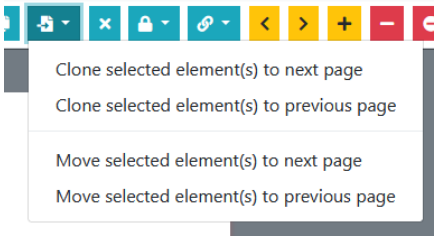

Fig. 53: Clone/move submenu

2.6.10 Moving

The editor does not allow the user to directly move elements between pages. However, the editor offers two alternative ways, which are present in the clone/move submenu (fig. 53):

- Move the selected elements to the next page
- Move the selected elements to the previous page

The moved elements will have the same relative position in their new page.

2.6.11 Copy/Paste Formatting - The brush

To transfer the display properties from one element to another, we use the *Copy / Paste Formatting* tool. To used it proceed as follows (fig. 54):

1. Select one or more elements. If more than one is selected, it will only consider the first one;
2. Select the *Copy / Paste Formatting* button. This action will the button red and to blink;
3. Select one or more elements. This action will paste the formatting properties to these elements;
4. Unselect the *Copy / Paste Formatting* button. This action will turn the button blue again.

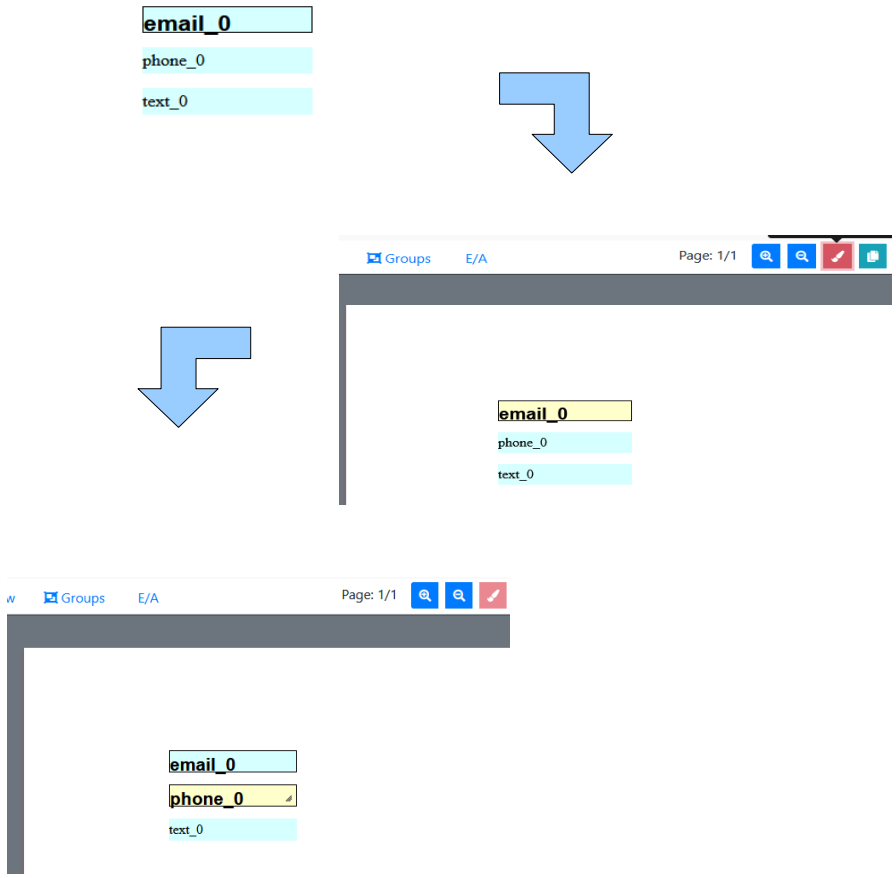

Fig. 54: The brush tool

Copied/pasted properties (except for tables):

- Visibility
- Cross

- Z-Index
- Font
- Decoration
- Weight
- Horizontal alignment
- Color
- Back color
- Back alpha
- Borders
- Border style
- Border width
- Border radius
- Rotation

It doesn't copy:

- Top
- Left
- Width
- Height

THE BRUSH REMAINS ACTIVE UNTIL IT'S DEACTIVATED.

In the case of *Table* elements, the brush will only work on another tables. The copied properties are:

- Visibility
- Cross
- Z-Index

and all the ones from the Table Panel, except the number of rows, columns and the header titles.

2.7 ListView

2.7.1 UI

The *ListView* UI consists of the respective context menu and two main lists (fig. 55):

- **sections** – a list of all sections in order, with the *Default* section as the first;
- **elements** – a list of all elements in order, of the currently selected section.

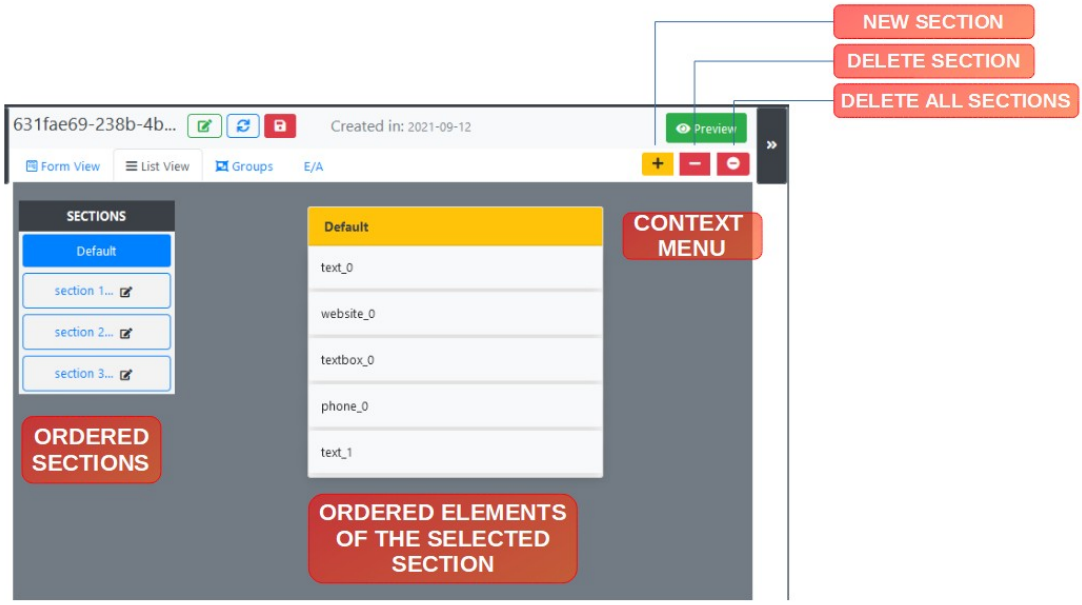

Fig. 55: Main view of the ListView tab

The context menu of the *ListView* consists of only three buttons:

- add section;
- remove section;
- remove all sections.

The *Default* section always exists and it cannot be renamed, deleted or moved. By default, every new element is placed inside this section, unless it's a cloning element. In this case, the new element will belong to the same section as the parents.

2.7.2 Ordering

The order of both the sections and the elements will be kept when the form is presented to the user (fig. 56). The ordering is done by dragging and dropping the elements/sections.

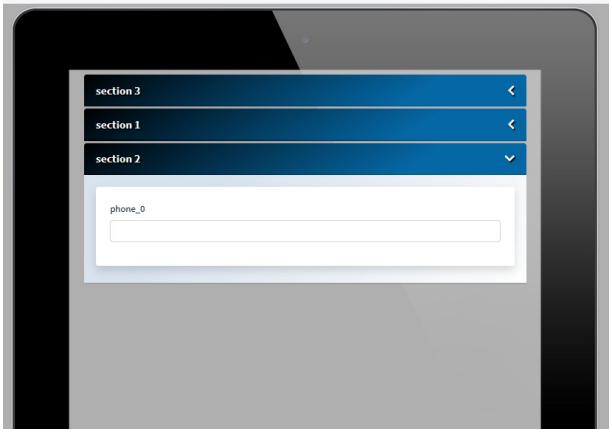

Fig. 56: Previewing 3 ordered sections

2.7.3 Changing Sections

To move elements between sections, either:

- drag the elements from its list to the sections list on the left, see fig. 57;
- in the *FormView*, by selecting the elements and selecting the Section at the properties panel, see fig. 58.

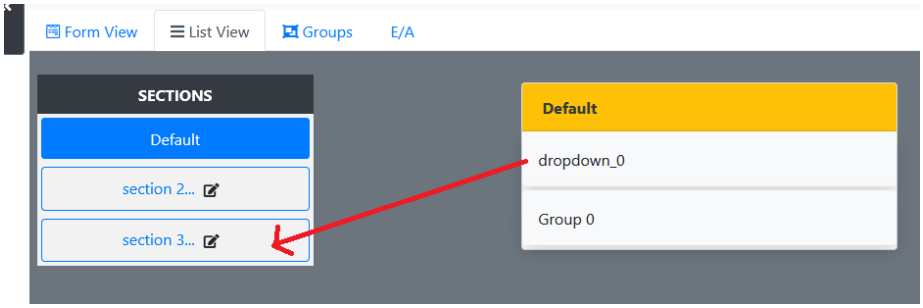

Fig. 57: Changing the section of an element by dragging it

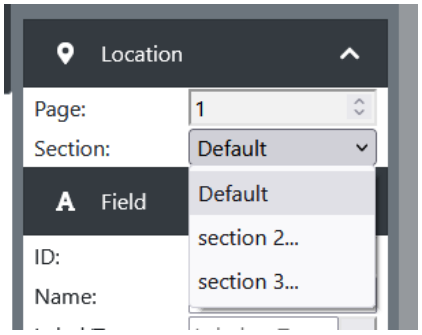

Fig. 58: Changing the section of an element or more by selection the new section in the Properties sidebar

2.7.4 Renaming Sections

Once a section is created, we can renamed it by clicking in the pen icon, present in all section in the list of sections, input the new name and press enter (or click outside) (fig. 59).

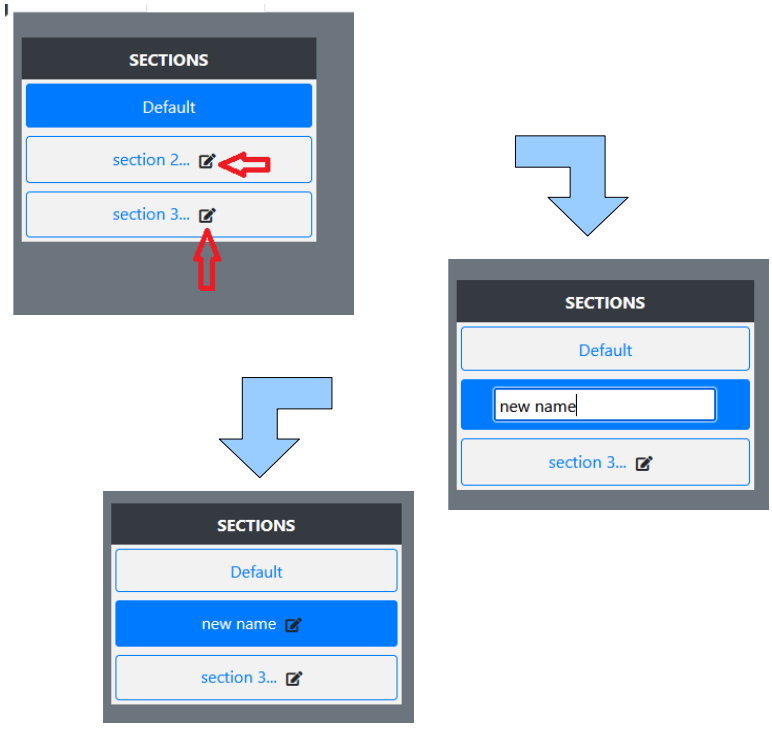

Fig. 59: Changing the name of a section

2.7.5 Important Remarks

Despite static elements like labels and squares are not visible and "editable" in the list view, it's possible to put them inside a section. This should be done through the *Properties* sidebar (fig. 60).

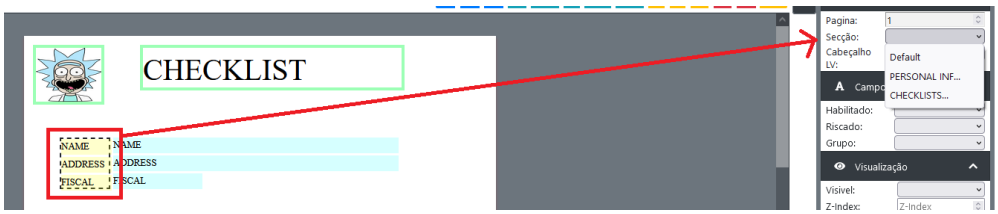

Fig. 60: Setting the section of static elements

2.8 Groups

2.8.1 UI

The *Groups* UI consists only of the respective context menu, which consists of four buttons: collapse groups, expand groups, add group and remove all groups (fig. 61).

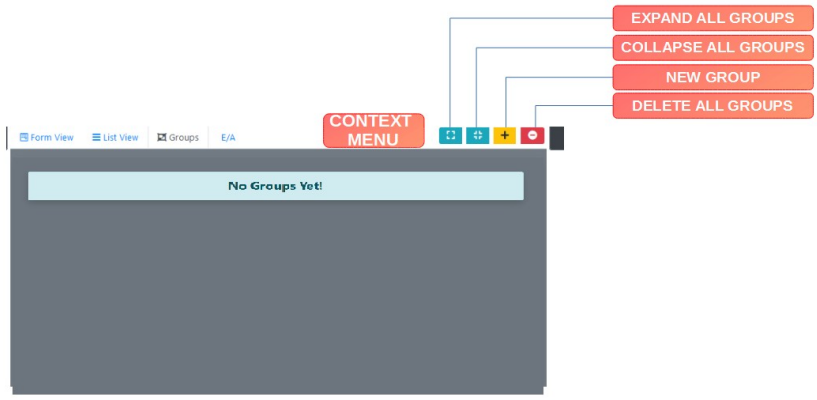

Fig. 61: Main view of the Groups tab

Figures 62 and 63 shows an example of two groups and its final result, both in the *FormView* and in the *ListView* is presented in figures 64 e 65.

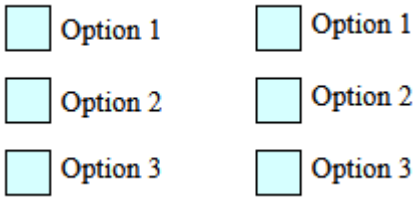

Fig. 62: Six checkboxes

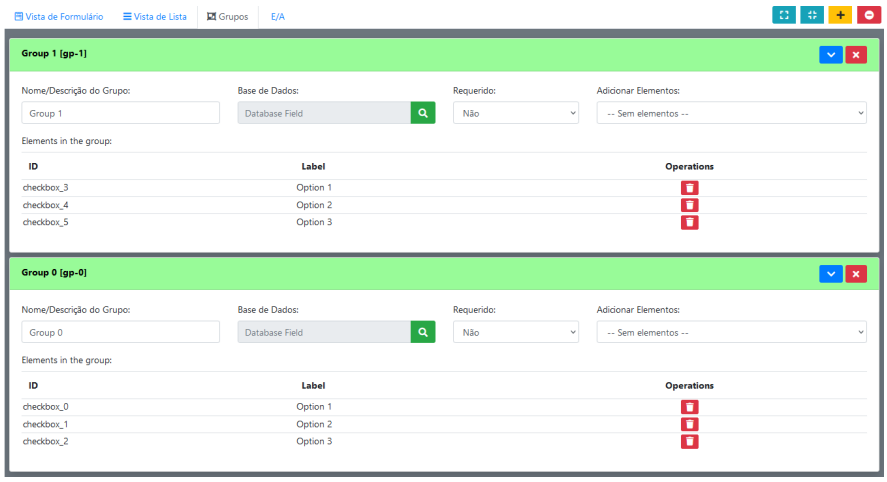

Fig. 63: Creating two groups with three checkboxes each

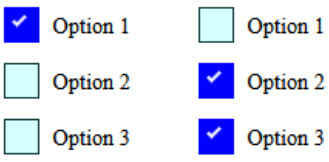

Fig. 64: View of the checkboxes in the FormView

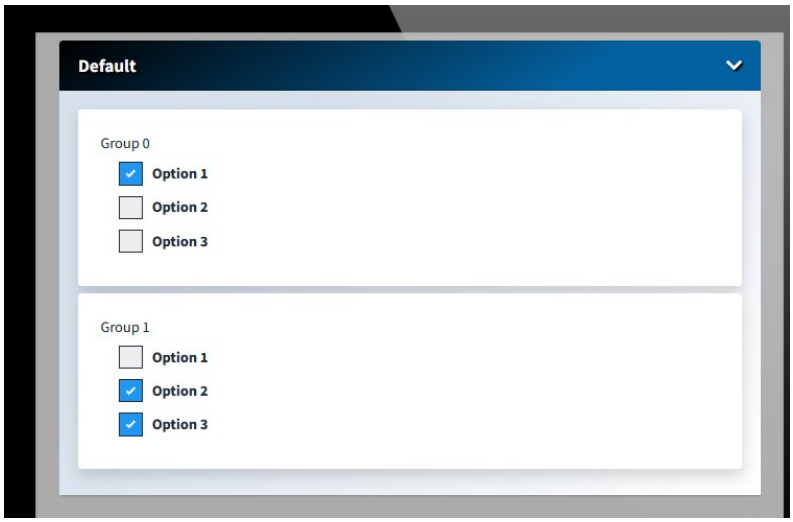

Fig. 65: View of the checkboxes in the ListView

2.8.2 Default Groups

Checkboxes and radio elements not belonging to a group, are actually grouped in a default group. It's possible to visualize this default group either in the *ListView* (fig. 66) or in the *preview*, where radios and checkboxes are separated (fig. 67).

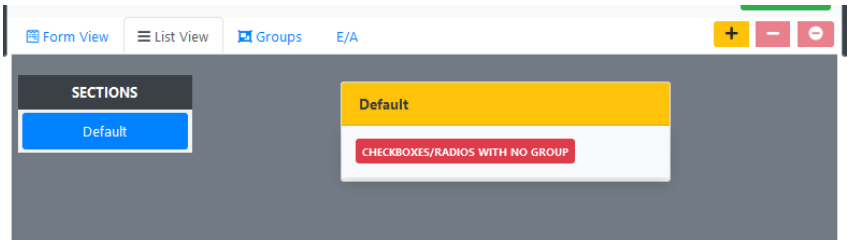

Fig. 66: How radios and checkboxes without a group are view in the ListView tab

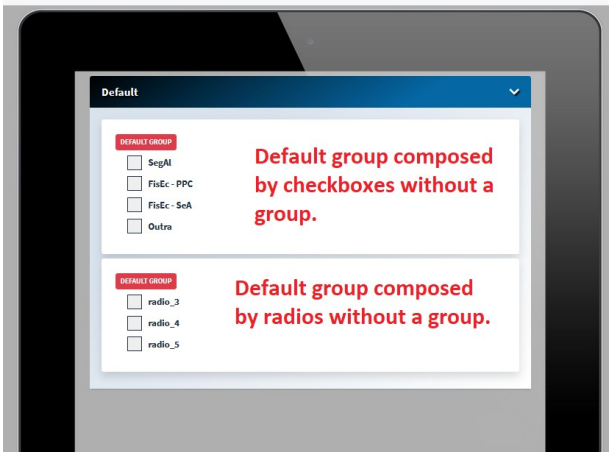

Fig. 67: How radios and checkboxes without a group are viewed in the preview

2.8.3 Group Type

A group can be either a checkbox or a radio group. The type is set by the first element added to the group. After the type is set, only elements of that type will appear in the *Element to Add* list.

A simple example is presented in fig. 68. Here we have 3 checkboxes and 3 radios. When we create a group, all elements, no matter it's type, are available to be added to the group. But, as soon we add the first one, a checkbox in this case, only elements of that type will be available to be added to that specific group.

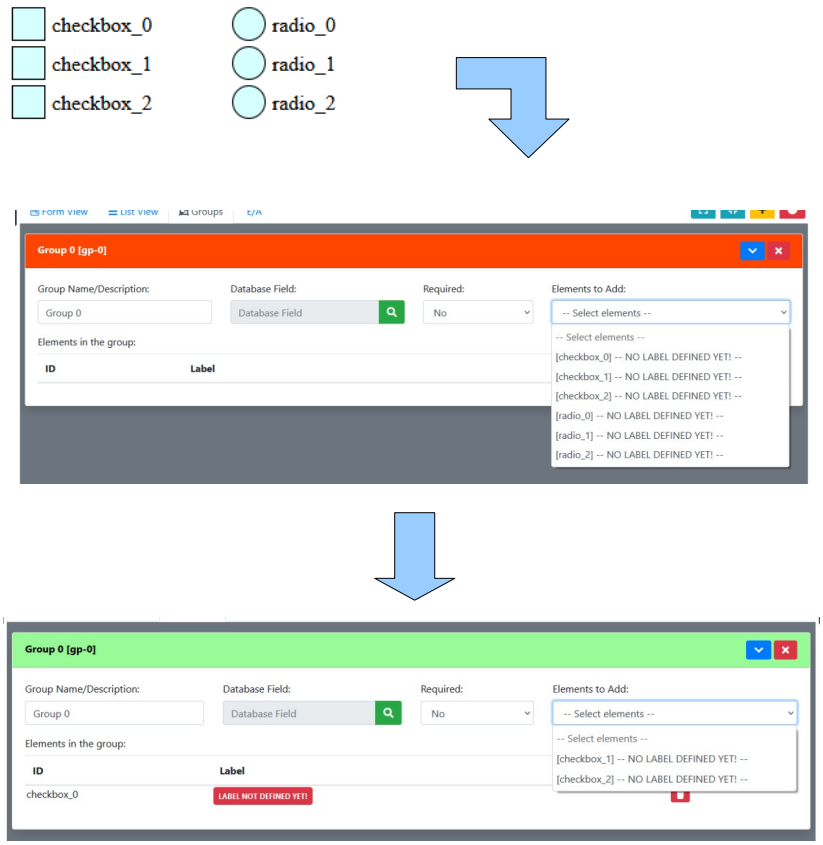

Fig. 68: Group types vs List of elements

2.8.4 Important Remarks

For quickly adding elements to groups, it's preferable to do it through the *Properties* sidebar (fig. 69). First create the groups, then select the elements and add them to their respective groups.

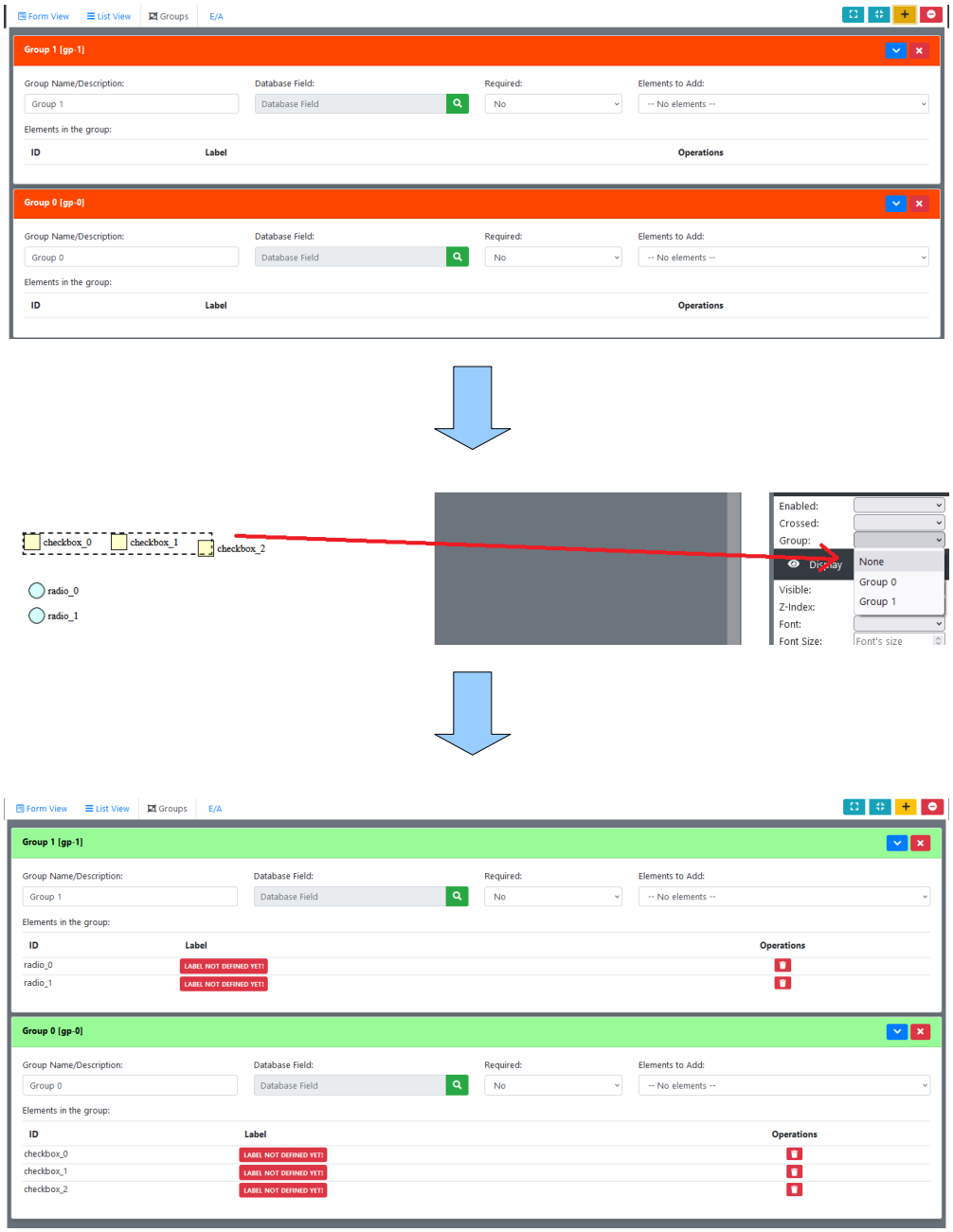

Fig. 69: Process of adding elements to groups through Properties

2.9 E/A

The *Events/Actions* systems allows the user to automate certain aspects of the forms. One or more events are used to trigger a specific action (see fig. 70).

