

# Widgets

## Objective

---

The main aim of this documentation is to inform the back-end team of the expected configuration types and objects for the front-end to build the widgets dynamically.

## Overview of the Widgets design architecture

---

The idea behind the widgets design architecture is to have widget types and each type would have its corresponding configs object that determines how that widget should be displayed.

The following enumeration represents all the available widget types so far and it should be extended as needed as we go.

```
1 export enum WidgetTypes {
2   EMPTY = 'empty',
3   LAYOUT = 'layout',
4   TABS = 'tabs',
5   FORM = 'form',
6   LIST = 'list',
7   STATISTICS = 'statistics',
8   CATEGORIES = 'categories',
9   DOCUMENTS = 'documents',
10  ORDER = 'order',
11  ADDRESSES = 'addresses',
12  LOG_NOTE = 'log-note',
13  BRANCHES = 'branches',
14  INSURANCE = 'insurance',
15  FREE_FORM = 'free-form'
16 }
```

Any widget that is going to be displayed should have the following type

```
1 export type WidgetType =
2   | EmptyWidgetType
3   | LayoutWidgetType
4   | TabsWidgetType
5   | FormWidgetType
6   | ListWidgetType
7   | StatisticsWidgetType
8   | CategoriesWidgetType
9   | DocumentsWidgetType
10  | OrderWidgetType
11  | AddressesWidgetType
12  | LogNoteWidgetType
13  | BranchesWidgetType
14  | InsuranceWidgetType
15  | FreeFormWidgetType;
```

Each Widget type in the union above is going to be discussed in details in the following section. Each one will have a type and a configs object if needed.

## Attribute Tags

Tag	Meaning
OPTIONAL	Means that the attribute may or may not be provided.
MENDATORY	Means that the attribute must be provided.
LOCALIZABLE	Means you could provide this attribute with or without localization. Any attribute with this tag could be provided as a string, or a localization object. <pre>1 type LocalizationObject = { 2   en: string; 3   ar: string; 4 }</pre>

## Supported widgets

### ☐ Empty Widget

```
1 export type EmptyWidgetType = {  
2   type: WidgetTypes.EMPTY;  
3 };
```

This is the most simple type of widget were nothing is displayed in its place. It could be useful inside layouts or any other types of widgets that take other widgets as children to leave a certain space empty. It has no configs object.

### ☐ Layout Widget

```
1 export type LayoutWidgetType = {  
2   type: WidgetTypes.LAYOUT;  
3   configs: {  
4     backgroundColor?: string; //defaults to the inherited background color.  
5     textColor?: string; //defaults to the inherited text color  
6     views: {  
7       backgroundColor?: string; //defaults to the inherited background color.  
8       textColor?: string; //defaults to the inherited text color  
9       occupies: Occupation;  
10      widget: WidgetType;  
11    }[];  
12  };  
13 };
```

This is the widget used to determine how the other widgets should be laid out on the screen. It should have an array of other widgets where for each one you specify how much it occupies from the screen, the background color and the text color.

### Configs attributes

Attribute	Tags	Description
-----------	------	-------------

backgroundColor	OPTIONAL	The background color for the whole layout.
textColor	OPTIONAL	The text color for the whole layout.
views	MENDATORY	This should be an array of other views each view could also have a background and text color, in addition to how much it should occupy from the screen and the widget itself. The occupies and widget attributes are going to be discussed below.

#### View attributes

Attribute	Tags	Description
occupies	OPTIONAL	This determines how much the widget occupies from the screen according to the layout system explained below.
widget	MENDATORY	This could be any other widget, including other layout widgets as well. Which means you could have as many nested layouts as needed.

#### Layout System

The layout system is based on a 12-column grid representing the entire screen. The following type represents how many columns should the widget occupy from a screen of 12 cols (6 means 50% of the screen width, 12 means 100% of the screen width...etc.)

```
1 export type OccupationRange = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12;
```

But the `occupies` attribute itself has the following type

```
1 export type Occupation =
2   | OccupationRange
3   | { smallScreen: OccupationRange; largeScreen: OccupationRange }
4   | undefined;
```

- If provided a number this will be considered for the large screen size and the small screen size will occupy the full width by default (12 cols).
- If provided an object then you specify the number for the small screen size and the number for the large screen size.
- If undefined then it occupies the full width by default (12 cols) for both the large and small screen size.

#### Tabs Widget



```
1 export type TabsWidgetType = {
2   type: WidgetTypes.TABS;
3   configs: {
4     design:
5       | 'material' //The underlined tabs with the primary color
6       | 'bootstrap'; //The bubbled tabs with the grey color
7     tabs: { name: string | LocalizationObject; widget: WidgetType }[];
8   };
9 };
```

This widget is used to display tabs in the screen each tab should have a name and its corresponding widget.