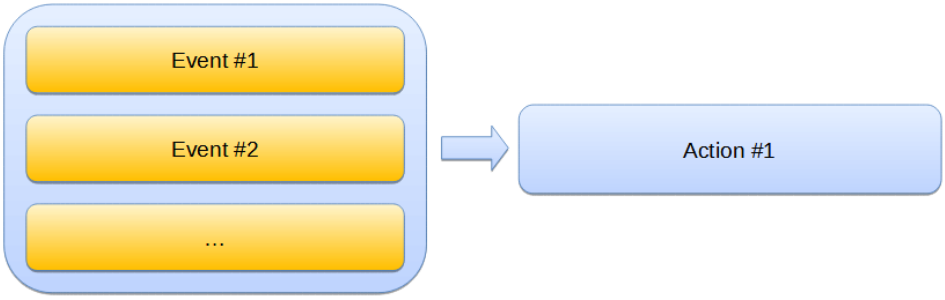

Fig. 70: Events → Action

Limitations:

One cannot reverse an action by reversing an event.
For example, if an E/A is defined by an action X that is triggered by checking a specific checkbox, one cannot reverse action X by unchecking that same checkbox. For that, it's necessary to create a new E/A with that unchecking event and a reverse X event.

No events are triggered at the start (except OnFormOpened).
So, if the visibility of an element is determined by a checkbox, when the form is loaded the first time, the system will not check the status of the checkbox and trigger any visibility event. Therefore, it's recommended for the user to set the initial or the default visibility of the element in its properties.

2.9.1 UI

The context menu of the *E/A* tab consists of four buttons: collapse E/As, expand E/As, add E/A and remove all E/As (fig. 71).

Each E/A follows the same UI principle: an events section and an action area.

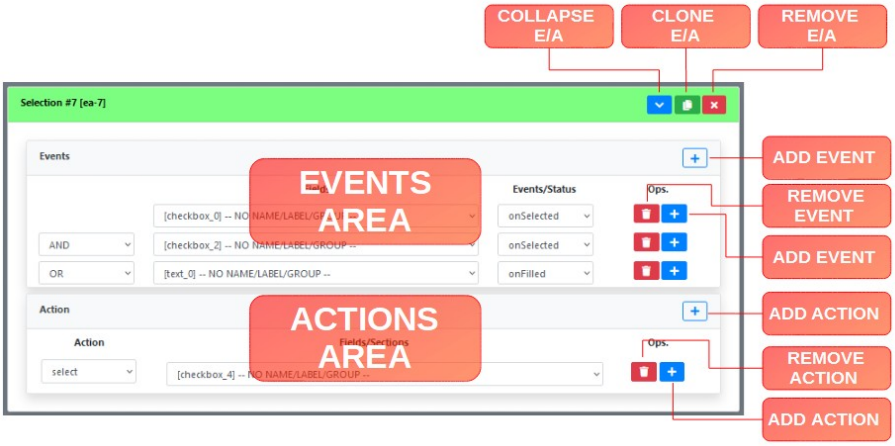

Fig. 71: An E/A example

2.9.2 Events

There are six classes of events, which are described in table 9.

Event	Description
OnChanged	Triggered when the value of an element changes
OnCleared	Triggered when the value is deleted
OnFilled	Triggered when filling an element
OnFormOpened	Triggered when the forms opens
OnSelected	Triggered when a dropdown/list value is selected or a checkbox / radio is checked
OnUnSelected	Triggered when a checkbox / radio is unchecked

Tab. 9: Events

Events can be composed, i.e., we can create a logical expression where multiple events are combined as one. An example is presented in fig. 72.

Fig. 72: Composing events

When creating these type of events, it's important to note that the *AND* operator has precedence do *OR*. In fig. 73 we give the truth tables for these operators.

x	y	x and y
false	false	false
false	true	false
true	false	false
true	true	true

x	y	x or y
false	false	false
false	true	true
true	false	true
true	true	true

Fig. 73: Boolean truth tables for AND and OR operators

So, for the example above, by using the precedence rule we have:

```
checkbox_0 AND checkbox_2 OR text_0 => (checkbox_0 AND checkbox_2 ) OR text_0
```

Another example would be:

```
checkbox_0 AND checkbox_2 OR text_0 AND radio_0 => (checkbox_0 AND checkbox_2 )
                                                    OR
                                                   (text_0 AND radio_0)
```

2.9.2.1 *Second Order Events*

A second order event occurs when an automatic change of a field triggers another event. Second order events are limited to certain situations. The only E/As where these events exists are in the *Append*, the *Database* and *Tables Queries*. When any of these E/As are executed, an "*onChanged*" event is called on the affected fields. So, if for example, after a database query, the name and the address fields are changed, then, an "*onChanged*" event will be called on these two fields.

2.9.3 Actions

In table 10, we present all available actions.

Action	*Sub-Action*	*Description*
Selection	**select / check**	Selects or checks a checkbox or radio element
	unselect / uncheck	Unselects a checkbox or radio element
Status	**cross**	Disables an element and draws a cross on it
	uncross	Enables an element and removes the cross on it
	enable	Disables an element (can't change any value)
	disable	Enables an element (can change its value)
Visibility	**show**	Shows an element
	hide	Hides an element
Append		Joins the value of one or more elements, and puts the result into another. The values can be connected by a specific connector
Format	**uppercase**	Uppercases the values of some specific elements
	lowercase	Lowercases the values of some specific elements
	first letter	Capitalizes the value of some specific element

Action	Sub-Action	Description
		(all words)
Query Database		Executes database query, using the values of specific *origin* elements and put the result in some *target* elements
Qyery Table		Executes a table query (fetched a value or column), using the values of some elements to fulfil some condition and puts the result in some *target* element.
Temporal	**now**	Sets the current date or time
Required	**required**	Sets that a value/option is required
	not required	Sets that a value/option is not required

Tab. 10: E/A actions

2.9.4 Creating an E/A

An E/A is created by selecting the respective action in the E/A context menu (fig. 74).

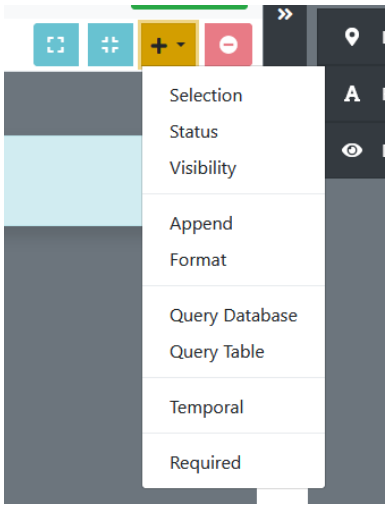

Fig. 74: Creating an E/A

Once selected, the respective E/A card appears (fig. 75).

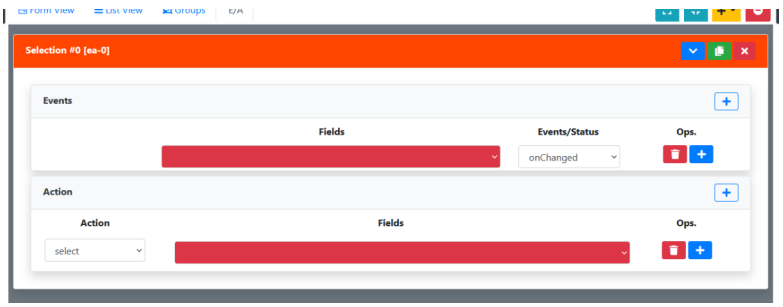

Fig. 75: The select E/A card

2.9.5 E/As

We now present an example for each available E/A.

2.9.5.1 Selection

In this example, we have a text and two checkboxes, as depicted in fig. 76). For the E/A we want to automatically check two checkboxes (checkbox_0 and checkbox_1) when element *text_0* is filled. Also, uncheck those 2 checkboxes when *text_0* is cleared.

Fig. 76: One text and two checkboxes

We start by selecting a *Selection* E/A (fig. 77).

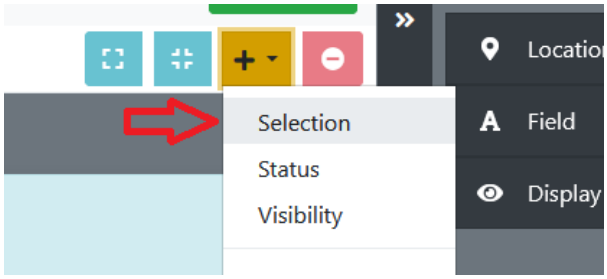

Fig. 77: New Select E/A card

We start with the first goal, which is to select both checkboxes when the text element is filled. So, we fill the event part, by selecting the text element (text_0) and the event *onFilled*. For the action part, we select both checkboxes and *select* as the action (fig. 78).

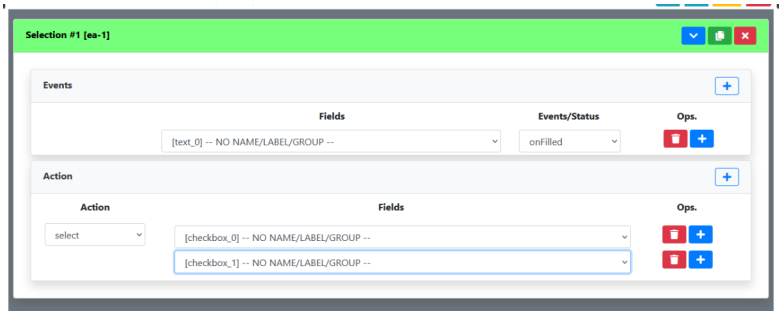

Fig. 78: onFilled select the checkboxes

For the second goal, which is to unselect both checkboxes when the text element is cleared, we either add a new E/A card by selecting in the context menu or by pressing the clone button of the previous card. Since *selection* and *unselection* are both sub-actions of the *Selection* E/A, it's recommended to use the clone button, since this will create not only the same type of card but also with all its fields already filled. In this

case, we only need to change the Action, from *select* to *unselect* and the Event from *onFilled* to *onCleared* (fig. 79).

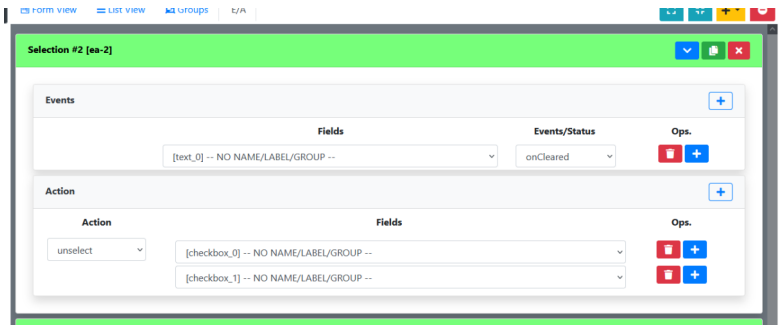

Fig. 79: onCleared unselect the checkboxes

Previewing the result we can observe the final result (fig. 80).

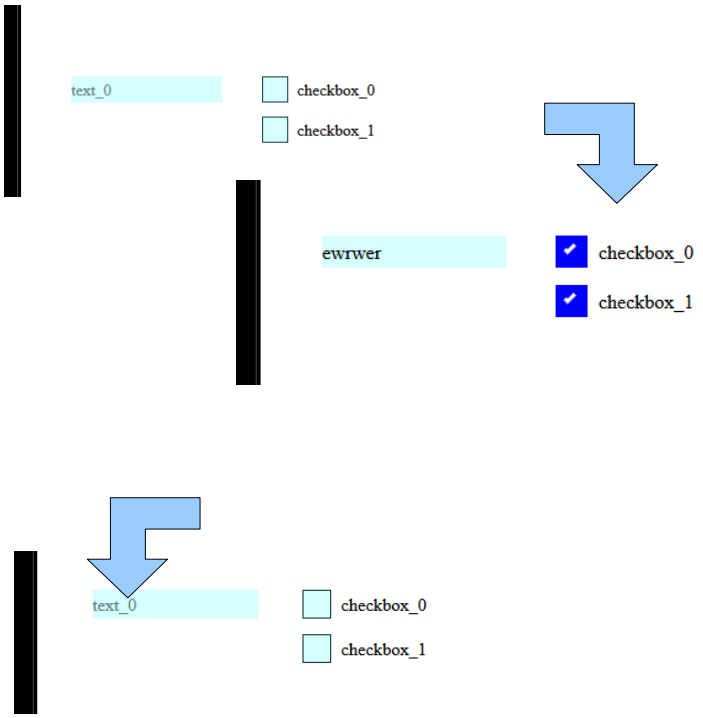

Fig. 80: On filling text_0, both checkboxes are selected and by clearing text_0 both checkboxes are unselected.

2.9.5.2 Status

In this example, we have a text and two checkboxes, as depicted in fig. 81. For the E/A we want to automatically cross and disable text_0 and checkbox_1 when checkbox_1 is checked.

Fig. 81: One text and two checkboxes

We start by selecting a *Status* E/A (fig. 82).

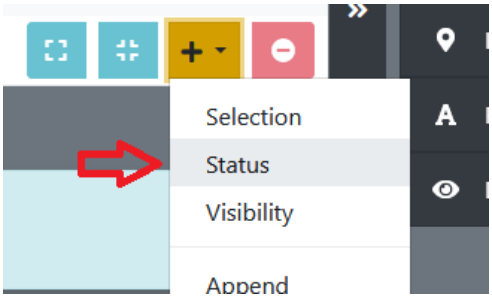

Fig. 82: New Status E/A card

So, we fill the event part, by selecting the checkbox_0 and the event *onSelected*. For the action, we select the remaining elements and *cross* as the action (fig. 83).

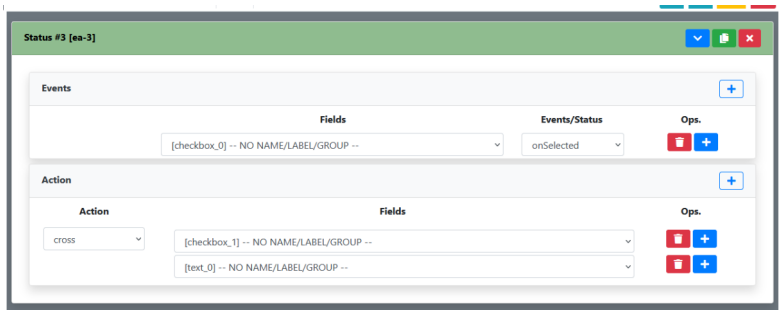

Fig. 83: onSelect checkbox_1 cross checkbox_1 and text_0

Previewing the result we observe the final result as depicted in fig. 84.

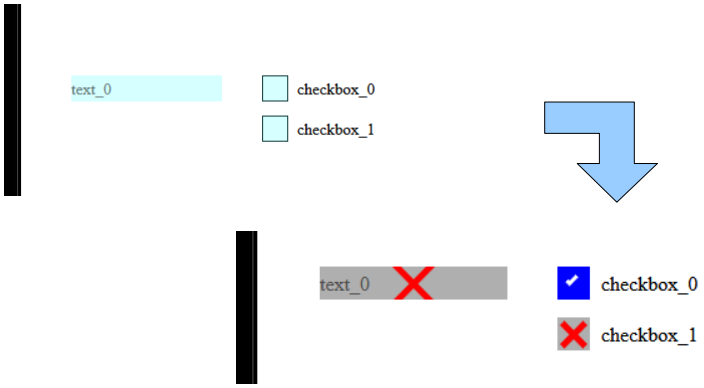

Fig. 84: On selecting checkbox_1, text_0 and checkbox_1 are crossed and disabled

2.9.5.3 Visibility

In this example, we have (fig. 85):

- 2 text elements in section A
- 2 checkboxes in section B

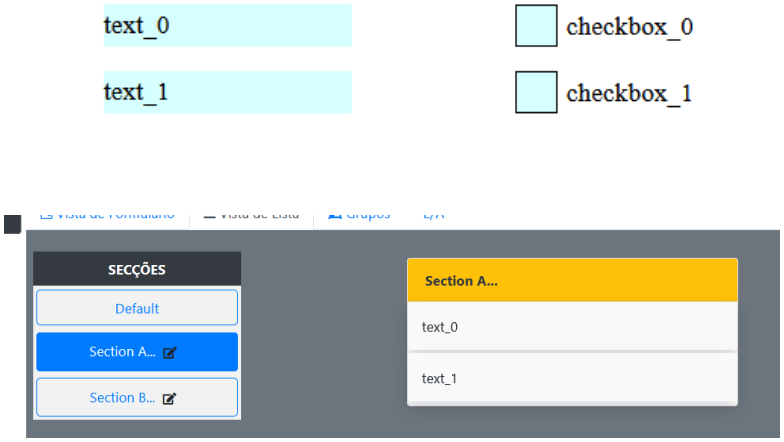

Fig. 85: Four elements and 2 sections

We two objectives:

1. hide Section A and all its elements when checkbox_0 is checked;
2. show Section A and all its elements when checkbox_1 is checked.

We start by selecting a *Visibility* E/A (fig. 86).

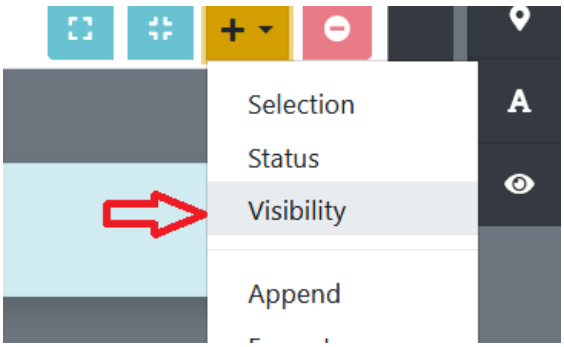

Fig. 86: New Visibility E/A card

So, we fill the event part, by selecting the checkbox_0 and the event *onSelected*. For the action, we select *Section A* and *hide* as the action (fig. 87).

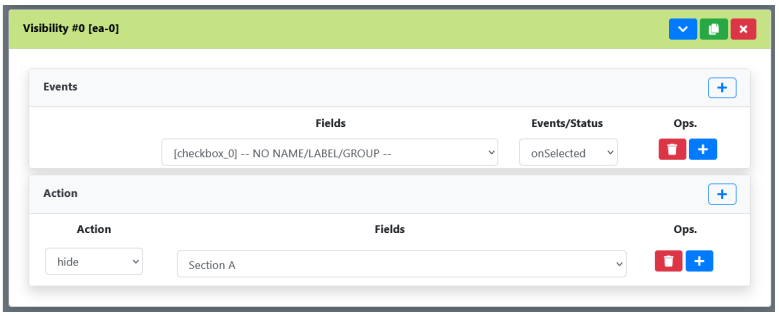

Fig. 87: onSelect checkbox_0 hides Section A and all its elements

For the second goal, which is by selecting checkbox_1, Section A is visible, we either add a new E/A card by selecting in the context menu or by pressing the clone button of the previous card. Since *hide* and *show* are both sub-actions of the *Visibility* E/A, it's recommended to use the clone button, since this will create not only the same type of card but also with all its fields already filled. In this case, we only need to change the Action from *hide* to *show* and in the event select checkbox_1 (fig. 88).

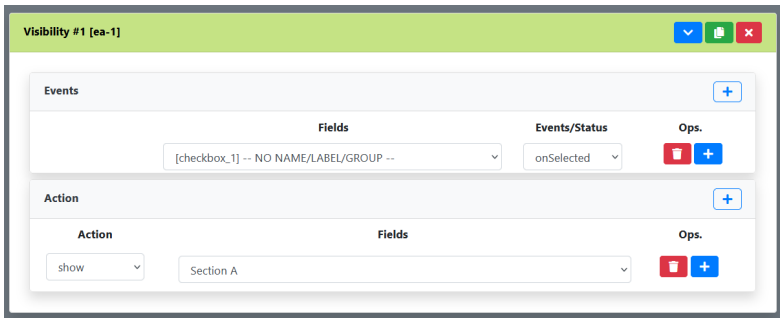

Fig. 88: onSelect checkbox_1 show Section A and all its elements

Previewing the result we observe the final result as depicted in fig. 89.

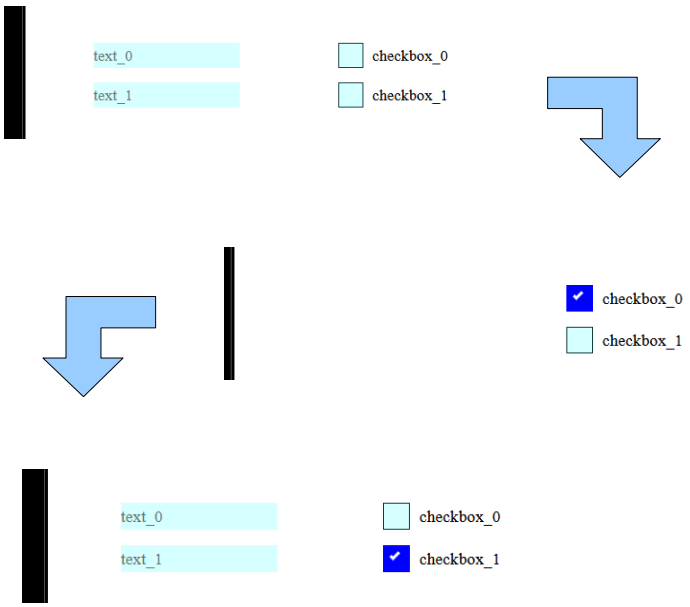

Fig. 89: On selecting checkbox_0, hide Section A and on selecting checkbox_1 show it again

2.9.5.4 Append

In this example, we have 3 text elements, text_0, text_1 and text_2, as depicted in fig. 90.

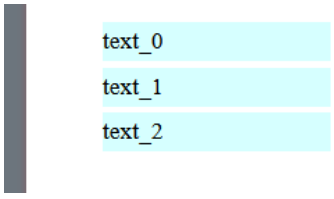

Fig. 90: Three text elements

For the E/A we want to automatically concatenate the contents of *text_0* and *text_1* and put the result in *text_2*.

We start by selecting an *Append* E/A (fig. 91).

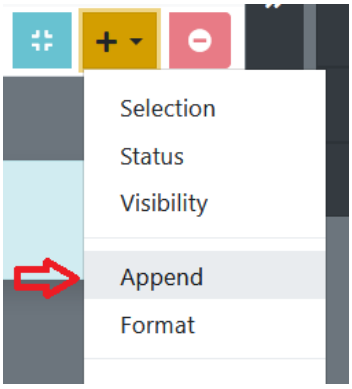

Fig. 91: New Append E/A card

So, we fill the event part, by selecting the *text_0* and *text_1* and the event *onChanged* or *onFilled*. For the action, we select *text_2* as target and *text_0* and *text_1* as the origin fields (fig. 92). Also, as the connector we just input a white space.

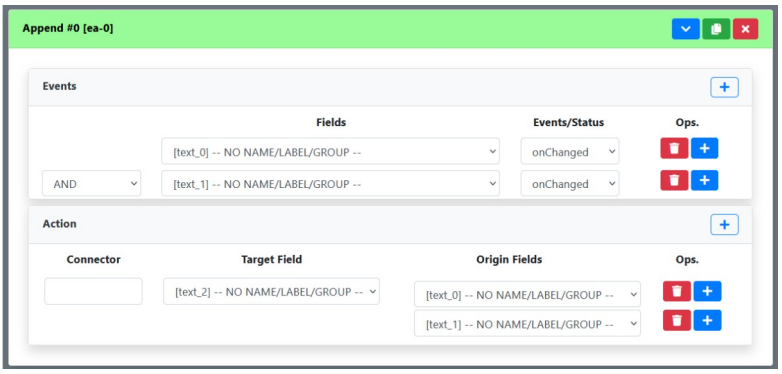

Fig. 92: text_2 = text_0 + text_1

Previewing the result we observe the final result as depicted in fig. 93.

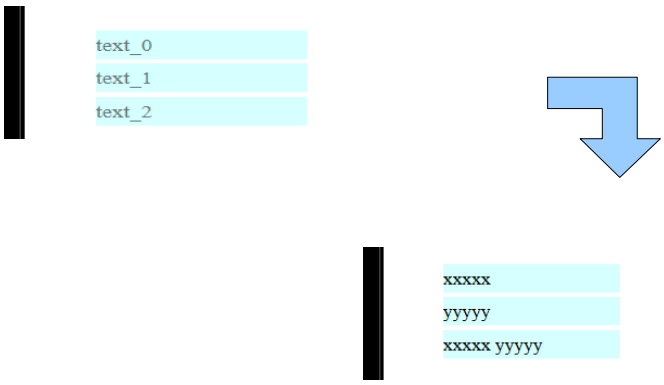

Fig. 93: On changing text_0 and text_1 contents, join their contents and put it in text_2

2.9.5.5 Format

In this example, we have 3 text elements, text_0, text_1 and text_2, as depicted in fig. 94.

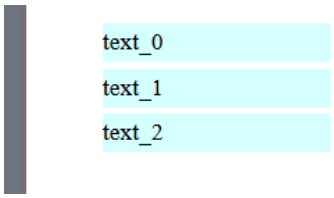

Fig. 94: Three text elements

In this example we will try all 3 sub-actions:

1. when text_0 is filled, turn all its characters uppercase;
2. when text_1 is filled, turn all its characters lowercase;
3. when text_2 is filled, turn all its first characters uppercase.

We start by selecting a *Format* E/A (fig. 95).

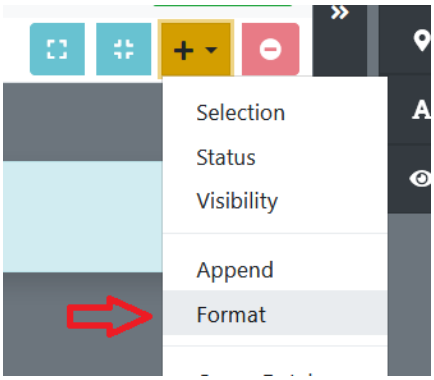

Fig. 95: New Format E/A card

We start with the first case, as show in fig. 96.

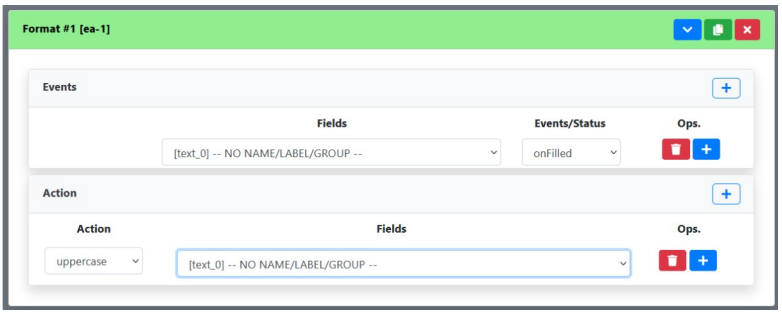

Fig. 96: When text_0 is filled, turn all its characters uppercase

We then move to the second and third cases, as show in figs. 97 and 98.

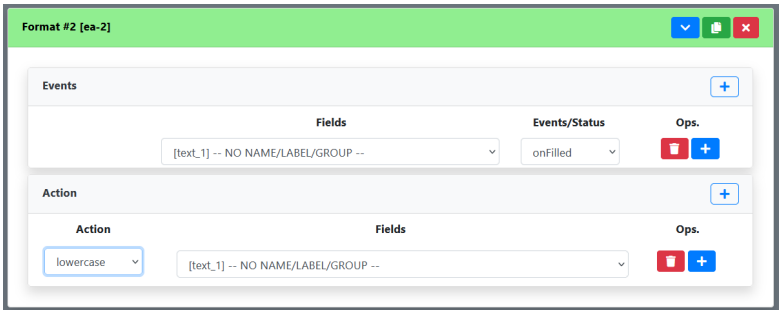

Fig. 97: When text_1 is filled, turn all its characters lowercase

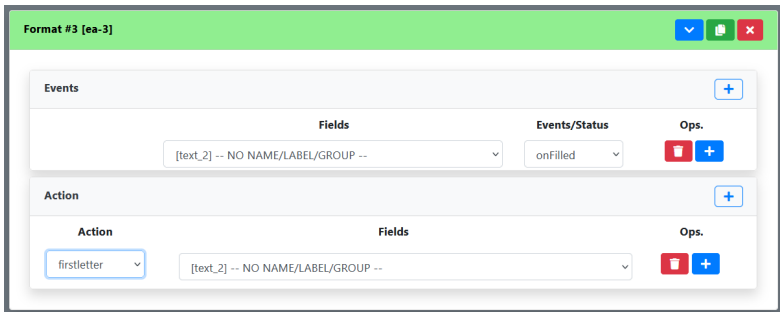

Fig. 98: When text_2 is filled, turn all its first characters uppercase

Previewing the result we observe the final result as depicted in fig. 99.

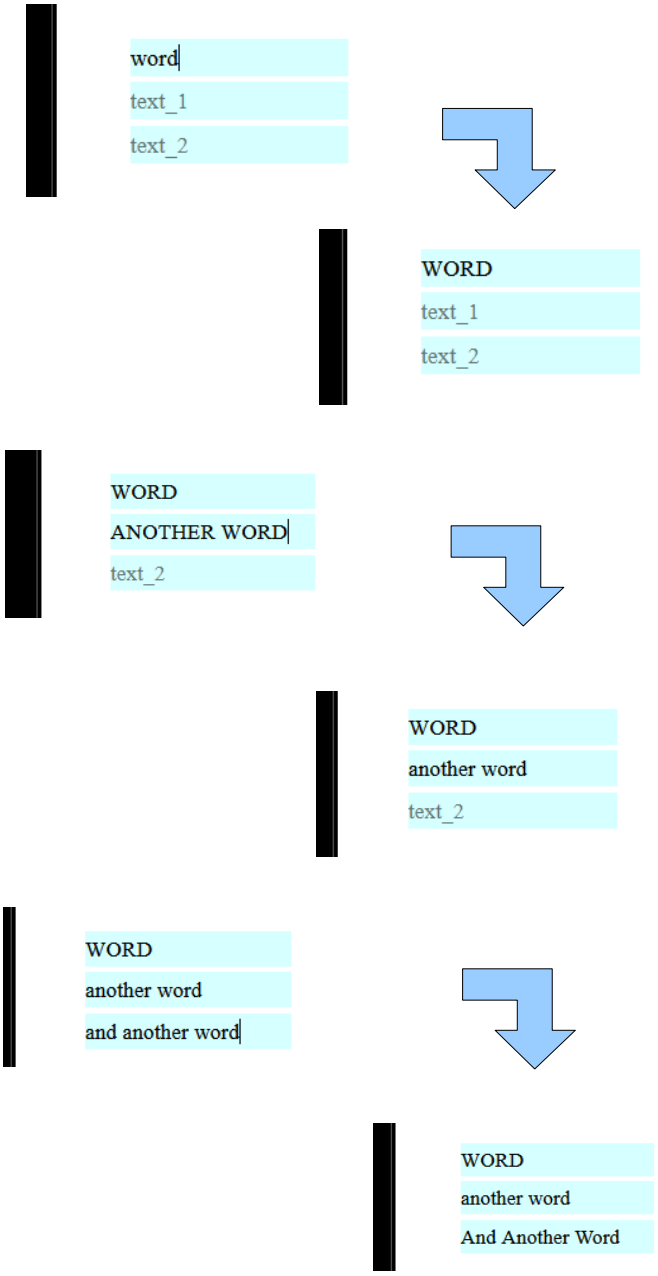

Fig. 99: Format on the 3 texts

2.9.5.6 Query Database

In this example, we have one checkbox: checkbox_0 and two lists: dropdown_o and dropdown_1, as depicted in fig. 100.

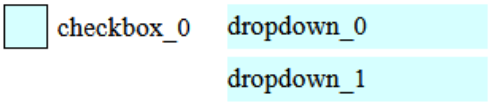

Fig. 100: One checkbox and two lists

Our goal is for when *checkbox_0* is checked, execute query *A001*, and put the results (names and addresses) in the list *dropdown_0* and *dropdown_1*.

Query *A001*:

```
SELECT $name$, $address$, $country$, $phone$ from somedatabase.target limit 10
```

The database is composed of only one table, with the data presented in fig. 101.

id	name	address	city	country	phone	email	nif
1	name 1111	address 1	city 1	country 1	phone 1	email 1	1111
2	name 2222	address 2	city 2	country 2	phone 2	email 2	2222
3	name 3333	address 3	city 3	country 3	phone 3	email 3	3333
4	name 4444	address 4	city 4	country 4	phone 4	email 4	4444

Fig. 101: Data to query on

We start by selecting a *Query Database* E/A (fig. 102).

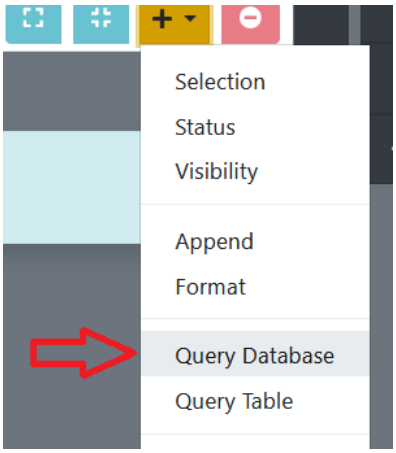

Fig. 102: New Query Database E/A

In the events section, we select *checkbox_0* and *onSelected* event. In the action section, we start by selecting the query *A001* and clicking Auto Populate button, which will automatically create all row according to the query (see fig. 103).

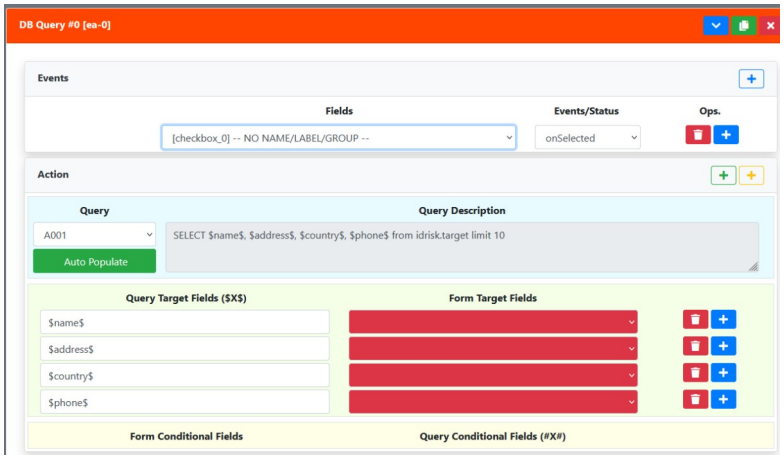

Fig. 103: First steps in setting up a database query E/A

Since we only need the names and addresses, we delete the country and phone rows, and then select where the results will be stored, which in our case, will be dropdown_0 and dropdown_1, for the names and addresses respectively (fig. 104).

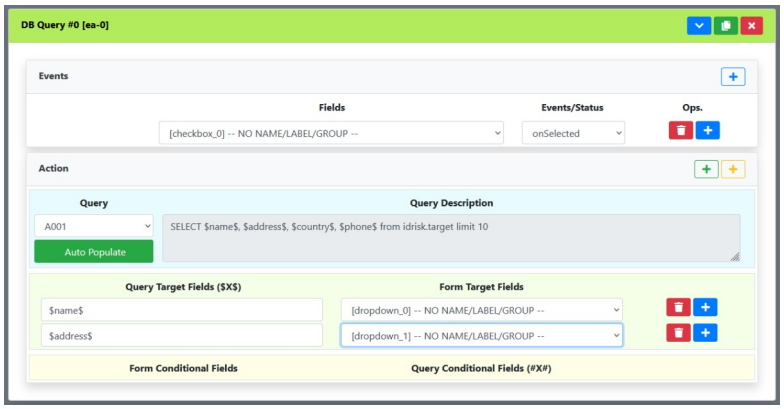

Fig. 104: Final steps in setting up a database query E/A

Previewing the result we observe the final result as depicted in fig. 105.

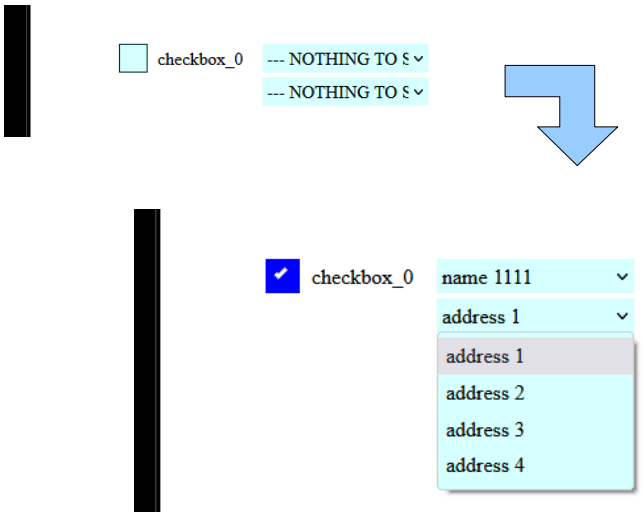

Fig. 105: Querying a database

2.9.5.7 Table Query

In this example, we have three list elements (fig. 106).

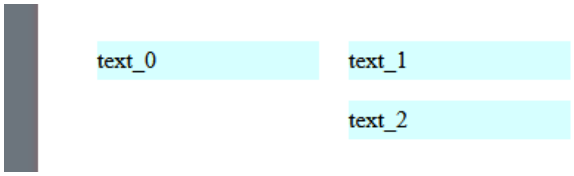

Fig. 106: Three text elements

We also have a table with five columns, containing data about a group of individuals (see table 11).

ID	Name	Address	Telephone	Email
1000	John Doe	23 Glendale Ave	+1 229-370-2722	EQFKG8N9BHJG@email.com
2000	Jane Doe	1051 Northview St	+1 582-222-5935	2YVJ6W27Y7WO@emai.com
3000	Monsieur Untel	505 E Grant St #A	+1 412-706-6431	6ESUUDXKXJOX@emai.com
4000	Madame Unetelle	333 Groton St	+1 202-812-3949	7N511GNA17VF@emai.com
5000	Ivan Horvat	22821 Ann Miller Rd	+1 505-399-3549	6ZFMXY56L656@emai.com
6000	Ivana Horvat	9099 Waters Edge	+1 210-472-6043	W4M48FE70K1O@emai.com
7000	Erika Mustermann	1811 W Chelten-ham Ave	+1 582-444-4325	MBB77U2Q44L2@emai.com
8000	Lieschen Müller	301 NE 174th St #611	+1 513-219-2333	IXYE63S4DQD6@emai.com
9000	Nguy?n V?n A	6387 Dustin Rd	+1 458-477-9497	QMP4LFAVJ9MH@emai.com
10000	Nguy?n Th? B	114 Myrtle Ave #B4	+1 614-630-2364	HQ1H6DX6JN8Y@emai.com
11000	Jan Jansen	1395 Zion Ridge Rd	+1 223-289-9765	AUCOU7PPKXR0@emai.com
12000	Maija Meikäläinen	89911 52nd St	+1 517-940-5628	K9B60LZJ38UX@emai.com

Tab. 11: Data to query

The goal for this example is to fetch some data about an individual, by given his ID.

We start by adding a *csv* file containing the data to the form. For that, we select Tables in the top menu and we either select or drop the file in the Tables Manager dialogue (see fig. 107).

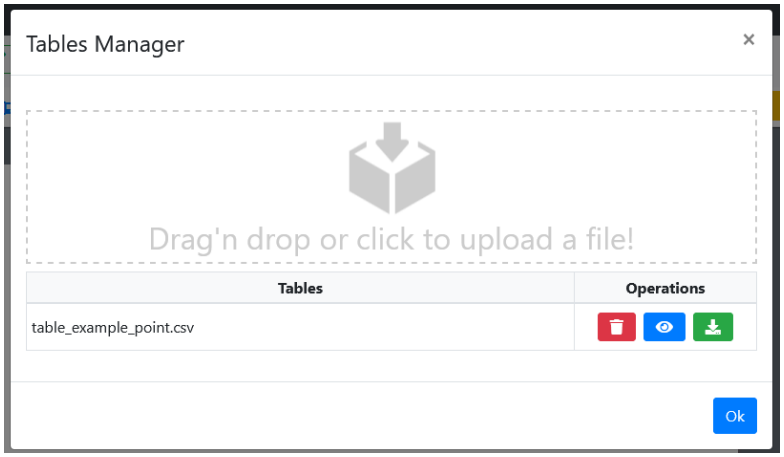

Fig. 107: Adding a csv table

Be aware that it you save the form right now, this table will be automatically removed from the form, since it's not being used.

We then select a *Query Table* E/A (fig. 108).

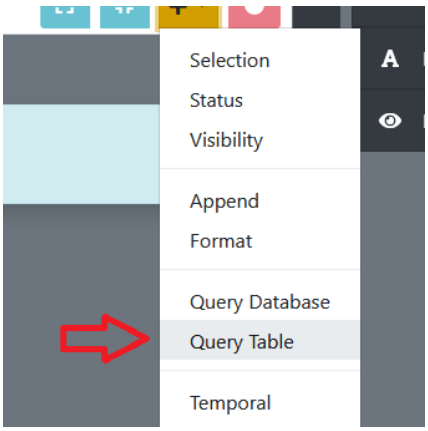

Fig. 108: New Query Table E/A

For the events we select *text_0* and *onFilled*. In the actions, we select the table, we add a new row and select the two columns we want to fetch, Name and Email, and where we want to put the data: *text_1* and *text_2*. We also need to set the condition, which in this case is ID = *text_0*. In fig. 109 we present the final form of the E/A.

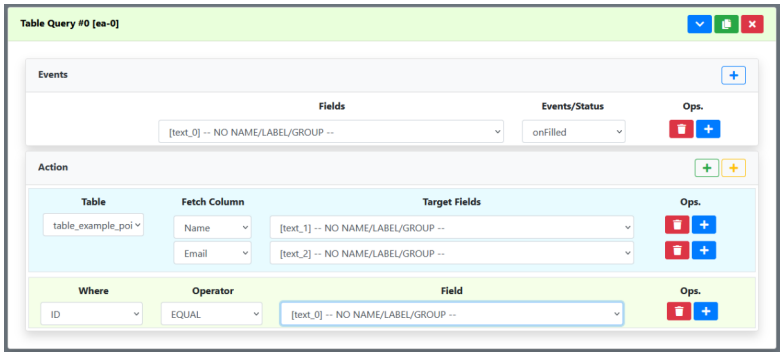

Fig. 109: Final steps in setting up a table query E/A

Previewing the result we observe the final result as depicted in fig. 110.

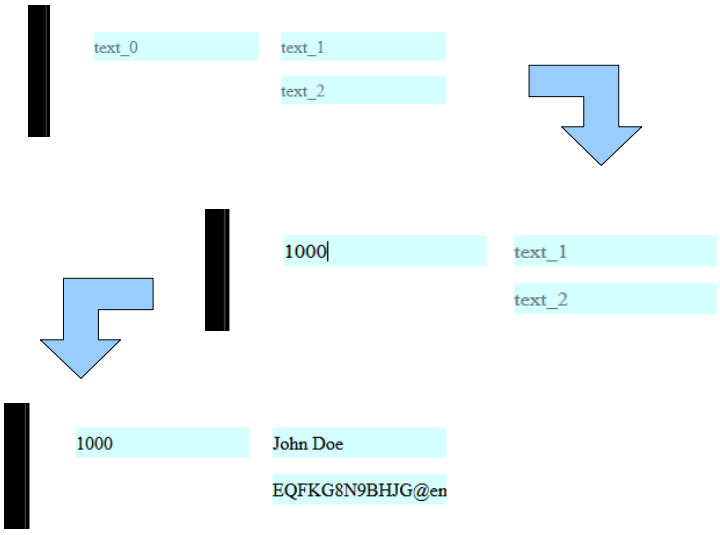

Fig. 110: Querying a table

2.9.5.8 Temporal

In this example, we have two elements, a date and a time elements (fig. 111).

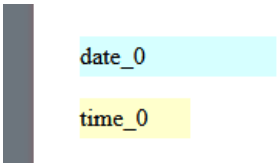

Fig. 111: A date and a time element

The goal is to automatically set both the date and the time as soon as the form is opened.

We start with the date. For that we select a *Temporal* E/A (fig. 112).

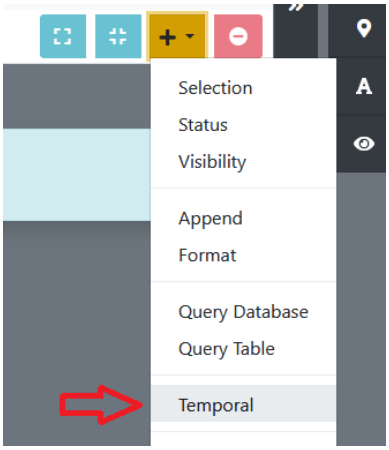

Fig. 112: New Temporal E/A

And we set the card, as showed in fig. 113.

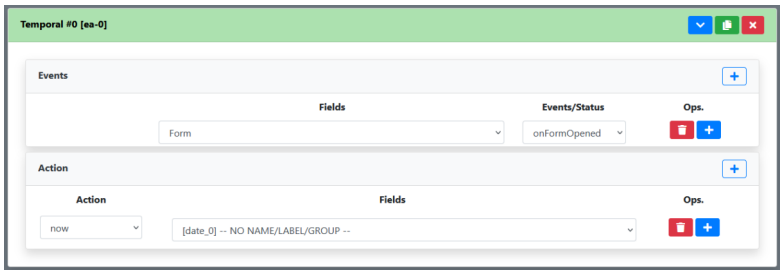

Fig. 113: Setting up a Temporal E/A for the date

Next, we add another *Temporal* card and filled for the time, as showed in fig. 114.

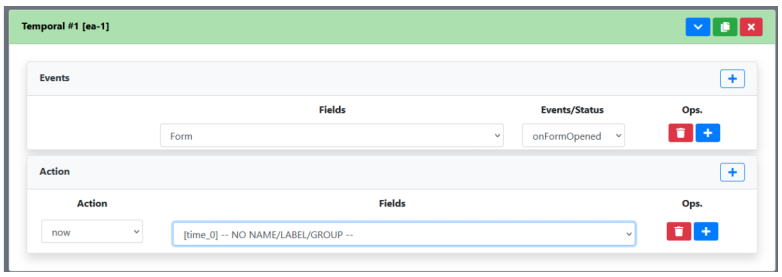

Fig. 114: Setting up a Temporal E/A for the time

Instead of two *Temporal* E/A cards it's possible to use a single card, as depicted in fig. 115.

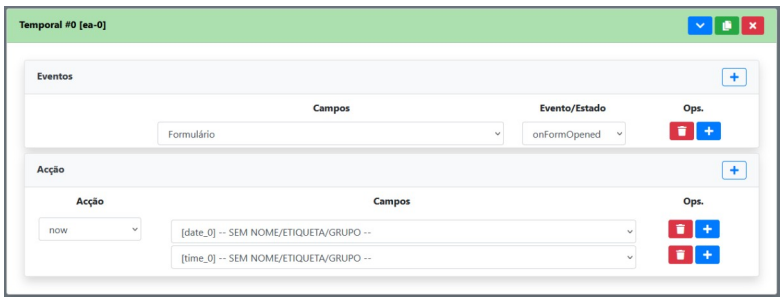

Fig. 115: Setting up a Temporal E/A for date and time in and single card

Previewing the result we observe that as soon the form is opened, both the date and the time, have the current date/time (see fig. 116).

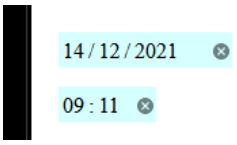

Fig. 116: Setting both the date and time when the form is opened

Be aware, that in the previous example saved values have precedence to the E/A action, i.e., the saved values will be maintained even if the operation is opened next day.