## Configs attributes

Attribute	Tags	Description
design	OPTIONAL defaults to bootstrap	This determines the look of the tabs. It could be either bootstrap or material design. The difference between the two is illustrated below.
tabs	MENDATORY	This should be an array of tabs where each tab has a name and a widget. The names are displayed on the tabs header and each widget is displayed below the tabs header according to its corresponding name when clicked. The widget attribute could be any other widget. Including other tabs, or layouts...etc.

## Design attribute

Design	Look
material	
bootstrap	

## Tab attribute

Attribute	Tags	Description
name	MENDATORY LOCALIZABLE	Displayed on the tabs header.
widget	MENDATORY	Any other widget from the supported widgets.

## ☐ Statistics Widget

```
1 export type StatisticsWidgetType = {
2   type: WidgetTypes.STATISTICS;
3   configs: {
4     cards: {
5       backgroundColor?: string; //defaults to the inherited background color.
6       textColor?: string; //defaults to the inherited text color
7       svgIcon?: string; //defaults to no icon
8       name: string | LocalizationObject;
9       fieldKeys: string[]; //each string would be a key to some value in the returned data,
10      //if it is a nested value then keys are separated by the dot annotation
11      //(ex: "numberOfPatients" or "patients.count")

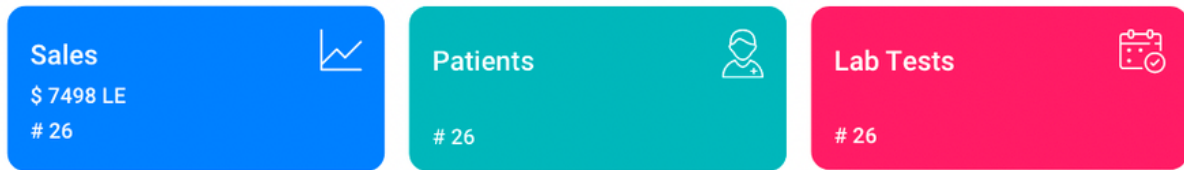
```

```

12     }[];
13   };
14 };

```

This is used to display any statistics that you specify for the current module. For each type of statistics you specify the field keys to use from the returned data of an agreed upon request. The end result should look something like the screenshot below:



If there is no enough space horizontally the cards will wrap to the next line.

### Configs attributes

Attribute	Tags	Description
<code>cards</code>	MENDATORY	An array of cards to display for the statistics. For each card you specify a background color, text color, name, icon, and the field keys used to access the data. Each attribute of the card is discussed in details below.

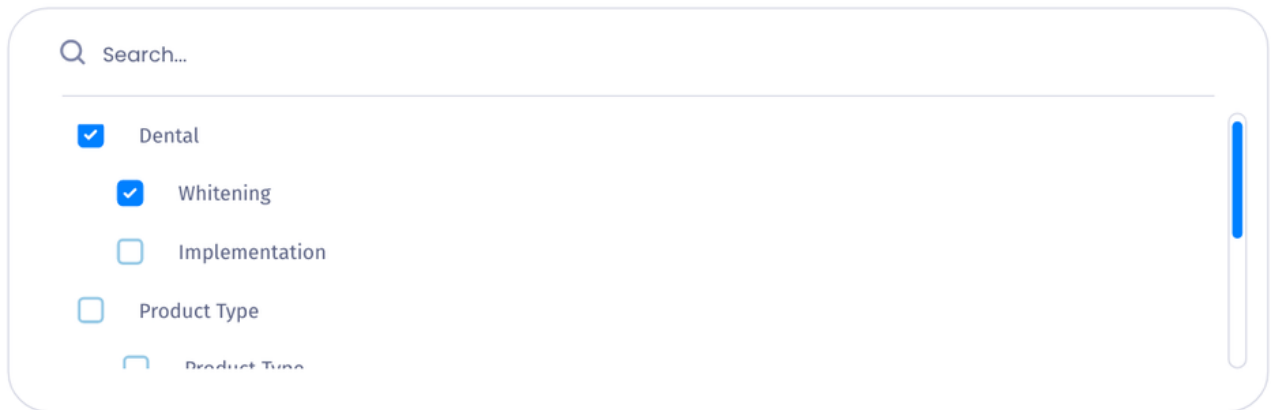
### Card attributes

Attribute	Tags	Description
<code>backgroundColor</code>	OPTIONAL	The color of the card. Defaults to the inherited background color.
<code>textColor</code>	OPTIONAL	The color of the text on the card including the icon color as well. Defaults to the inherited text color.
<code>svgIcon`</code>	OPTIONAL	The icon to be displayed on the right most of the card. If no icon is provided nothing is displayed.
<code>name</code>	MENDATORY LOCALIZABLE	The title displayed on the card.
<code>fieldKeys</code>	MENDATORY	It is an array of strings. Each string would be a key to some value in the returned data of a request that we would agree upon (this request would be dynamic based on the current module). If it is a nested value then keys are separated by the dot annotation. (ex: <code>numberOfPatients</code> or <code>patients.count</code> )

## ☐ Categories Widget

```
1 export type CategoriesWidgetType = {
2   type: WidgetTypes.CATEGORIES;
3   configs: {
4     categories: CategoriesType[];
5   };
6 };
7
8 export type CategoriesType = {
9   id: string;
10  name: string | LocalizationObject;
11  children?: CategoriesType[];
12 };
```

This widget is used to display nested categories in a recursive manner with the ability to search in and select from those categories and sub categories.