2.9.5.9 Required

In this example, we have two elements, a checkbox and a text element (fig. 117).

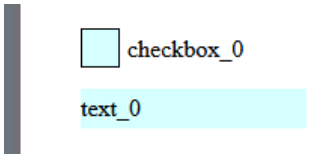

Fig. 117: A checkbox and a text element

Goal: when *checkbox_0* is checked, *text_0* is required to have a value. On the other hand, if *checkbox_0* is unchecked, *text_0* is no longer required to have a value.

We start by selecting a *Required* E/A (fig. 118).

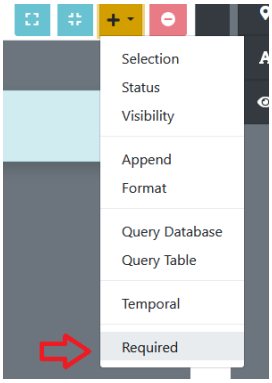

Fig. 118: New Required E/A

And we set the card, as showed in fig. 119, where we set that if checkbox_0 is selected, then text_0 must have a value.

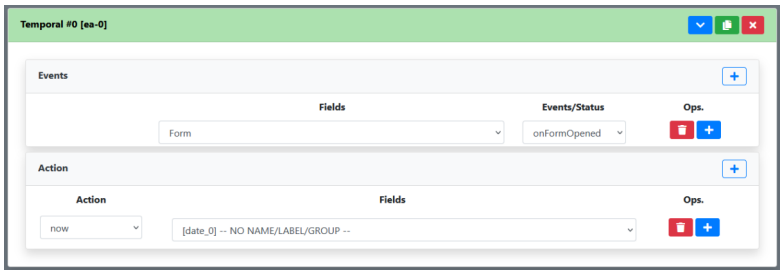

Fig. 119: Setting up a Required E/A for text_0

We now set the reverse: if checkbox_0 is not selected, then text_0 is not required to have a value (fig. 120).

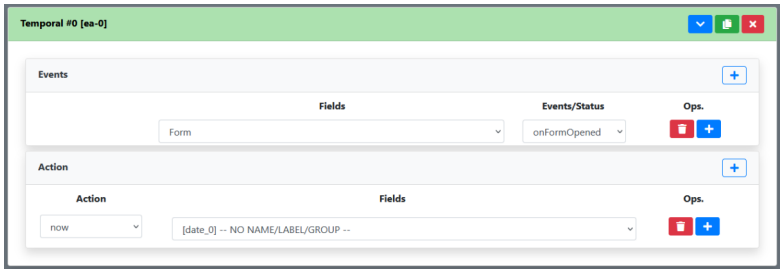

Fig. 120: Setting up a reverse Required E/A for text_0

Unfortunately it's not possible to watch this E/A in action the Preview. We must actually use it in an operation. The result can be seen in fig. 121.

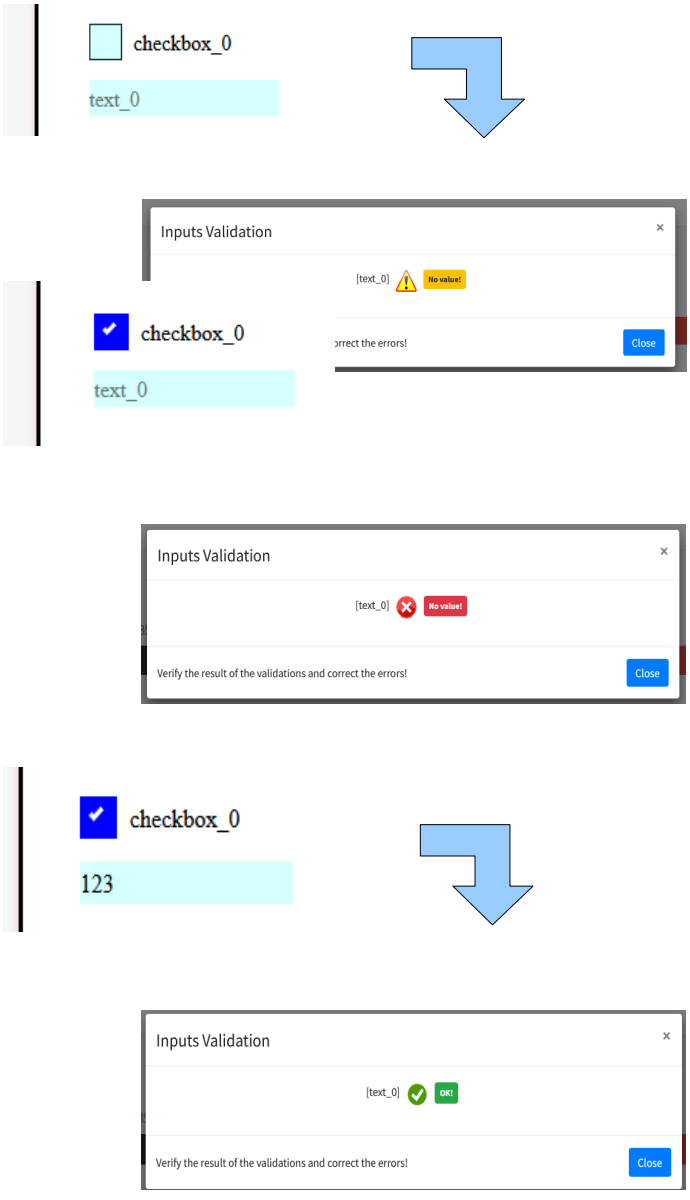

Fig. 121: Required E/A end result

2.10 Specifics

2.10.1 Headers and footers

2.10.1.1 FormView

See section 2.6.3 for this subject.

2.10.1.2 ListView

To create a header for the *ListView*, on only needs to set the *LV Header* property of the elements to appear in the header to *yes* (fig. 122).

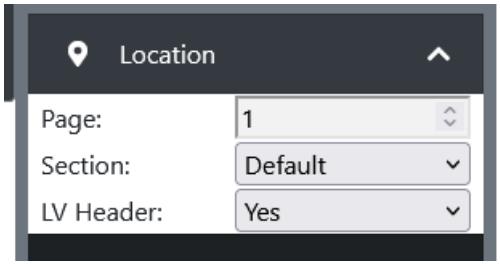

Fig. 122: The LV Header property

Note that only static elements can be part of the this header and for the moment it's only visible in *Edge, Opera* and *Chrome* like browsers. The creation process can be seen if fig. 123.

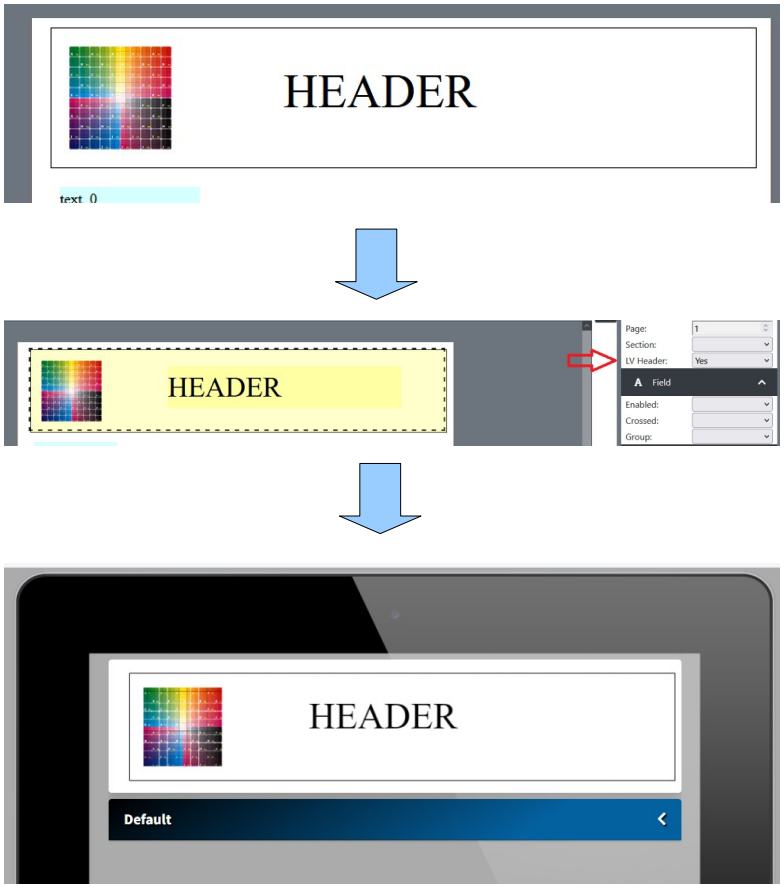

Fig. 123: Creating a header for the ListView

2.10.2 Options + "Other" – avoiding the Required E/A

Consider the typical example, with four checkboxes and a place to write a non-existent option (fig. 124). The “other option” is disabled by default and is required to *yes* (fig. 125).

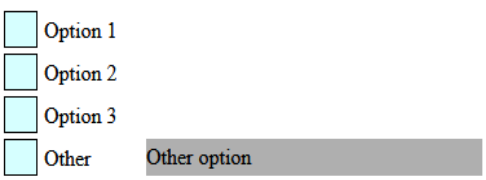

Fig. 124: The options and the other

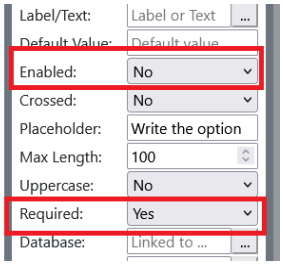

Fig. 125: Properties of the "Other option"

We start by creating two *Status* E/A to enable or disable the "*Other option*" depending on the status of the "*Other*" checkbox (figs. 126 and 127).

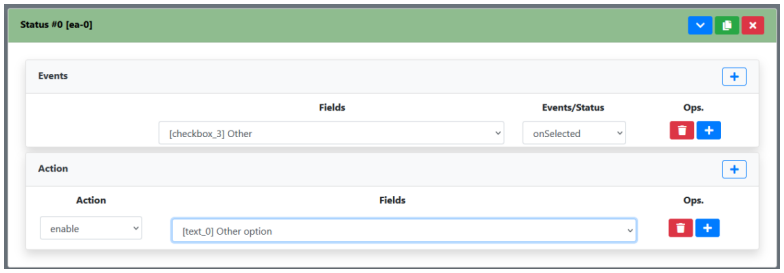

Fig. 126: Enable the input

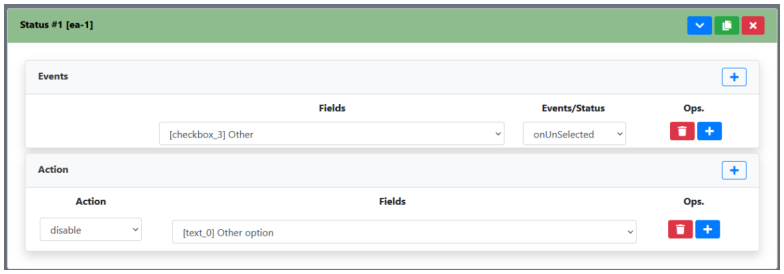

Fig. 127: Disable the input

The result can the see in fig. 128. Note that validations only work when the form is used in an operation.

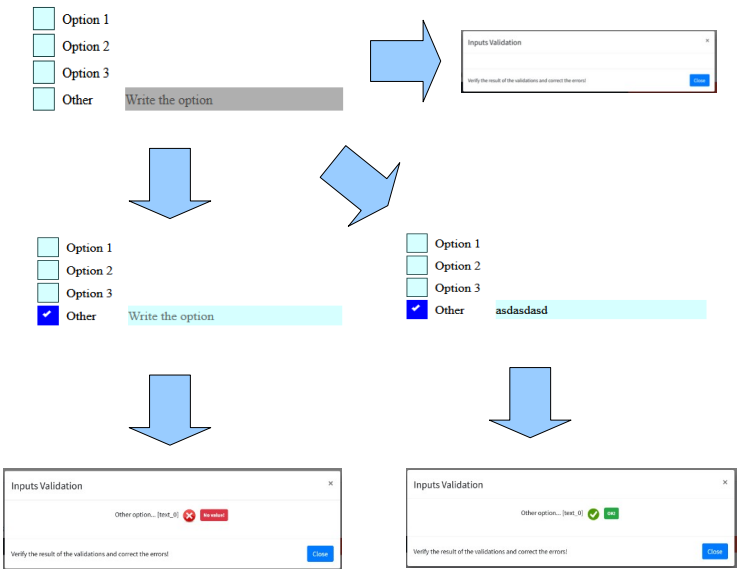

Fig. 128: Required end result

Since only visible and enabled elements are validated, if the "Other" checkbox is not selected, the required property of the "Other option" is ignored. We therefore don't need to use a *Required* E/A for these type of situations.

2.10.3 Database queries

2.10.3.1 Create a Query

For security reasons, only administrators can insert new database queries into the database.

The structure of a query follows the same as the targeted database service. Consider the following *Sql* query:

```
SELECT
   field_1, field_2, ...
from
   DATABASE.table_A
where
   field_x = X
or
   field_x = Y
...
```

MAKE SURE TO SPECIFY THE DATABASE OF THE TABLES IN THE QUERY!

All that is required is to place the fields of the *select* clause between **$** and the fields of the *where* clause (if any) between #:

```
SELECT
   $field_1$, $field_2$, ...
from
   DATABASE.table_A
where
   field_x = #X#
or
   field_x = #Y#
...
```

Usage example:

```
SELECT
   $name$, $address$, $country$, $phone$
from
   IDRISK.target
where
   nif = #X#
or
   nif = #Y#
```

In fig. 129 we can see the above query used in the E/A example.

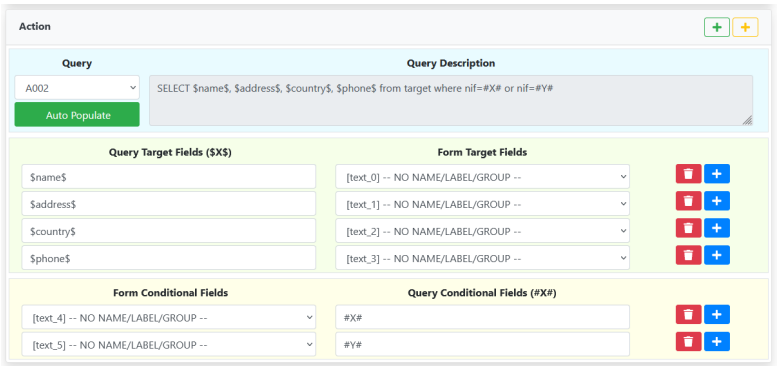

Fig. 129: Complete query in an E/A

While some of the *select* clause's fields can be ignored (fig. 130), the ones from the *where* clause must be all have some value, or an error will occur.

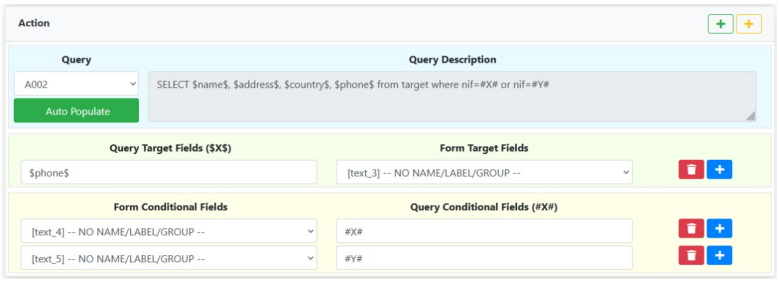

Fig. 130: Functional incomplete query in an E/A

2.10.3.2 Adding a Query to the Database

Onle an administrator can create and add queries to the database. All queries are set in the *Query* table. To add a query, the administrator should access the backoffice and access the Designer > Queries table (fig. 131). There he can add, remove and edit any query.

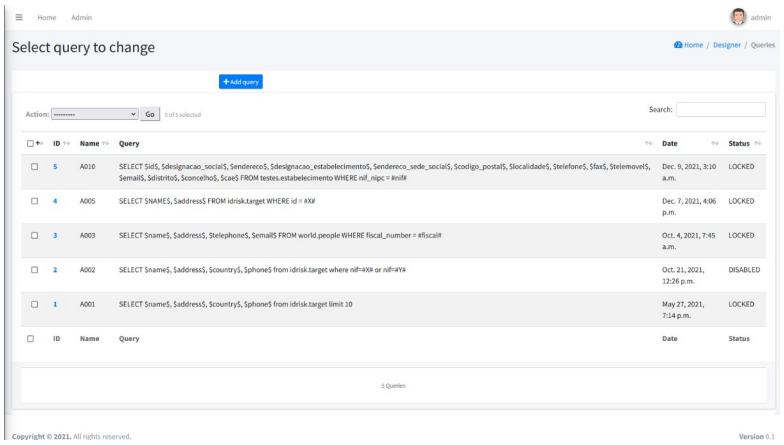

Fig. 131: The Queries tables

A query should be in one of two states: *LOCKED* (to be used) or *DISABLED* (no longer in use).

DO NOT DELETE A QUERY ONCE IT'S IN USE
OR WAS USED IN A FORM!
JUST SET IT TO DISABLED!

3 Preview

Preview allows the user to visualize, test and print the current opened form. By pressing the *Preview* button (fig. 132), a new tab will be opened where the form will be visualized inside a tablet frame (fig. 133), since it can help the user to experience what the form's user will experience.

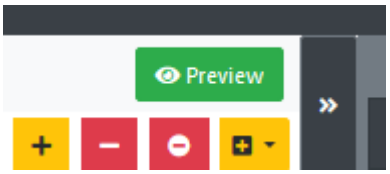

Fig. 132: Preview button in the Designer

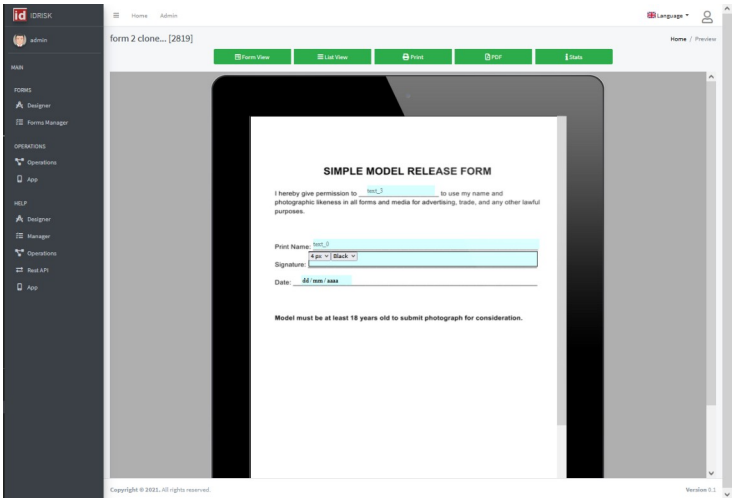

Fig. 133: Main view of the Preview

The form is automatically saved, every time it's previewed.

4 Forms Manager

All forms and their current status are managed by this application. It's composed by a single table (see fig. 134), which by default, displays: the forms IDs, their names, their description and a collection of buttons containing all possible actions that can be performed to any respective form. The table also contains several other relevant columns, which are hidden by default and can be displayed by selecting them in the *Columns Visibility* button (see fig. 135).

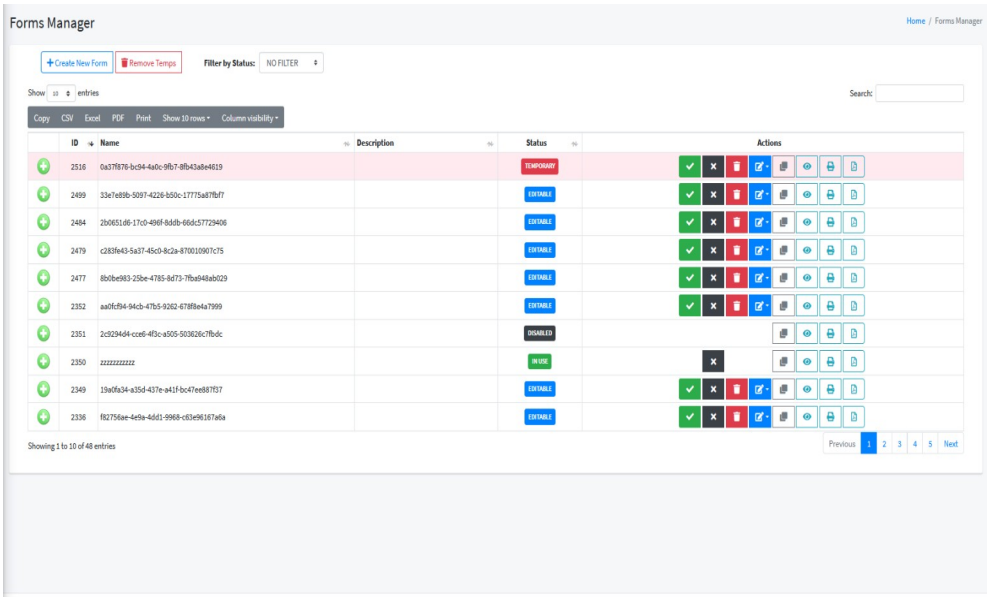

Fig. 134: Main view of the Forms Manager

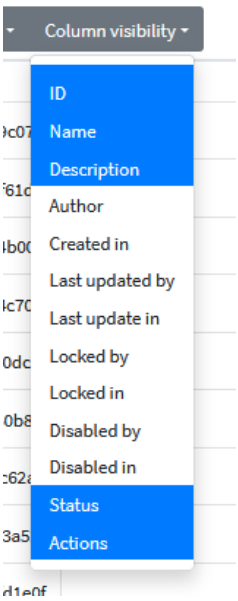

Fig. 135: Toggling the columns' visibility

4.1 Status

A form can be in any of four states: EDITABLE, IN USE, DISABLED, TEMPORARY. A form is in the TEMPORARY state, only when it hasn't being save it after its creation. After the first save, the form will be in the EDITABLE state. However, it's also possible to transition from TEMPORARY to IN USE or DISABLED.

In table 12, we have the 4 possible states and the respective available operations.

Operations	EDITABLE	IN USE	DISABLED	TEMPORARY
Can be Edited	X			X
Can be used in operations		X		
Can be deleted	X			X

Tab. 12: Form' state vs operation

The user can change a form's status by clicking the respective action button (table 13).

Button	Action
	Set to IN USE
	Disable form

Tab. 13: Available buttons to change a form's state

A form is in the **TEMPORARY** state, only when it hasn't being save it after its creation. After the first save, the form will be in the **EDITABLE** state. However, it's also possible to transition from **TEMPORARY** to **IN USE** or **DISABLED**.

Be warned that you cannot undo a change of state. For that, you will need to contact an administrator or clone the Form, which will create an exact copy of the form which will be in the **EDITABLE** state.

4.2 Actions

Figure 136 shows all available actions each operation has access to. However, the availability of these actions are dependent on the current status of the form, as depicted in table 14.

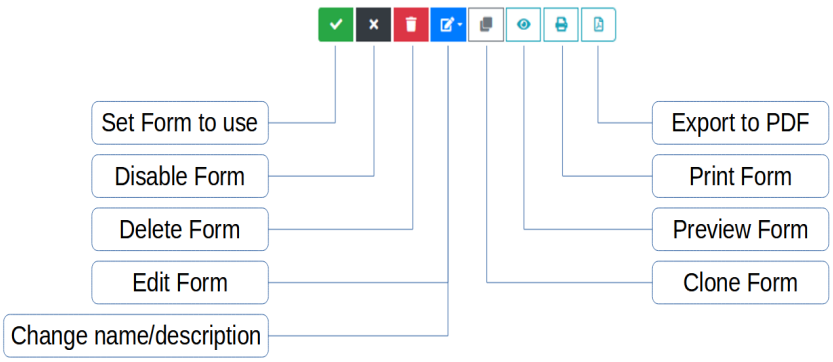

Fig. 136: All available actions

Operations	IN USE	DISABLE	DELETE	EDIT	CLONE	PREVIEW	PRINT	EXPORT TO PDF
EDITABLE	X	X	X	X	X	X	X	X
IN USE		X			X	X	X	X
DISABLED					X	X	X	X
TEMPORARY	X	X	X	X	X	X	X	X

Tab. 14: Actions vs Form state

Any change can't be undone. Only an administrator has that power. The admin can access the database through the Admin view (fig. 137).

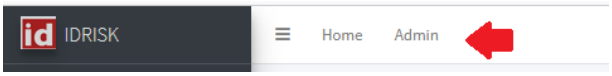

Fig. 137: Access to the Administrator views

There, the administrator has access to all relevant tables as we can see in fig. 138.

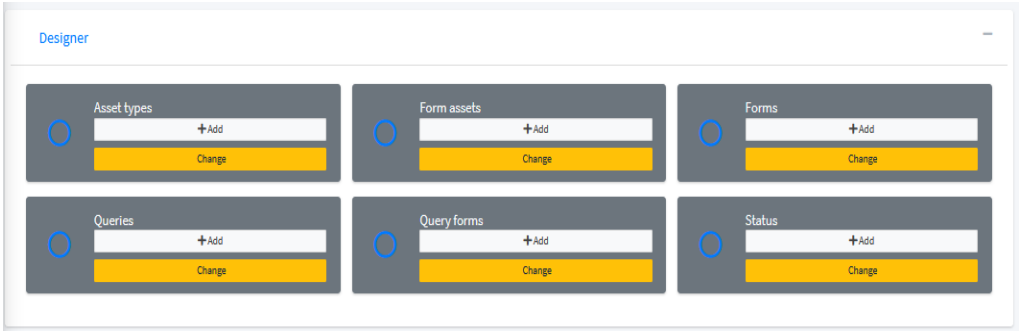

Fig. 138: Tables used by the Designer application

4.3 Assets

In the first column of the table, we can see a list of the assets (images and tables) that a form uses and visualize/download them by clicking them (fig. 139).

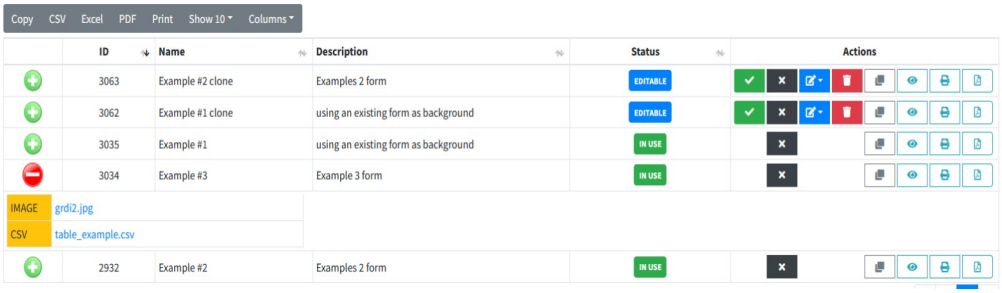

Fig. 139: List of all assets used by a form

Note that as long the form is EDITABLE, this list is composed by every asset that the form ever used, whether or not they are still in use. As soon the form is changed to IN USE, all unused assets are deleted.

4.4 Temporary Forms

The first time a form is created at the Designer, it will remain in the state TEMPORARY until the first save.

In case the user exits the Designer while the form is in this state, it should automatically be delete. However, there are occasions that temporary forms are not deleted. The user is then left with 3 options:

1. remove it at the manager by using the action button;
2. deleting all temporary forms through Remove Temps button;
3. allow the server to automatically remove them, as part of one of its daily tasks.

5 Operations Manager

It manages all operations and their current status. It's composed by a single table, listing all operations, regardless of their state. By default, this table displays: the operation ID, the operation name, its description, the original author and data of creation, the last update (author and date), the current status of the operation and a collection of buttons containing all possible actions that can be performed to the respective action (fig. 140).

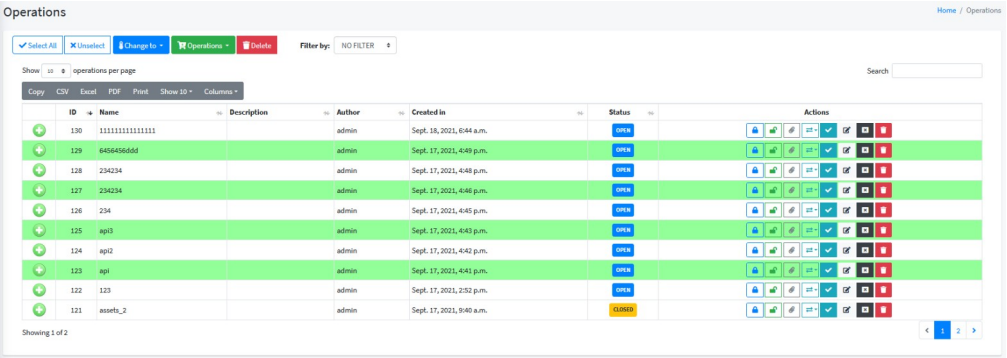

Fig. 140: Main view of the Operations Manager

5.1 Status

An Operation can be in one of three states:

- **OPEN**
- **CLOSED**
- **COMPLETED**

Any operation can be set into one of these states by clicking the respective action button (table 15).

Button	Action
	Set to OPEN
	Set to CLOSED
	Set to COMPLETED

Tab. 15: Possible states of any operation

The OPEN state represents an ongoing operation, i.e., the user is currently filling the respective form. If CLOSED, then the filling process is completed, and the user is happy with the result and with the final validations. COMPLETED, means that the operation cannot be changed any more, since it's already processed by the organization.

ONCE AN OPERATION IS SET TO COMPLETED, IT'S ONLY POSSIBLE TO CONSULT IT. IT'S NO LONGER POSSIBLE TO EDIT ANY PART OF THE OPERATION! ANY ATTEMPT, WILL BE MET WITH A 403 FORBIDDEN ERROR (fig. 64).

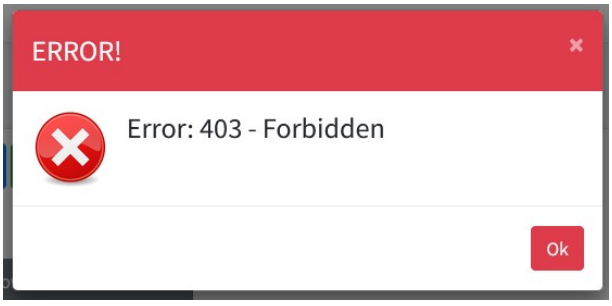

Fig. 141: When trying to do something forbidden ...

TO CHANGE THE STATUS OF A COMPLETED OPERATION, IT'S ONLY POSSIBLE BY AN ADMINISTRATOR THROUGH THE ADMINISTRATIVE PANEL OR DIRECTLY IN THE DATABASE.

5.2 Actions

Fig. 142 presents all available the actions to each operation.

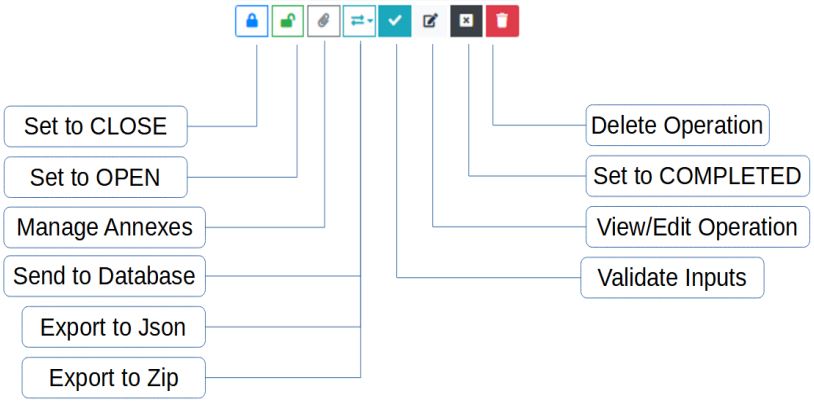

Fig. 142: Available actions to each operation

While Managers and Administrators have the capability to change the status of any operation to any state, it's highly recommended that once an operation is set as COMPLETED for not changing it back. The reason for this, has to do with unlikely synchronization issues with the App.

5.3 Global Actions

These are actions that are executed on one or many operations at a time (fig. 143):

- select all operations;
- unselect all selected operations;
- change all selected operations to state X;
- export the operation inputs into a json file or also include all assets (images, documents, annexes, etc) and export as a zip file.

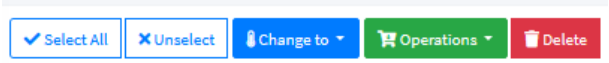

Fig. 143: Actions available to apply to 1 or more operations

To be applied, it's necessary to select the operations either by clicking each operation individually (which paints the row green) or by clicking *Select All*, which selects all operations currently on display (figs. 144, 145 and 146). These actions only become available once one or more operations are selected.

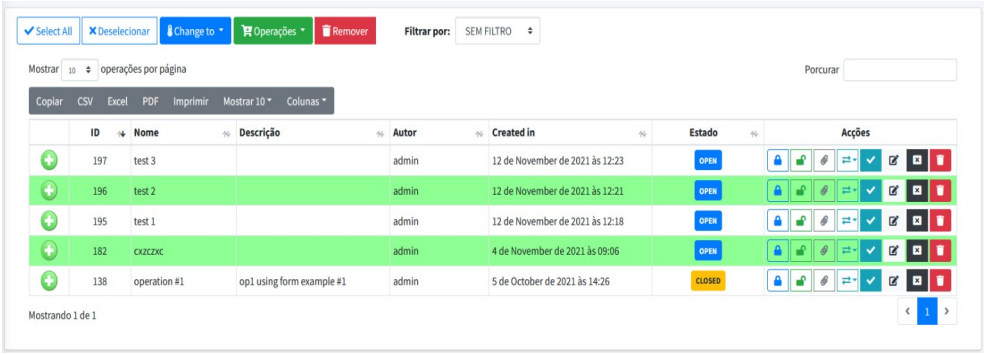

Fig. 144: Selecting multiple operations

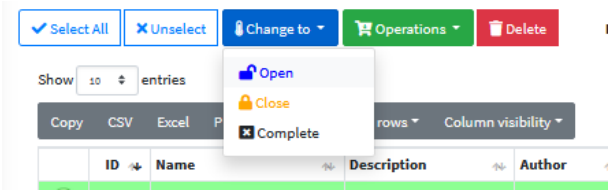

Fig. 145: Changing the states of multiple selected operations

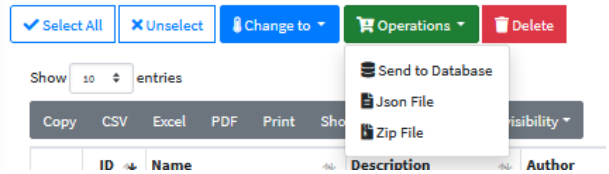

Fig. 146: What to do about multiple selected operations

5.4 Assets / Images

In the first column of the table, we are able to see a list of all assets, from inputs and annexes. It also provides an indication of whether they are annexes or user inputs, like documents and photos. We can also visualize or download them by selecting them (fig. 147).

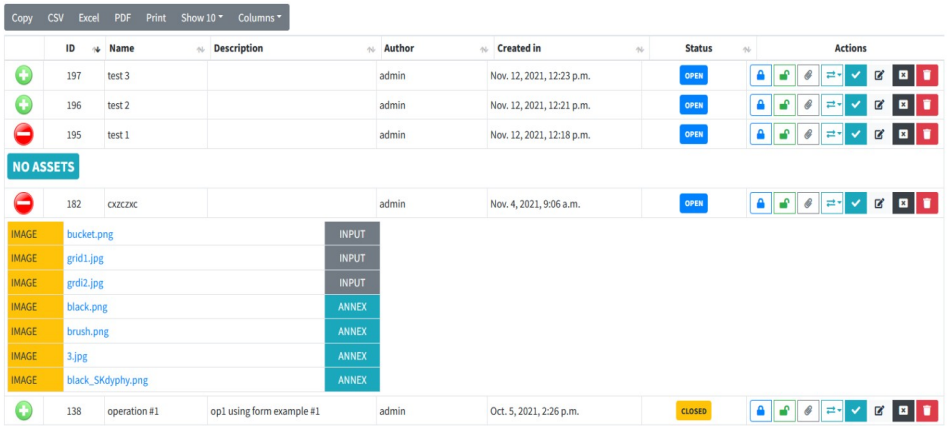

Fig. 147: Operations' assets and annexes

5.5 Edit/View Operations

By pressing the , button a new tab is created, where we can visualize, edit, manage annexes, print and export to PDF an operation (fig. 148).

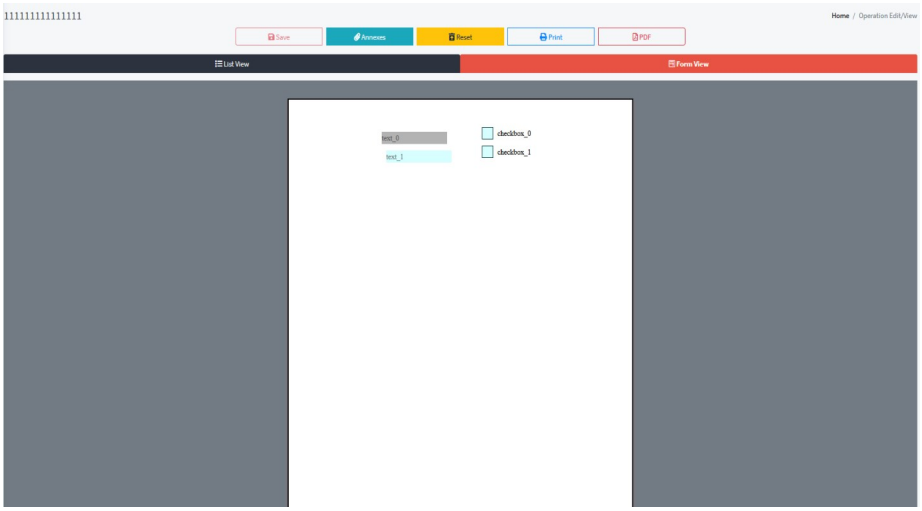

Fig. 148: Main view of the Operation Editor

5.6 Validate Operation Inputs

When validating an operation, the user's inputs will be analysed and verified against the validations required for each input as defined in the form. These validations include:

- No value
- Incorrect format
- No uppercase
- Max length exceed
- Max value exceeded
- Min value exceeded
- No value selected

The return of the validation process, is composed by two groups: Errors and Warnings (fig. 149). An error happens when a mandatory requirement is not fulfilled, for example, a phone number with an incorrect format. A warning happens when a value is not mandatory but it is missing or a value is not selected. When no errors or warnings are found, an OK message is presented (fig. 150).

Important remark: the system only validates enabled and visible elements.

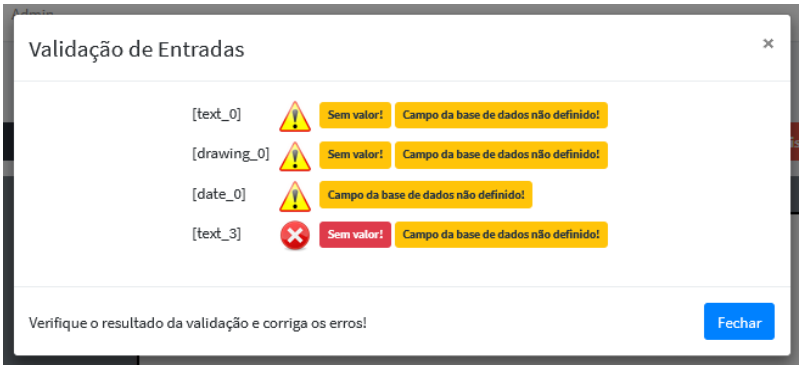

Fig. 149: Errors and warnings

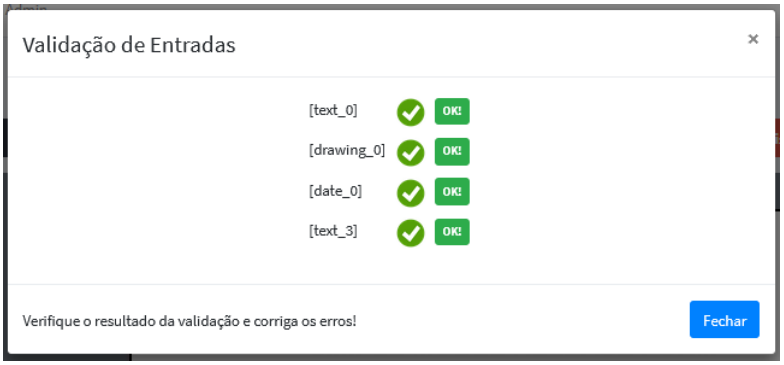

Fig. 150: All inputs OK

6 App

The Inspector's app, is a Progressive Web Application (PWA), which means it's designed to make the user feels like it's a native application and also, can operate without being connected to the web.

6.1 Installation

The App can be used either inside a browser or the user can "install" it in his platform. Once the user opens the App page in Chrome, a notification will appear asking if the user desires to install the app (see figs. 151 and 152).

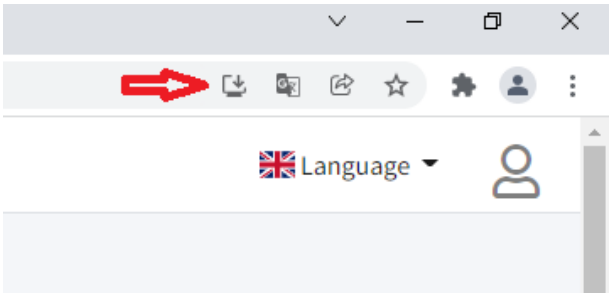

Fig. 151: Icon to install the App

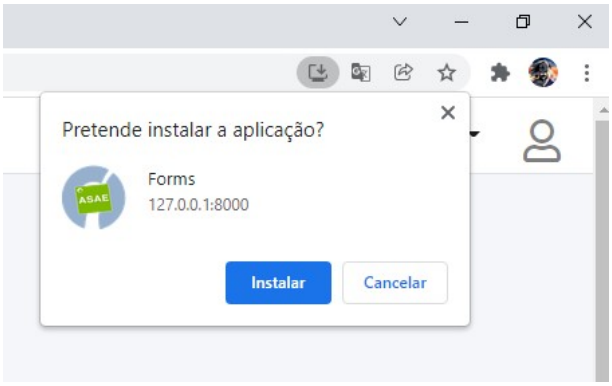

Fig. 152: Confirmation dialogue

An icon will then be installed in the device (fig. 153).

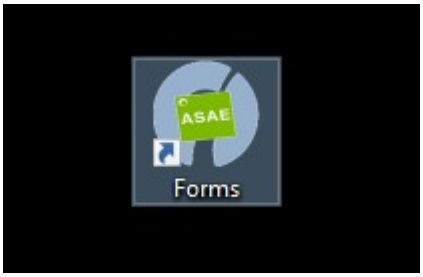

Fig. 153: Installed icon

Once it's installed, the user only has to click the icon, and a window will open the App page, as depicted in fig. 154.

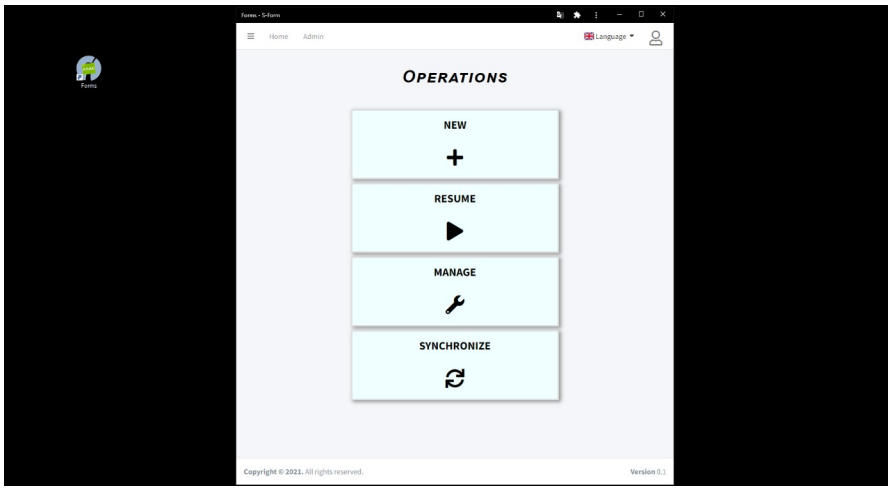

Fig. 154: App opened through the icon

6.2 Main Menu

The main screen is composed by a four buttons menu (fig. 155). Each operation function is described in table 16.

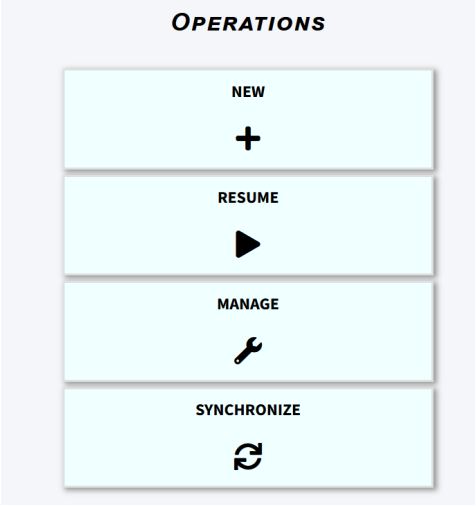

Fig. 155: Main screen

Button	Description
New	Creates a new operation.
Resume	Resumes any OPEN operation.
Manage	Manages all OPEN / CLOSED operations.
Synchronize	Forces synchronization between the server and the local app.

Tab. 16: Main menu

In figure 156, we present the screens transitions of the App.

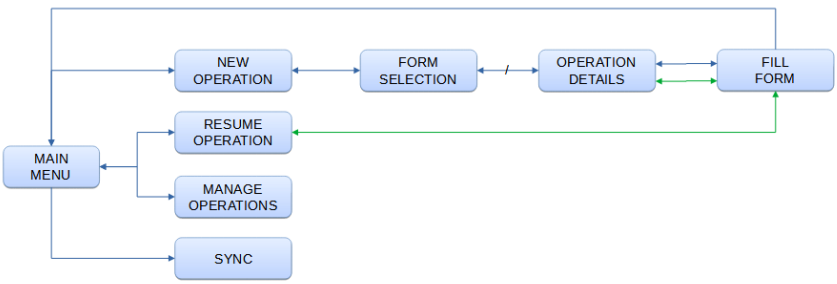

Fig. 156: Screens transitions

6.2.1 New Operation

Once a New Operation is selected /fig. 157), the user must select the form to be used in the operation (fig. 158).

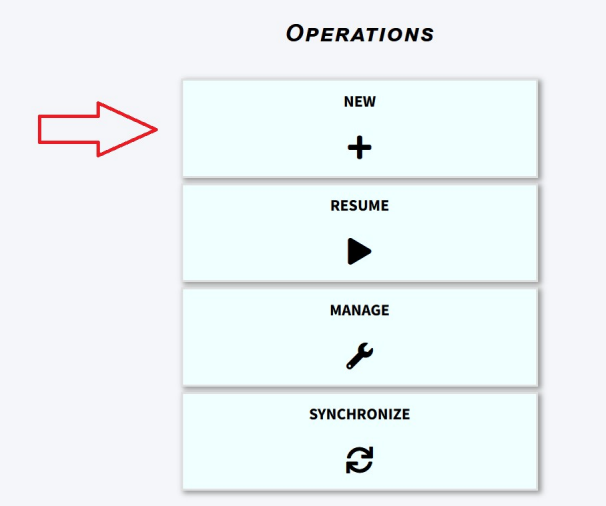

Fig. 157: New Operation

Fig. 158: Form Selection

Note that only forms with the current state as IN USE will be listed. In this screen, the user can also print an empty form or export it to pdf.

Once a form is selected, the user will have to fill in the operation details: *name* (mandatory) and *description* (fig. 159).

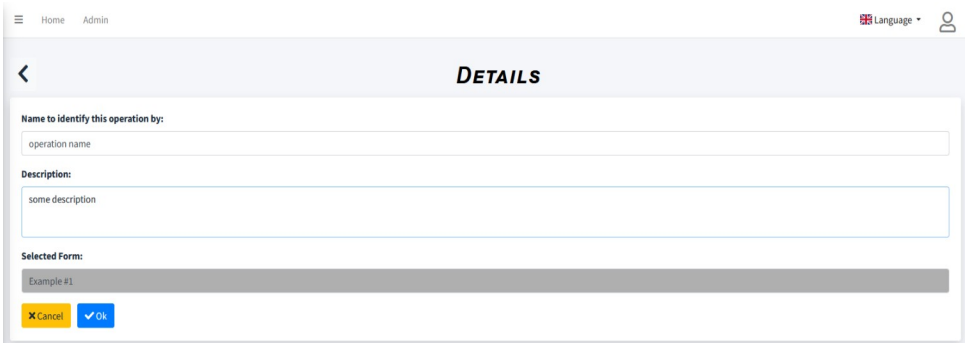

Fig. 159: Operation details

By pressing Ok, the form will be presented to the user and he can finally start filling it, either in the *Form View* (fig. 160) or in the *List View* (fig. 161). Since both views are synchronized, the user can jump between both views anytime.

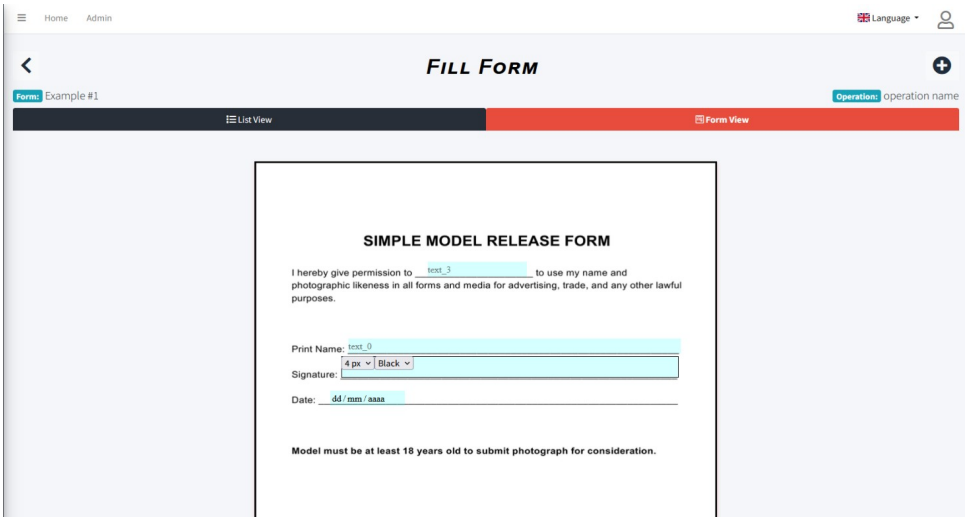

Fig. 160: Filling up the form in the Form View

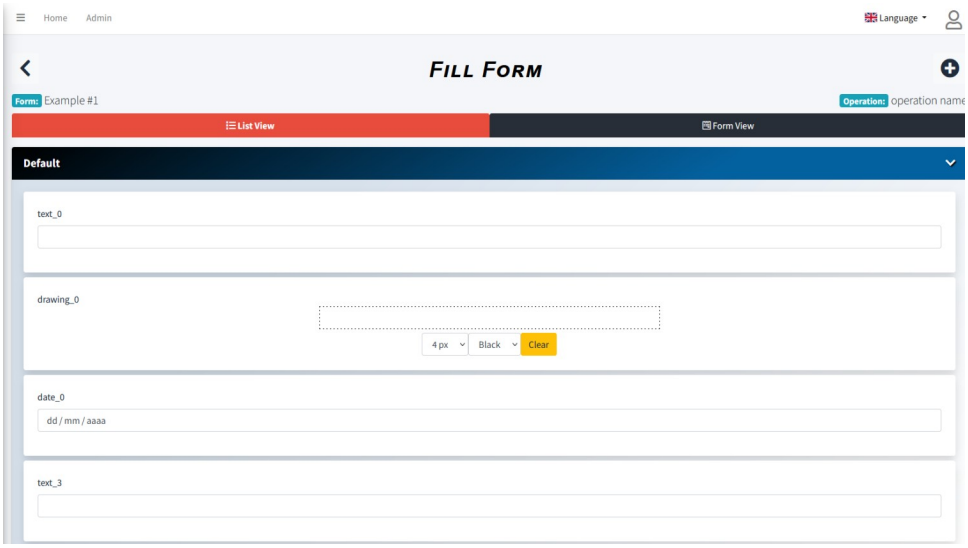

Fig. 161: Filling up the form in the List View

On the top row we have a back button, which will take the user back to the details pages and a menu button containing several options (fig. 162), which are described in table 17. The form's name and the operation's name.

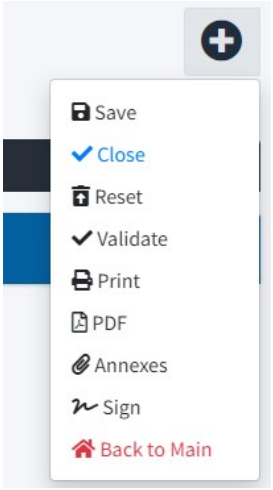

Fig. 162: Fill Form menu

Button	Description
Save	Saves the current operation's inputs
Close	Sets the operations OPEN to CLOSED to
Reset	Clears all the inputs
Validate	Validates all inputs according to the rules established in the form (fig. 163)
Print	Prints the form and current inputs (fig. 164)
PDF	Exports the form and current inputs to a PDF file
Annexes	Opens the annexes dialogue, allowing the user to add, remove and download documents, pictures, files, etc. (fig. 165)
Sign	Allows to sign the current operation with a smart card (fig. 166)
Back to Main	Returns the user to the main menu

Tab. 17: Fill form menu descriptions

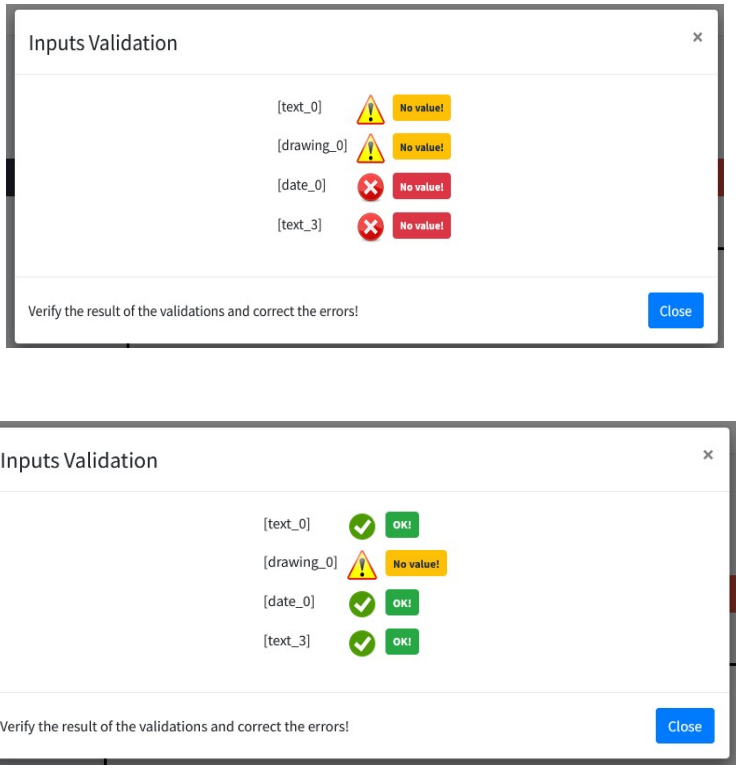

Fig. 163: Inputs validation

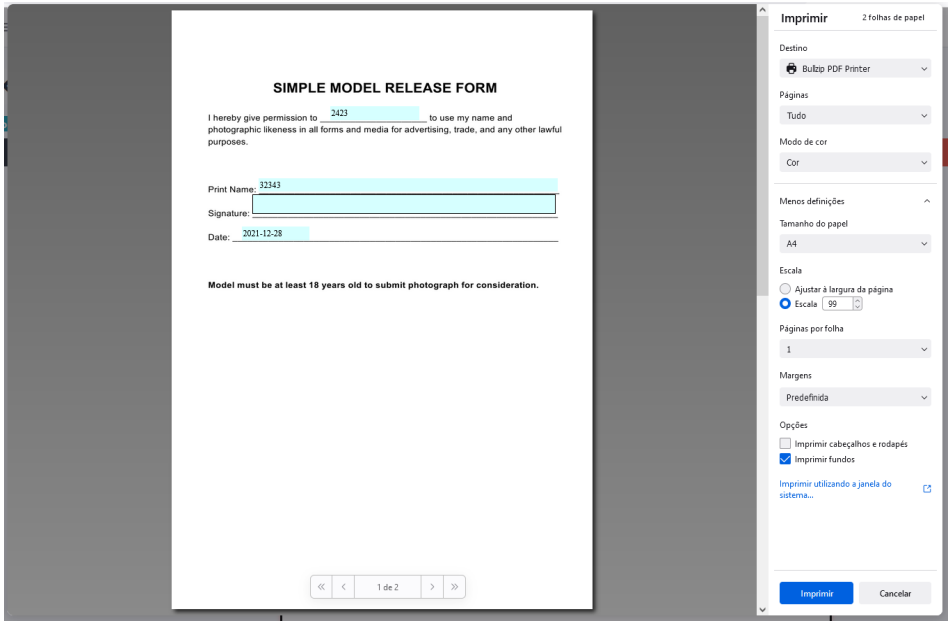

Fig. 164: Printing the form and its inputs

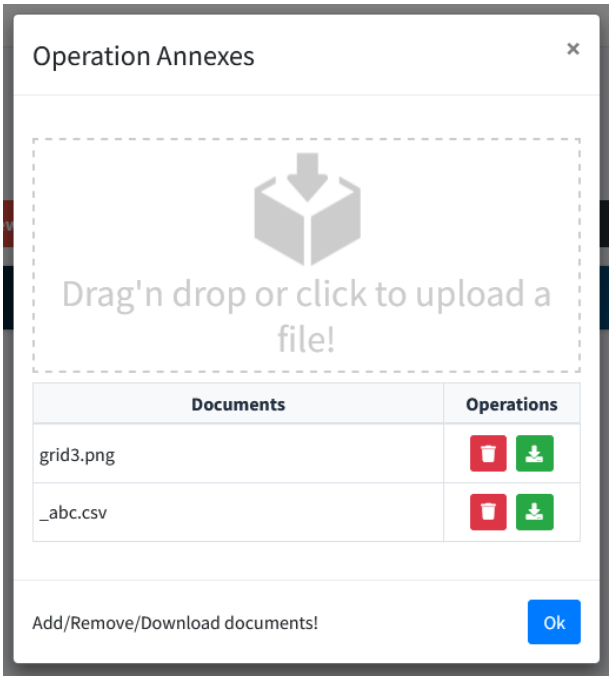

Fig. 165: Operation annexes dialogue

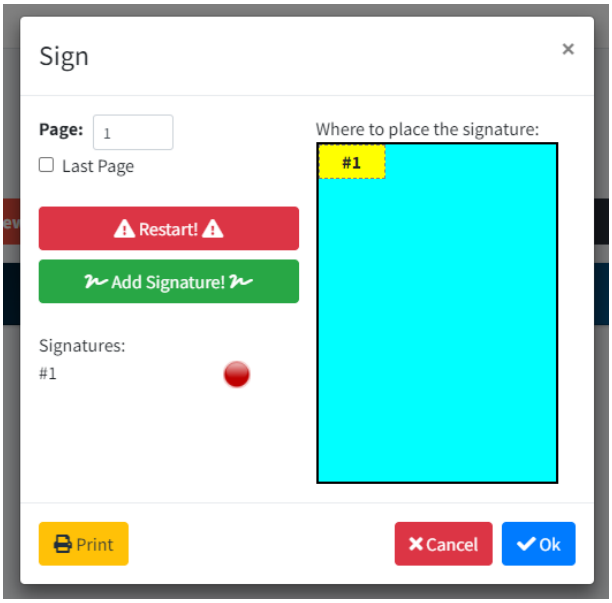

Fig. 166: Signing Modal

6.2.2 Resume Operations

Any OPEN operation can be resumed by the user at any time. These operations are listed in a table in the Resume Operations screen (fig. 167). A single click or touch will open the selected operation to be edited.

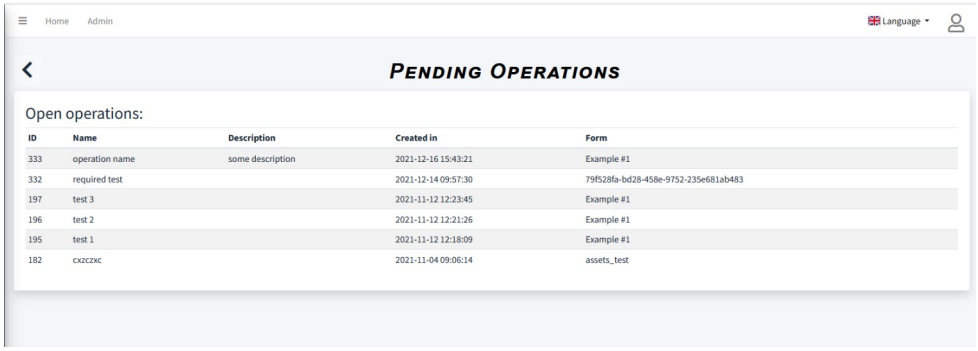

Fig. 167: Resume operations screen

6.2.3 Manage Operations

In the operations manager screen (fig. 168), the user can operate on all **OPEN** and **CLOSED** operations.

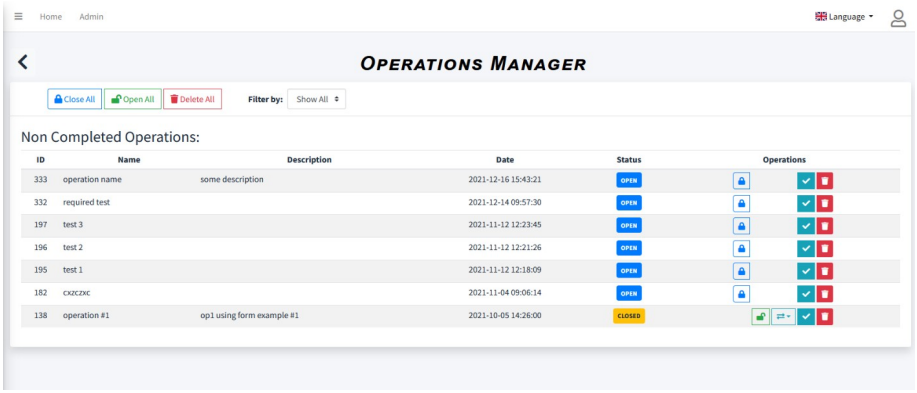

Fig. 168: Operations manager screen

There are three global actions:

- Close All – set the status of all operations to **CLOSED**.
- Open All – set the status of all operations to **OPEN**.
- Delete All – deletes all operations.

Each operation also has several actions associated with it (tab. 18). Their availability is dependent on the state of the operation.

Button	Action
	Sets to **OPEN**
	Sets to **CLOSED**
	Downloads inputs as a json file or as a zip file containing a json file and all input “images” and annexes.
	Validates operation inputs
	Deletes operation

Tab. 18: Possible states of any operation

6.2.4 Synchronization

For the App to operate when there’s not network, the app requires constant synchronization with the server. When the user open the App, it will automatically attempt to synchronize with the server. In this case a visual indication will be presented to the user on this fact (fig. 169).

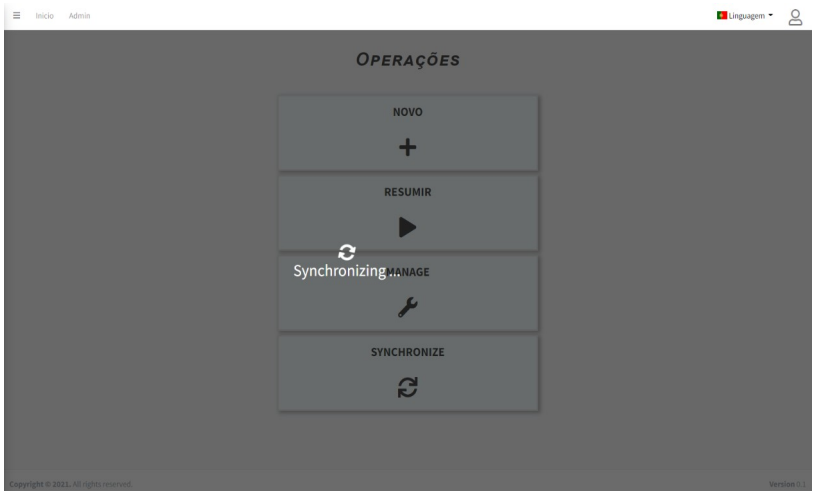

Fig. 169: App synchronizing

The user also has the option by manually triggering this synchronization by clicking the “Synchronization” button in the main screen. If the App is online, then it will sync, otherwise, it will present a warning message to the user (fig. 170).

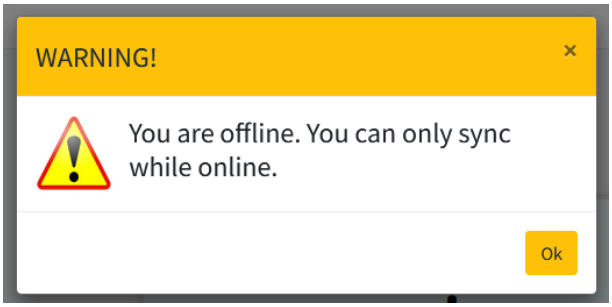

Fig. 170: Offline warning

6.3 Map

By either clicking either on the magnifying glass icon or on the image the coordinates are automatically fetched (fig. 171).

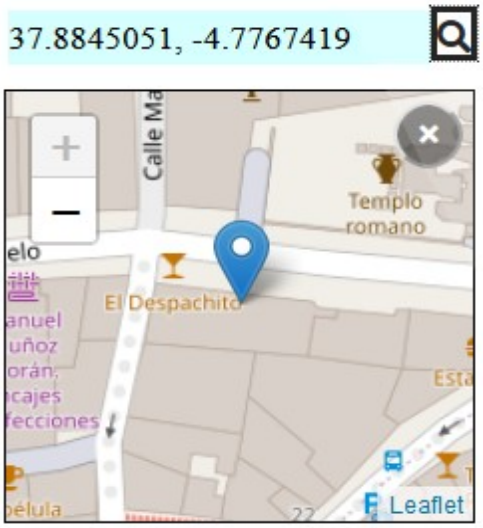

Fig. 171: Map result

6.4 Barcode Reader

There are 2 types of elements to read/input barcodes. One is only for text and the other for images (see fig. 172).

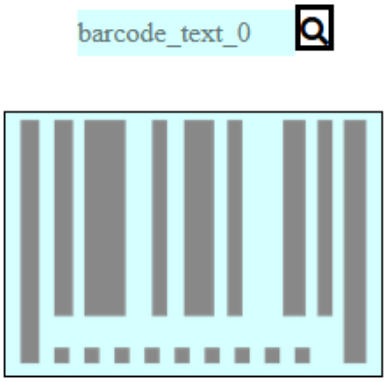

Fig. 172: Barcode elements

The barcode modal (fig. 173) appears by either clicking in the magnifying glass icon or the barcode image. While both can be used to automatically detect barcodes from the camera, only the text one can be used for manual input (fig. 174).

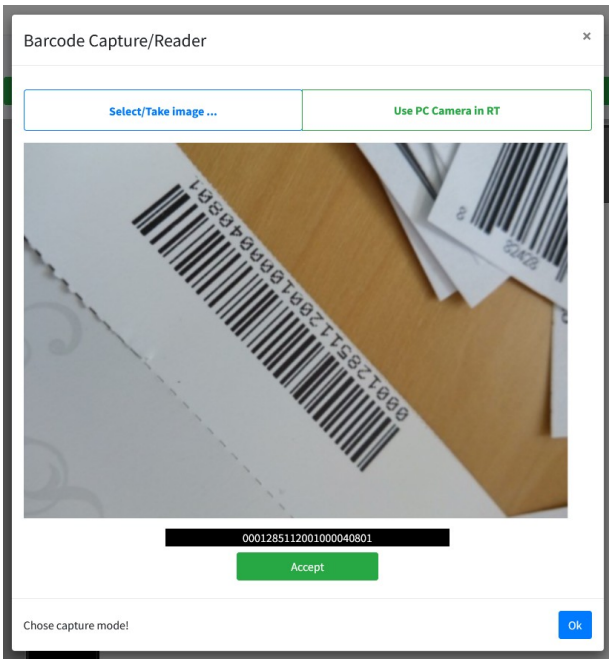

Fig. 173: Barcode Modal

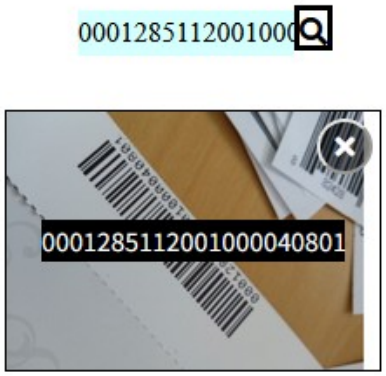

Fig. 174: Barcode values

6.5 Foodex

The Foodex modal appears by clicking in the magnifying glass icon of the element (fig. 175).

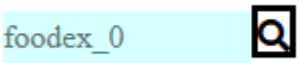

Fig. 175: Foodex element

The selection is done through a modal that displays the core Foodex2 hierarchy (fig. 176).

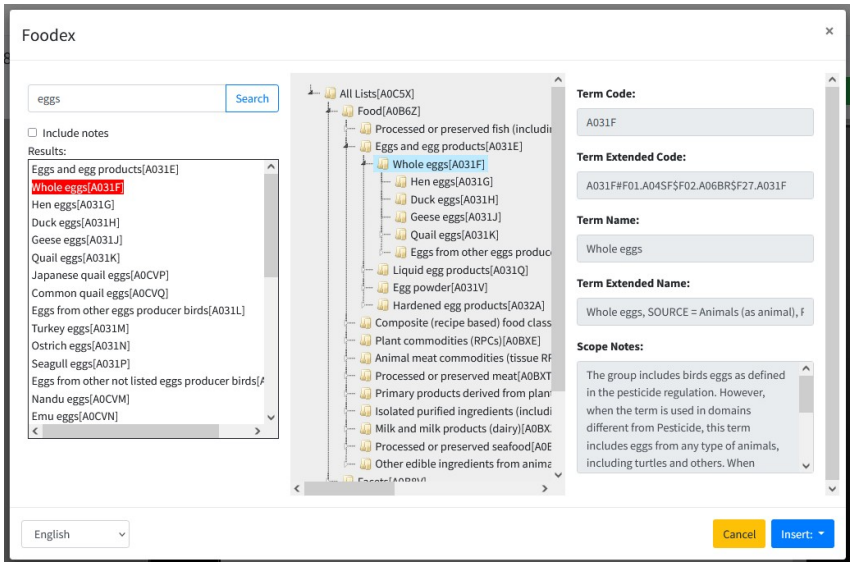

Fig. 176: Foodex modal

After selection, the user can chose to insert one of the following data (fig. 177):

- term code
- term extended code
- term name
- term extended name

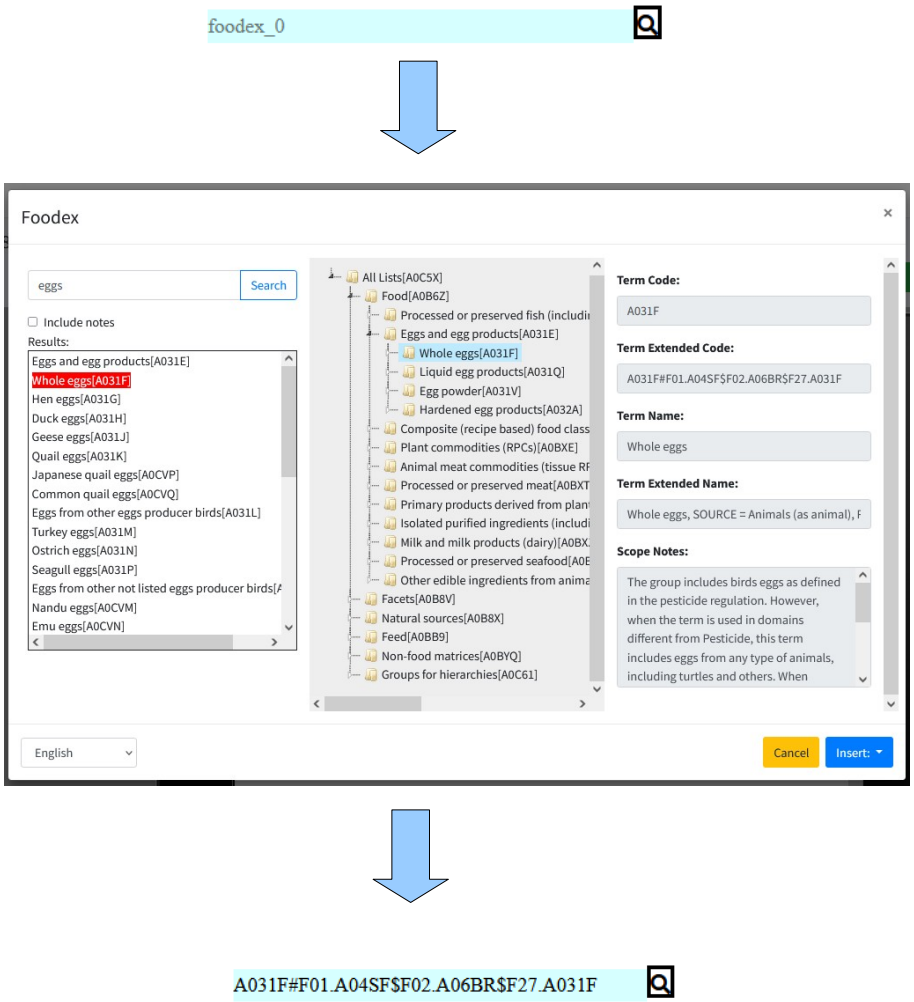

Fig. 177: Foodex Element

6.6 Digitally Signing an Operation

6.6.1 Requirements

Currently signing is only available on Windows (64 bits) and using the Chrome browser. For mobile devices is still under development.

Requirements (select *APPLICATION* in the sidebar to open the Application page – figure 178):

- Aplicação Autenticação.gov (https://www.autenticacao.gov.pt/web/guest/cc-aplicacao);
- Extensão chrome (https://chrome.google.com/webstore/detail/pdf-pt-cc-signing/agmijmlopfbbgdbliclncpnddhhdmodo). For manual installation, download the extension directly from ({server}/media/files/chrome_extension.zip);
- Aplicação *CCPDFSign* ({server}/media/files/CCNativeApp.exe).

Notes:

- the installation of *CCPDFSign* requires administrator rights;
- the chrome extension is not publicly listed at the Chrome store.

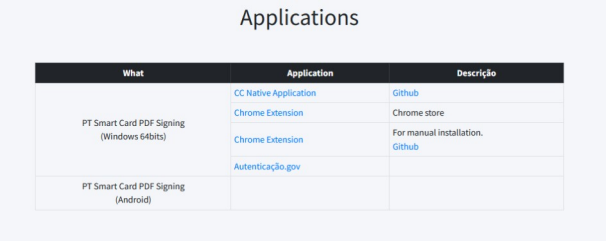

Fig. 178: Applications

6.6.2 Process

The signing of an operation is done through the Sign Modal (fig. 179).

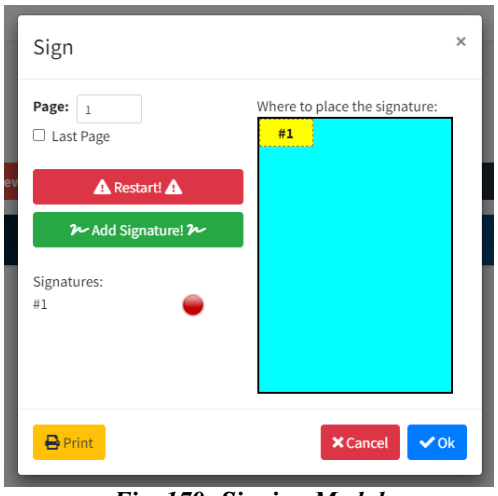

Fig. 179: Signing Modal

The procedure is straightforward:

1. Select the page (or the last page);
2. Drag the yellow rectangle to the position in the page where the signature is to be placed;
3. Click on *"Add Signature"*;
4. A dialogue will appear asking for the pin (fig. 180). Note that by default, the user has 30 seconds to complete the operation: Add signature → pin insertion → Ok, otherwise a warning dialogue will pop up. However, this will not affect the operation at all, and the user may close the dialogue and continue;
5. If everything goes alright, a green icon will appear on the right of the signature number and the yellow rectangle will also become green (fig. 181);
6. Repeat steps 3 to 5 as many times as necessary (fig. 182);
7. Clicking "*Ok*", will annex the signed operation as a PDF document, named **DOC_SIGNED.pdf**.

Fig. 180: Pin dialogue

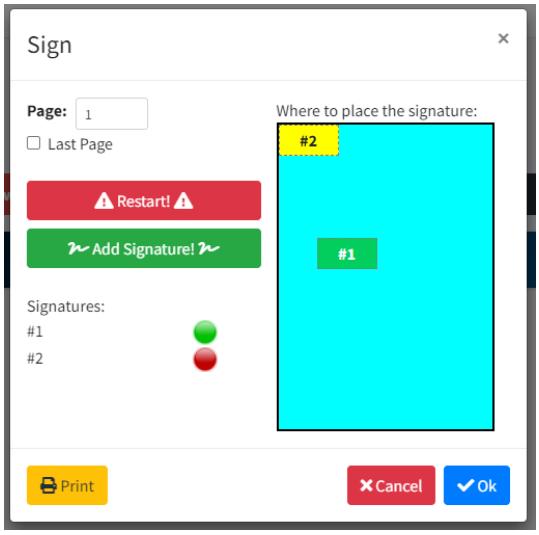

Fig. 181: A signature was added successfully.

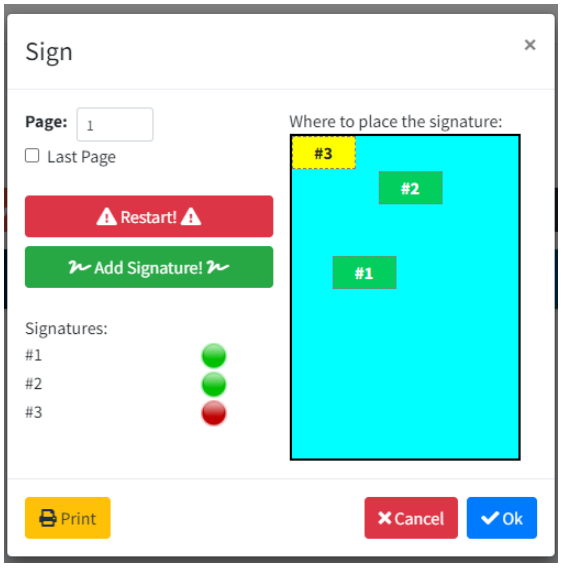

Fig. 182: Multiple signatures.

When the user opens the modal and if an operation is already signed, a warning message will pop up, indicating that the operation is already signed (fig. 183). This check is done against the existence of a **DOC_SIGNED.pdf** file. The user is left with 2 choices: either deletes that file or continues and the new signed document will be named **DOC_SIGNED_*.pdf**., where * indicates a random sequence of characters.

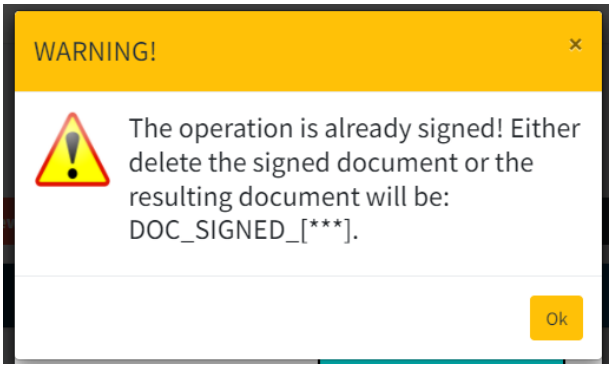

Fig. 183: Warning message in case the user attempts to sign an already signed operation.

Pressing "*Restart"*, will clear all current signatures. It will not affect any already signed documents.

In PC/Windows, the signing operation requires the user to install both an application and a Chrome extension. In case any of these are not installed, a warning message will pop up.

6.7 Online / Offline Operations

As stated in the previous sections, the App operates both online and offline. For that, is recommended for the user to open its App in a network covered place before going to a non-covered place and wait for the synchronization to conclude. This will

ensure the user will have the most up to date forms and operations in local storage. It also serves as a security measure for any critical incident that can destroy all local records.

All actions are functional, regardless of the network status, so the user doesn't have to worry about the functionality of the operation. However, there is an element that will not work while offline: the map / gps element. So be aware of this restriction.

6.8 Inputs Validations

When the user validates the inputs, he can be confronted with three types of messages for each input:

- OK
- WARNING
- ERROR

OK means that the input complies with all restrictions, such as, maximum length and right format.

Warning also means that the input complies with all restrictions but it could be better. Warnings are usually associated with non-mandatory empty fields.

Error means that at least one restriction is unfulfilled, and the user should correct it, before closing the operation.

Errors / Warnings:

- No Value!
- Incorrect format/pattern!
- No uppercase!
- Max length exceed!
- Max value exceeded!
- Min value exceeded!
- No option selected!

6.9 Printing Notes

While there is no browser recommendation for a good PDF export, on the other hand, for printing a form it's recommended to be performed using the *Chrome* browser. If *Firefox* is used, then it might be required to do minor adjusts to the scale. However this is usually only required for those forms containing *boxes*, *circles* and *lines*.

7 REST API

An API is provided as an easy way for an external application or tool to control and manage operations. All endpoints require a token, which only users with a manager or above privileges can obtain.

7.1 Token generation

POST {server}/rest/api/api-token-auth/

Token generation for authentication. ***REQUIRED FOR ALL ENDPOINTS***.

REQUEST BODY SCHEMA:

Field	*Type*	*Description*
username	**string**	User's username
password	**string**	User's password

RESPONSES:

200 Successful
The token, user ID and email.

400 Bad Request
Missing body parameters. Response example:

```
{
  "username": [
    "This field is required."
  ],
  "password": [
    "This field is required."
  ]
}
```

Wrong credentials. Response example:

```
{
  "non_field_errors": [
    "Unable to log in with provided credentials."
  ]
}
```

401 Unauthorized
No permission to generate the token.

```
{
  "message": "You do not have permission to generate a token"
}
```

EXAMPLE:

REQUEST

{server}/api-token-auth/

BODY:

```
{
  "username": "myusername",
  "password": "mypass"
}
```

RESPONSE

```
{
  "token": "token_here",
  "user_id": 1,
  "email": "some@email.com"
}
```

7.2 List all Operations

GET {server}/rest/api/list_operations/

Get a list of all operations, independently of their status, or an empty array if none.

HEADER PARAMETERS:

Field	Type	Description
Authorization	-	Token [token]

RESPONSES:

200 Successful
List of all operations.

403 Forbidden
Invalid token or Authentication credentials were not provided.

RESPONSE FIELDS DESCRIPTION:

Field	Type	Description
id	**integer**	Operation's ID
name	**string**	Operation's name
description	**string**	Operation's description
form	**integer**	Form's ID used in the operation
date_creation	**date/time**	Operation's Date/Time of creation
date_updated	**date/time**	Operation's Date/Time of the last update
author	**integer**	Original operation's author ID
author_name	**string**	Original operation's author name
updated_by	**integer**	ID of the user responsible for the last update
updated_by_name	**string**	Name of the user responsible for the last update
status_name	**string**	Current status of the Operation: OPEN CLOSED COMPLETED

EXAMPLE:

REQUEST

{server}/rest/api/list_operations/

RESPONSE


```
[
  {
    "id": 87,
    "name": "operation 1",
    "description": "dfsfsd",
    "form": null,
    "date_creation": "2021-09-05T18:37:54.804253Z",
    "date_updated": "2021-09-06T09:06:45.706464Z",
    "author": 1,
    "author_name": "admin",
    "updated_by": 1,
    "updated_by_name": "admin",
    "status_name": "OPEN"
  },
  {
    "id": 93,
    "name": "operation 3",
    "description": "",
    "form": 2838,
    "form_name": "8f295735-e760-4014-ba18-21c4d70ab071",
    "date_creation": "2021-09-06T09:06:00.650324Z",
    "date_updated": "2021-09-06T09:14:10.755081Z",
    "author": 1,
    "author_name": "admin",
    "updated_by": 1,
    "updated_by_name": "admin",
    "status_name": "COMPLETED"
  },
]
```

7.3 List all non-Completed Operations

GET {server}/rest/api/list_non_completed_operations/

Get a list of all **OPEN** and **CLOSED** operations or an empty array if none exists.

HEADER PARAMETERS:

Field	Type	Description
Authorization	-	Token [token]

RESPONSES:

200 Successful
List of all non completed.

403 Forbidden
Invalid token or Authentication credentials were not provided.

RESPONSE FIELDS DESCRIPTION:

Field	Type	Description
id	**integer**	Operation's ID
name	**string**	Operation's name
description	**string**	Operation's description
form	**integer**	Form's ID used in the operation
date_creation	**date/time**	Operation's Date/Time of creation
date_updated	**date/time**	Operation's Date/Time of the last update
author	**integer**	Original operation's author ID
author_name	**string**	Original operation's author name
updated_by	**integer**	ID of the user responsable for the last update
updated_by_name	**string**	Name of the user responsable for the last update
status_name	**string**	Current status of the Operation: OPEN CLOSED

EXAMPLE:

REQUEST

{server}/rest/api/list_non_completed_operations/

RESPONSE


```
[
    {
        "id": 87,
        "name": "operation 1",
        "description": "dfsfsd",
        "form": null,
        "date_creation": "2021-09-05T18:37:54.804253Z",
        "date_updated": "2021-09-06T09:06:45.706464Z",
        "author": 1,
        "author_name": "admin",
        "updated_by": 1,
        "updated_by_name": "admin",
        "status_name": "OPEN"
    },
    {
        "id": 94,
        "name": "ope 4",
        "description": "",
        "form": 2838,
        "form_name": "8f295735-e760-4014-ba18-21c4d70ab071",
        "date_creation": "2021-09-06T09:36:53.510211Z",
        "date_updated": "2021-09-06T09:36:56.486207Z",
        "author": 1,
        "author_name": "admin",
        "updated_by": 1,
        "updated_by_name": "admin",
        "status_name": "OPEN"
    },
]
```

7.4 Operation

GET {server}/rest/api/get_operation/{operation_id}/

Get the details of a specific Operation, or an empty array if *operation_id* does not exist.

PATH PARAMETERS:

Field	Type	Description
operation_id	**number**	ID of the operation

HEADER PARAMETERS:

Field	Type	Description
Authorization	-	Token [token]

RESPONSES:

200 Successful
Operation details.

403 Forbidden
Invalid token or Authentication credentials were not provided.

RESPONSE FIELDS DESCRIPTION:

Field	Type	Description
id	**integer**	Operation's ID
name	**string**	Operation's name
description	**string**	Operation's description
form	**integer**	Form's ID used in the operation
date_creation	**date/time**	Operation's Date/Time of creation
date_updated	**date/time**	Operation's Date/Time of the last update
author	**integer**	Original operation's author ID

author_name	**string**	Original operation's author name
updated_by	**integer**	ID of the user responsible for the last update
updated_by_name	**string**	Name of the user responsible for the last update
status_name	**string**	Current status of the Operation: **OPEN** **CLOSED** **COMPLETED**

EXAMPLE:

REQUEST

{server}/rest/api/get_operation/96

RESPONSE


```
[
  {
    "id": 96,
    "name": "op8",
    "description": "",
    "form": 2838,
    "form_name": "8f295735-e760-4014-ba18-21c4d70ab071",
    "date_creation": "2021-09-06T09:40:05.287208Z",
    "date_updated": "2021-09-06T09:40:21.040743Z",
    "author": 1,
    "author_name": "admin",
    "updated_by": 1,
    "updated_by_name": "admin",
    "status_name": "OPEN"
  }
]
```


7.5 Operation Data

GET {server}/rest/api/get_operation_data/{operation_id}/

Get the data (inputs and state) of a specific Operation, or an empty array if *operation_id* does not exist or there's no data (blank form maybe?).

All assets (images and documents) are accessible at `{server}/media/operation_assets/{operation_id}/{asset_file}`.

PATH PARAMETERS:

Field	Type	Description
operation_id	**number**	ID of the operation

HEADER PARAMETERS:

Field	Type	Description
Authorization	-	Token [token]

RESPONSES:

200 Successful
Operation details.

403 Forbidden
Invalid token or Authentication credentials were not provided.

RESPONSE FIELDS DESCRIPTION:

`Global:`

Field	Type	Description
id	**integer**	Operation's Data ID
operation	**integer**	Operation's ID
data	**json**	Data and status of each interactable field. Three main sections: • *elements*: state and input value of each field • *extras*: extra information • *annexes*: list of annexed files

elements:

Field	Type	Description
id	**string**	Element's ID
value	**string / json (*)**	Value inputted by the user. It's type depends on the element's type.
enabled	**boolean**	Is the element active?
crossed	**boolean**	Is the element crossed?
visible	**boolean**	Is the element visible?
selected	**boolean**	Is the element selected? (only applicable to CHECKBOXES and RADIOS elements)
n_cols	**number**	Number of columns (only in TABLES elements)
n_rows	**number**	Number of rows (only in TABLES elements)

extras:

Currently the only extra information for this field is to indicate whether or not a field is required or not. This data is originated by the *Required* E/A and not by the *required* property that each field possesses.

```
"extras": {
  "ELEMENT_ID": {
    "REQUIRED": true | false
  },
  ...
},
```

annexes:

Simple list of filenames:

```
"annexes": [
  FILENAME_1, FILENAME_2, ...
],
```

value (each element contains this field, which depends on the element's type):

Element Type	Value type	Description
TEXT	**string**	-
EMAIL	**string**	-
PHONE	**string**	-
WEBSITE	**string**	-

TEXTBOX	**string**	-
NUMBER	**string**	-
DROPDOWN	**string**	-
CHECKBOX	**-**	Always null (use *selected* instead)
RADIO	**-**	Always null (use *selected* instead)
DATE	**string**	YYYY-MM-DD
TIME	**string**	HH:MM
PHOTO	**string**	filename
BARCODE_TEXT	**string**	-
BARCODE_IMAGE	**object**	`{code: string_value, image: string_value}` • *code*: [string] containing the bar-code numbers • *image*: [filename] path to the bar-code picture
GPS_TEXT	**object**	`{lat: decimal, lng: decimal}` • *lat*: [decimal] latitude • *lng*: [decimal] longitude
GPS_MAP	**object**	`{lat: decimal, lng: decimal}` • *lat*: [decimal] latitude • *lng*: [decimal] longitude
SIGNATURE	**string**	filename
DRAWING	**string**	filename
USERIMAGE	**string**	Filename

TABLE	**object**	`{`
		`row_x_col_y: value,`
		`...`
		`}`
		Example:
		`{`
		`row_0_col_0: 10,`
		`row_0_col_1: 20,`
		`row_1_col_0: 40,`
		`row_1_col_1: 60,`
		`}`

FOODEX	**string**	-

EXAMPLE:

REQUEST

{server}/rest/api/get_operation_data/96/

RESPONSE


```
[
  {
    "id": 128,
    "operation": 129,
    "data": {
      "elements": [
        {
          "id": "text_0",
          "value": "sdfsdsfsdfsdfsdf",
          "enabled": true,
          "crossed": false,
          "visible": true,
          "selected": false
        },
        ...
        {
          "id": "checkbox_1",
          "value": null,
          "enabled": true,
          "crossed": false,
          "visible": true,
          "selected": true
        }
      ],
      "extras": {
        "text_0": {
          "REQUIRED": true
        }
      },
      "annexes": [
        "_abc.csv",
        ...
      ]
    }
  }
]
```

7.6 Operation Data and Form

GET {server}/rest/api/get_operation_form_data/{operation_id}/

Returns the data (inputs and state) of a specific operation and its respective form (only id, name and form), or an empty array if *operation_id* does not exist.

The form is returned just in case, the user required some extra information of some field.

All assets (images and documents) are accessible at `{server}/media/operation_assets/{operation_id}/{asset_file}`.

PATH PARAMETERS:

Field	Type	Description
operation_id	**number**	ID of the operation

HEADER PARAMETERS:

Field	Type	Description
Authorization	-	Token [token]

RESPONSES:

200 Successful
Operation details.

403 Forbidden
Invalid token or Authentication credentials were not provided.

404 Not Founded
Specified operation does not exist.

500 Internal Server Error
Unknown error.

RESPONSE FIELDS DESCRIPTION:

The response is slitted into two objects:

- *form*: description of the form
- *data*: the same object as presented in Operation Data

EXAMPLE:

REQUEST

{server}/rest/api/get_operation_form_data/97/

RESPONSE


```
{
  "form": {
    "id": 2916,
    "name": "dce3c08d-0014-493f-a12a-10dac2c244ba",
    "form": {
      "id": 2916,
      "name": "dce3c08d-0014-493f-a12a-10dac2c244ba",
      "description": "",
      "date_created": "2021-09-17",
      "rasters": [],
      "pages": [
        "page-1"
      ],
      "dropdowns": {},
      "repeatables": [],
      "counter": {
        "TEXT": 1,
        "CHECKBOX": 1
      },
      "groups": [
        {
          "id": "gp-0",
          "name": "Group 0",
          "database": "",
          "required": "no"
        }
      ],
      "elements": [
        {
          "id": "text_0",
          "page": "page-1",
          "section": "section-0",
          "type": "TEXT",
          "props": {
```

```
        "properties-page-number": "1",
        "properties-section": "section-0",
        "properties-id": "text_0",
        "properties-name": "",
        "properties-label": "",
        "properties-show-label": "yes",
        "properties-default": "",
        "properties-enabled": "yes",
        "properties-placeholder": "",
        "properties-max-length": "100",
        "properties-max": "100",
        "properties-min": "0",
        "properties-step": "1",
        "properties-uppercase": "no",
        "properties-required": "no",
        "properties-checked": "no",
        "properties-database": "",
        "properties-group": "none",
        "properties-pattern": "",
        "properties-image-url": "",
        "properties-visible": "yes",
        "properties-zindex": "10",
        "properties-top": "102",
        "properties-left": "204",
        "properties-width": "150",
        "properties-height": "26",
        "properties-font": "Times",
        "properties-font-size": "16",
        "properties-font-style": "normal",
        "properties-font-decoration": "none",
        "properties-font-weight": "normal",
        "properties-horizontal-alignment": "left",
        "properties-color": "#000000",
        "properties-back-color": "#33ffff",
        "properties-back-alpha": "0.2",
        "properties-border-borders": "all",
        "properties-border": "none",
        "properties-border-width": "1",
        "properties-border-radius": "0",
        "properties-rotation": "0"
      }
    },
    {
      "id": "text_1",
      "page": "page-1",
      "section": "section-0",
      "type": "TEXT",
      "props": {
        "properties-page-number": "1",
        "properties-section": "section-0",
        "properties-id": "text_1",
        "properties-name": "",
        "properties-label": "",
        ...
        "properties-border-borders": "all",
        "properties-border": "none",
        "properties-border-width": "1",
```

```
            "properties-border-radius": "0",
            "properties-rotation": "0"
          }
        },
        ...
      ],
      "sections": [
        {
          "id": "section-0",
          "name": "Default"
        }
      ],
      "sections_items": {
        "section-0": [
          "text_0",
          "text_1",
          "gp-0"
        ]
      },
      "sections_groups": {
        "gp-0": {
          "name": "Group 0",
          "section": "section-0"
        }
      },
      "eas": {
        "ea-0": {
          "id": "ea-0",
          "type": "REQUIRED",
          "ea": {
            "events": [
              {
                "logic": null,
                "event": "onFilled",
                "field": "checkbox_0"
              }
            ],
            "actions": {
              "action": "required",
              "target_fields": [
                "text_0"
              ]
            }
          }
        }
      }
    }
  },
  "data": {
    "id": 128,
    "operation": 129,
    "data": {
      "elements": [
        {
          "id": "text_0",
          "value": "sdfsdsfsdfsdfsdf",
          "enabled": true,
```

```
          "crossed": false,
          "visible": true,
          "selected": false
        },
        ...
        {
          "id": "checkbox_1",
          "value": null,
          "enabled": true,
          "crossed": false,
          "visible": true,
          "selected": true
        }
      ],
      "extras": {
        "text_0": {
          "REQUIRED": true
        }
      },
      "annexes": []
    }
  }
}
```


7.7 Get the validation of an operation's inputs

GET {server}/rest/api/validate_operation_inputs/{operation_id}/

Gets the result of applying the validation rules on the inputs of a specific operation.

The returned object is divided in 2 parts: *elements* and *groups*. *Elements* is an array containing all elements (except radios and checkboxes, which are includes in the groups themselves), their identification and the the validation results, which are divided into 2 groups: Errors and Warnings. *Groups* is an array for the inputs representing checkboxes and radios elements.

While Errors should be considered a critical issue, Warnings should not. An *error* occurs when some validation, either by default or defined by the form's creator. A *warning* is just an indication that something is not wrong but also not right. For example, if a field is not mandatory to be filled, and the user doesn't introduze any value, in it, then the system will just give a *warning* and not an *error*.

Possible messages (both for Errors and Warnings):

```
{'code':1, 'message': 'No value!'}
{'code':2, 'message': 'Incorrect format/pattern!'} # PATTERN
{'code':3, 'message': 'No uppercase!'}
{'code':4, 'message': 'Max length exceed!'}
{'code':5, 'message': 'Max value exceeded!'}
{'code':6, 'message': 'Min value exceeded!'}
{'code':7, 'message': 'No option selected!'}
```

Information returned:

For *elements*:

- Element ID
- Element Name
- Element Label
- Array with the Warnings messages
- Array with the Errors messages

For *groups*:

- Group ID
- Group Name
- Array with the Warnings messages
- Array with the Errors messages

Disabled or *hidden* elements or groups return null as their Warnings and Errors messages. Check element *text_1* and group *gp_1* in the example bellow.

PATH PARAMETERS:

Field	Type	Description
operation_id	**number**	ID of the operation

HEADER PARAMETERS:

Field	Type	Description
Authorization	**-**	Token [token]

RESPONSES:

200 Successful
Operation details.

403 Forbidden
Invalid token or Authentication credentials were not provided.

404 Not Founded
Specified operation does not exist.

500 Internal Server Error
Unknown error.

RESPONSE FIELDS DESCRIPTION:

The response is slitted into two objects:

- *elements*: array of elements (except CHECKBOXES and RADIOS), with their main properties (*id*, *name* and *label*) and two arrays of objects, *warnings* and *errors*, containing the results of the validation process
- *groups*: array of groups (of CHECKBOXES and RADIOS), with their *id* and *name* and two arrays of objects, *warnings* and *errors*, containing the results of the validation process

EXAMPLE:

REQUEST

{server}/rest/api/validate_operation_inputs/97

RESPONSE


```
{
  "elements": [
    {
      "id": "text_0",
      "name": "",
      "label": "",
      "errors": [
```

```
        {
          "code": 2,
          "message": "Incorrect format/pattern!"
        }
      ],
      "warnings": [
        {
          "code": 1,
          "message": "No value!"
        },
      ]
    },
    {
      "id": "drawing_0",
      "name": "",
      "label": "",
      "errors": [],
      "warnings": [
        {
          "code": 1,
          "message": "No value!"
        },
      ]
    },
    {
      "id": "text_1",
      "name": "",
      "label": "",
      "errors": null,
      "warnings": null
    },
    ...
  ]
"groups": [
    {
      "id": "gp-0",
      "name": "Group 0",
      "errors": [],
      "warnings": [
        {
          "code": 1,
          "message": "No value!"
        }
      ]
    },
    {
      "id": "gp-1",
      "name": "Group 1",
      "errors": null,
      "warnings": null
    }
  ]
}
```

7.8 Get Operation Data (inputs/state) + Validations

GET {server}/rest/api/get_operation_data_val/{operation_id}/

Returns an array of *elements* and *groups* with the following information:

For *elements*:

- Element ID
- Input Value
- Element Name
- Element Label
- Database Field
- Is Element enabled?
- Is Element visible?
- Object with all validations to apply to the element

For *groups*:

- Group ID
- Group Name
- Database Field
- Is Group enabled?
- Is Group visible?
- Is required to be selected?
- Group type: CHECKBOX or RADIO
- Array with the {ID, name, label} of the selected elements (radios and checkboxes)

All assets (images and documents) are accessible at **`{server}/media/operation_assets/{operation_id}/{asset_file}`**.

PATH PARAMETERS:

Field	Type	Description
operation_id	**number**	ID of the operation

HEADER PARAMETERS:

Field	Type	Description
Authorization	-	Token [token]

RESPONSES:

200 Successful
Operation details.

403 Forbidden
Invalid token or Authentication credentials were not provided.

404 Not Founded
Specified operation does not exist or No data to validate (no operation data).

500 Internal Server Error
Unknown error.

RESPONSE FIELDS DESCRIPTION:

The response is split into two objects:

- *elements*: array of elements (except CHECKBOXES and RADIOS), with:
  - main properties: *id* [string], *name* [string], *label* [string] and the *database field* [string]
  - status: *enabled* [boolean] and *visible* [boolean]
  - value: user's input [(*)]
  - validations: an object with all the validation to apply to this field, which may include:
    - max-length [string]
    - max [string]
    - min [string]
    - uppercase [boolean]

        - required [boolean]
        - pattern [string]
- *groups*: array of groups (of CHECKBOXES and RADIOS), with:
    - main properties: *id* [string], *name* [string], and the *database field* [string]
    - whether or not it's required to have at least one element selected: *required* [boolean]
    - status of the group: *enabled* [boolean] and *visible* [boolean]
    - A list of objects indicating the selected elements. Each of these objects should contain: *id* [string], *name* [string], and the *label* [string]
    -

`value (each element contains this field, which depends on the element's type):`

Element Type	Value type	Description
TEXT	**string**	-
EMAIL	**string**	-
PHONE	**string**	-
WEBSITE	**string**	-
TEXTBOX	**string**	-
NUMBER	**string**	-
DROPDOWN	**string**	-
CHECKBOX	**-**	Always null (use *selected* instead)
RADIO	**-**	Always null (use *selected* instead)
DATE	**string**	YYYY-MM-DD
TIME	**string**	HH:MM
PHOTO	**string**	filename
BARCODE_TEXT	**string**	-
BARCODE_IMAGE	**object**	`{code: string_value, image: string_value}` • *code*: [string] containing the bar-code numbers • *image*: [filename] path to the bar-code picture
GPS_TEXT	**object**	`{lat: decimal, lng: decimal}` • *lat*: [decimal] latitude • *lng*: [decimal] longitude
GPS_MAP	**object**	`{lat: decimal, lng: decimal}`

		• *lat*: [decimal] latitude • *lng*: [decimal] longitude
SIGNATURE	**string**	filename
DRAWING	**string**	filename
USERIMAGE	**string**	filename

TABLE	**object**	{
		row_x_col_y: value,
		...
		}
		Example:
		{
		row_0_col_0: 10,
		row_0_col_1: 20,
		row_1_col_0: 40,
		row_1_col_1: 60,
		}

FOODEX	**string**	-

EXAMPLE:

REQUEST

{server}/rest/api/get_operation_data_val/99

RESPONSE


```
{
  "elements": [
    {
      "id": "text_0",
      "value": "sdfsdsfsdfsdfsdf",
      "name": "",
      "label": "",
      "database": "",
      "type": "TEXT",
      "enabled": true,
      "visible": true,
      "validations": {
        "max-length": "100",
        "uppercase": false,
        "required": true,
        "pattern": ""
      }
    },
```

```
    {
      "id": "text_1",
      "value": "",
      "name": "",
      "label": "",
      "database": "",
      "type": "TEXT",
      "enabled": true,
      "visible": true,
      "validations": {
        "max-length": "100",
        "uppercase": false,
        "required": false,
        "pattern": ""
      }
    }
  ],
  "groups": [
    {
      "id": "gp-0",
      "name": "Group 0",
      "database": "",
      "required": true,
      "selected_elements": [
        {
          "name": "",
          "label": "",
          "id": "checkbox_0"
        },
        {
          "name": "",
          "label": "",
          "id": "checkbox_1"
        }
      ],
      "type": "CHECKBOX",
      "enabled": true,
      "visible": true
    }
  ],
  "annexes": []
}
```


7.9 Delete an Operation

POST {server}/api/delete_operation/

Deletes a specified COMPLETED operation, including all its assets, if any.

TO DELETE A COMPLETED OPERATION, AN ADMINISTRATOR MUST DELETED IT DIRECTLY IN THE DATABASE OR THROUGH THE ADMINISTRATIVE PANEL.

HEADER PARAMETERS:

Field	Type	Description
Authorization	-	Token [token]

REQUEST BODY SCHEMA:

Field	Type	Description
operation_id	**number**	ID of the operation

RESPONSES:

200 Successful
Successful message.

```
{
  "message": "Operation removed"
}
```

400 Bad Request
Body parameter missing or incorrect.

403 Forbidden
Invalid token or Authentication credentials were not provided. Attempt to change the status of a COMPLETED operation.

404 Not Founded
Specified operation does not exist.

500 Internal Server Error
Unknown error.

7.10 Set the Status of an Operation

POST {server}/api/set_operation_status/

Set the status of a specified non COMPLETED operation. If successful, returns the operation details.

States:

- **OPEN**
- **CLOSED**
- **COMPLETED**

ONCE AN OPERATION IS SET TO **COMPLETED**, IT'S ONLY POSSIBLE TO CONSULT IT. WILL NO LONGER POSSIBLE TO EDIT ANY PART OF THE OPERATION!

TO CHANGE THE STATUS OF A **COMPLETED** OPERATION, AN ADMINISTRATOR MUST CHANGE IT DIRECTLY IN THE DATABASE OR THROUGH THE ADMINISTRATIVE PANEL.

HEADER PARAMETERS:

Field	*Type*	*Description*
Authorization	**-**	Token [token]

REQUEST BODY SCHEMA:

Field	*Type*	*Description*
operation_id	**number**	ID of the operation
status	**string**	Status (OPEN | CLOSED | COMPLETED)

RESPONSES:

200 Successful
Successful message.


```
{
  "id": 100,
  "name": "group3",
  "description": "",
  "form": 2839,
  "form_name": "7bd9602c-d33a-4497-a23c-6dee57bf0bed",
  "date_creation": "2021-09-07T08:39:43.685505Z",
  "date_updated": "2021-09-07T08:53:59.884398Z",
  "author": 1,
  "author_name": "admin",
  "updated_by": 1,
  "updated_by_name": "admin",
  "status_name": "CLOSED"
}
```


400 Bad Request
Body parameter missing or incorrect.

403 Forbidden
Invalid token or Authentication credentials were not provided.
Attempt to change the status of a COMPLETED operation.

404 Not Founded
Specified operation does not exist.

500 Internal Server Error
Unknown error.

7.11 List all Assets of an Operation

GET {server}/rest/api/list_operation_assets/{operation_id}/

Returns a list of all assets associated with the specific operation, or an empty array if *operation_id* does not exist.

HEADER PARAMETERS:

Field	Type	Description
Authorization	**-**	Token [token]

REQUEST BODY SCHEMA:

Field	Type	Description
operation_id	**number**	ID of the operation

RESPONSES:

200 Successful
List of all assets.

403 Forbidden
Invalid token or Authentication credentials were not provided. Attempt to change the status of a COMPLETED operation.

RESPONSE FIELDS DESCRIPTION:

Element Type	Value type	Description
id	**integer**	Asset ID
name	**string**	Asset name
type	**integer**	Asset type
type_name	**string**	Asset type name
is_annex	**boolean**	Is annex?
date	**string | datetime**	When was the asset uploaded
operation	**integer**	Operation's ID

Assets types:

Type	Name
1	IMAGE
2	CSV
3	JSON
4	PDF
5	OTHER

EXAMPLE:

REQUEST

{server}/rest/api/get_operation_data/341/

RESPONSE

```
[
  {
    "id": 730,
    "name": "DOC_341_SIGNED.pdf",
    "asset": "http://127.0.0.1:8000/media/operation_assets/341/
DOC_341_SIGNED.pdf",
    "type": 4,
    "type_name": "PDF",
    "is_annex": true,
    "date": "2022-01-17T14:59:59.856123Z",
    "operation": 341
  },
  {
    "id": 730,
    "name": "brush.png",
    "asset": "http://127.0.0.1:8000/media/operation_assets/341/
brush.png",
    "type": 1,
    "type_name": "IMAGE",
    "is_annex": false,
    "date": "2022-01-18T20:12:33.949468Z",
    "operation": 341
  },
  ...
]
```

7.12 Examples

7.12.1 Example #2: Token generation

```
const options = {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json;charset=utf-8',
    },
    body: JSON.stringify({
        "username": "username",
        "password": "userpassword"
    }),
}
fetch('http://34.134.170.7/rest/api/api-token-auth/', options)
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.log(error))
```

7.12.2 Example #1: List all operations

```
const options = {
    method: 'GET',
    headers: {
        'Content-Type': 'application/json;charset=utf-8',
        'Authorization': 'Token token_here',
    },
}
fetch('http://34.134.170.7/rest/api/list_operations/', options)
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.log(error))
```

8 Administrative Back Office

8.1 Access

The access the Administrator Back-office requires administrative privileges. The back-office can be accessed directly by:

{server}/admin/

or by clicking on the *Admin* link on the top menu of any page (except in the Designer application).

In case the Admin is not logged yet, he will have input his credentials (fig. 184).

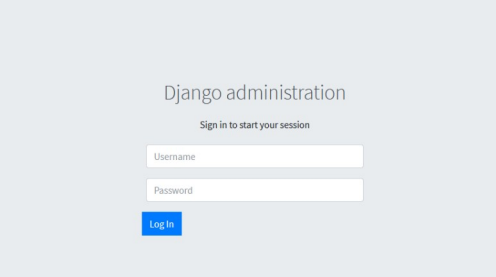

Fig. 184: Back-office login page

After the validation, the user will be taken to the back-office page, as depicted in fig. 185.

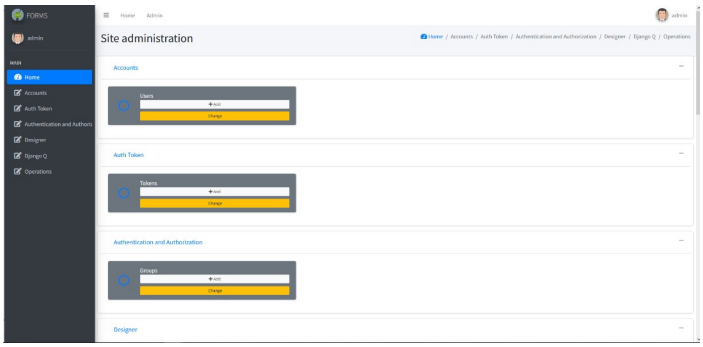

Fig. 185: Back-office main page

8.2 Tables

There are 15 tables available to an administrator. These are described in table 19.

Application	Table	Description
Accounts	*User*	All users and privileges
Auth Token	*Tokens*	All tokens
Authentication and Authorization	*Groups*	All groups that are used to define access privileges: admin / manager / ...
Files	*Files*	Configuration files, data files, application and programs.
Designer	*Asset types*	All assets type: PDF / IMAGE / ...
	Form assets	Assets description, their location, dates and associated forms
	Forms	Forms details and data
	Queries	Queries used in the database query E/As
	Status	Lists all possible states a form, operation and queries can be: EDITABLE / CLOSED / ...
Django Q	*Failed tasks*	These tables are related with the automatic tasks django performs every n interval: delete all temps forms / ...
	Queued tasks	
	Scheduled tasks	
	Successful tasks	
Operations	*Operation assets*	Assets description, their location, dates and associated operation
	Operations	Operations details
	Operations Data	Operations inputs

Tab. 19: Back-office tables

8.3 “Allowed” Writing Operations

Despite an administrator can change the content of any of the tables, it’s recommended that he should use its privileges and access only to perform essential tasks like the ones presented here. Any other change, other than these one, can have disastrous consequences, from malfunction of certain applications to total fail.

Naturally, consulting the contents of any table won’t have any negative effect.

8.3.1 Users and Privileges

To create a new user and its privilege we select *Accounts* > *Users*. There we select *+Add User* (fig. 186).

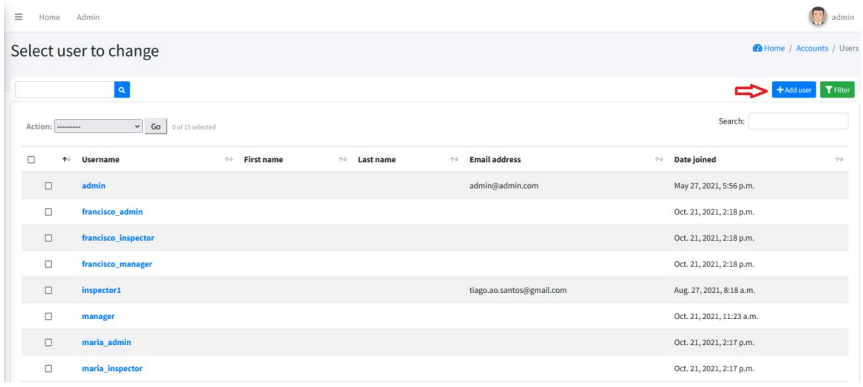

Fig. 186: Add user

After filling up the username and the password, the administrator can either Save and add another user or Save and continue in order to define this user's personal data and privileges (fig. 187).

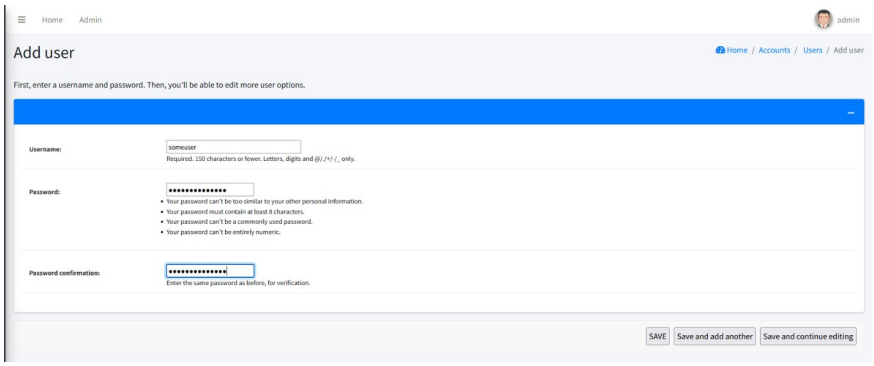

Fig. 187: Adding the new user's credentials

Selecting continue editing, we can input the names, email, status and also define the privileges. Privileges are set in the permissions section and defined by groups (fig. 188).

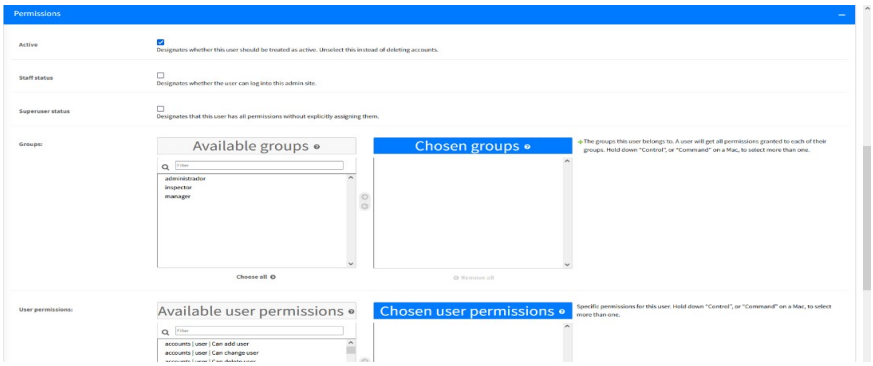

Fig. 188: Permissions

We have setup three groups:

- Administrator
- Manager
- Inspector

To set their access, we just need to follow table 20.

Level of access	***Groups to be included in***
Administrator	Administrator Manager Inspector
Manager	Manager Inspector
inspector	Inspector

Tab. 20: Groups vs privileges

So, for example, a manager will have to be included in the Manager's groups and in the Inspector's group (fig. 189). After all, a manager might also have a role as inspector.

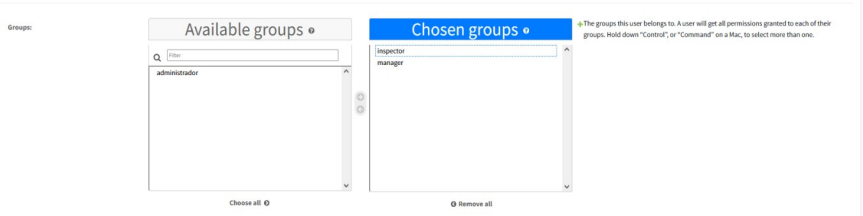

Fig. 189: Groups any manager must belong to

8.3.2 Form Status

Sometime a manager might accidentality change the status of a form to **DISABLED**. If this happens, the manager will either have to clone the form to make it **EDITABLE** or **IN USE**, or he can contact the administrator and ask him to change the state.

The administrator only needs to go to Designer > Forms, select the form, change the status field and save.

8.3.3 Operation Status

As in the previous section a manager might also accidentality change the status of an operation to **COMPLETE**. However, in this case, there is no other choice other than asking an administrator to change the state.

The administrator only needs to go to Operations > Operations, select the operation, change the status field and save.

8.3.4 Data and Application Files

There are certain files that are required for some pages. These are listed in table 21. Despite being possible to update these files directly in the server, it's recommended to updated them through the administrator interface, otherwise, conflicts with the dates may occur (fig. 190).

File	Description	Required fields	Data source
foodex_attributes_EN.csv	Latest Foodex2 attributes	*code* *label*	Export from the EFSA Cata-logueBrowser (Hierarchy: MTX)
foodex_terms_EN.csv	Latest Foodex2 terms	*termCode* *masterParentCode* *termScopeNote* *termExtendedName* *allFacets* *deprecated* *termExtendedName_pt* *termScopeNote_pt*	Export from the EFSA Cata-logueBrowser (Hierarchy: MTX)
CCNativeApp.exe	Application for windows 64bits, for signing a PDF document.	-	-
chrome_extension.zip	Chrome exten-sion, for signing a PDF docu-ment.	-	-
users_manual.pdf	Users' manual	-	-
forms_dev_notes.pdf	Development notes.	-	-

Tab. 21: Configuration and data files

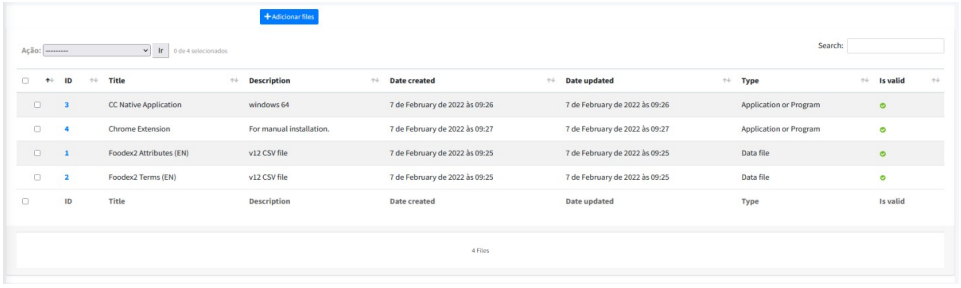

Fig. 190: Some files

The *is_valid* field, indicates that the uploaded file is not valid, most likely caused by incorrect fields.

9 Operational Diagrams

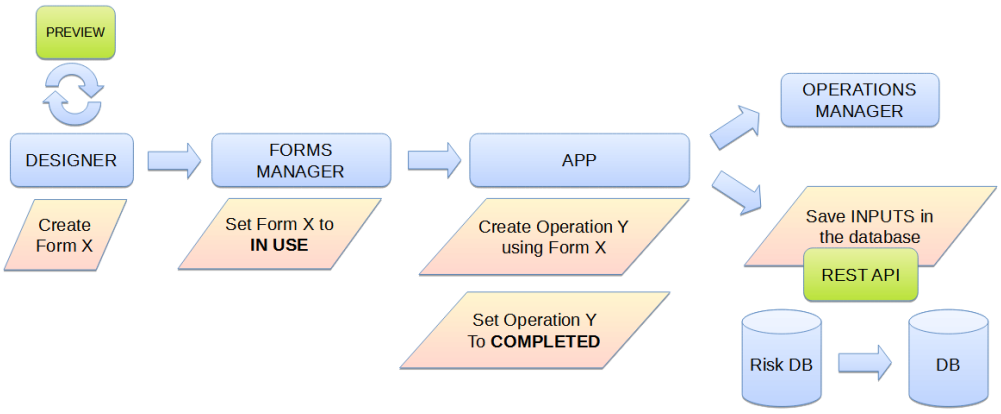

Fig. 191: From Designer to Database

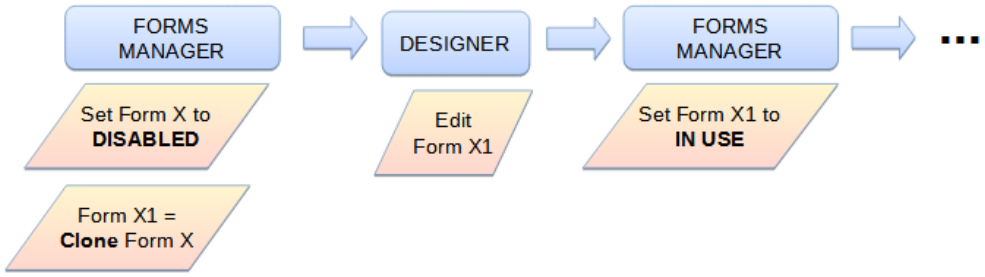

Fig. 192: Altering an IN USE Form

10TODO

We now present the list of some features yet to be implemented:

- **Designer**:
  - Richtext element
  - Validations for the FoodEx2 element
  - Import / Export - Import and export forms to doc format
  - Undo
  - Rest API E/A - An E/A that make an API call and to fills fields
  - SQL parser to avoid having to use markers in E/A Database query
  - Chain events
  - Graphical interface for the E/A system
  - Selection of multiple background images in one selection
  - PDF selection as background image, also capable to automatically add additional pages if required
- **App**:
  - Smart cart signature in mobile devices
  - Starting actions so that an action is executed if its event are already fulfilled at start
- **Global**:
  - Form's Table assets - Keep synchronized with an external source
  - Teams management system