### Configs attributes

Attribute	Tags	Description
categories	MENDATORY	An array of categories, where each category would have an id, name, and an optional nested sub categories array with the same structure.

### Category attributes

Attribute	Tags	Description
id	MENDATORY	A unique identifier for the category. This will be used in the mutation request to send an array of the selected categories Ids.
name	MENDATORY LOCALIZABLE	The display name of the category.
children	OPTIONAL	An optional array of sub categories. There could be as many nested categories as needed. To stop the nesting don't provide this attribute.

## Documents Widget

```
1 export type DocumentsWidgetType = {
2   type: WidgetTypes.DOCUMENTS;
3   configs: {
4     type?: DocumentTypes;
5   };
6 };
7
8 export enum DocumentTypes {
9   PDF = 'pdf',
10  IMAGE = 'image',
11  VIDEO = 'video'
12 }
```

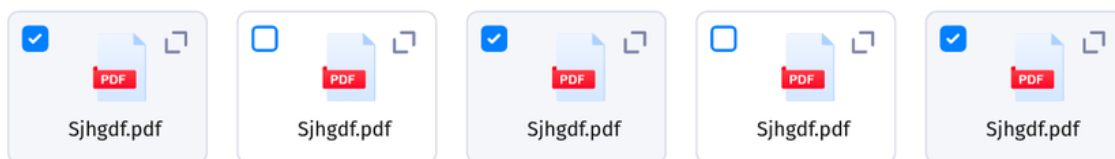
This widget displays the documents of a specific module. It supports three types of documents:

- Images
- Videos
- PDFs

The data of the documents should be obtained dynamically based on the module and the type of the documents we want to fetch from an agreed upon request. We should agree on the mutation requests as well to be dynamic across all the modules.

It supports the following features:

- Selecting Documents
- Deleting the selected documents
- Adding new documents from the documents manager integrated with AWS.
- Previewing documents



### Configs attributes

Attribute	Tags	Description
type	<b>OPTIONAL</b> <b>DEFAULTS TO</b> image	The type of the document. It could be "image", "video", or "pdf" <pre>1 export enum DocumentTypes { 2   PDF = 'pdf', 3   IMAGE = 'image', 4   VIDEO = 'video' 5 }</pre>

## ❑ Order/Invoice Widget

```
1 export type OrderWidgetType = {
2   type: WidgetTypes.ORDER;
3 };
```

The widget is used to display the order details, customer details, and invoice breakdown. It currently has no `configs` object, but we can add one later on to make it more customizable. The widget should get its data from an agreed upon back-end request that is dependent on the current order id. Hence, this widget should only be used in a page where an order id is accessible.

The order could also be mutated through:

- Updating the customer name
- Updating the customer address (Through a dropdown of all addresses)
- Adding a new invoice item line
- Editing an invoice item line
  - Product (Through a dropdown of all products)
  - Quantity
  - Tax
  - Discount
- Deleting an invoice item line

### Sales Order SO000334

Customer: [Hossam Hemaya](#)

[Edit](#)

Order Date: 05 Apr, 2023, 1:53:55 pm

Address: Nasr City, Al Waha street 34

[Change](#)

Invoice: INV37546

Building: 445, Floor: 3rd floor







Invoice Date: 05 Apr, 2023, 1:53:55 pm

City: Cairo

Currency: EGP

Country: Egypt

### Order Lines

Product	Description	Unit Price	Quantity	Tax	Discount	Total	Action
Test	Test	300.00	2	10.00%	0.00%	660.00	 
Test	Test	300.00	2	10.00%	0.00%	660.00	 
Test pro...	▼ Test product	100.00	<input type="text" value="1"/>	<input type="text" value="5"/> %	<input type="text" value="0"/> %	105.00	 
+ Add Line							

Untaxed Amount: 1300.00 EGP

Taxes: 125.00 EGP

 Print

**Total:** 1425.00 EGP

## ❑ Form Widget

```
1 export type FormWidgetType = {
2   type: WidgetTypes.FORM;
3   configs: {
4     fields: FormField[];
5     button?: {
```



```

6     text?: string | LocalizationObject; //defaults to "save"
7     icon?: string; // svg icon (defaults to a check mark)
8     align?: 'center' | 'start' | 'end' | 'full'; //defaults to end
9 };
10 };
11 };

```

```

1  export type FormField = PrimitiveFormField | ModuleFormField;
2
3  export type PrimitiveFormField = BaseFormField & {
4    type: FormPrimitiveTypes;
5    isModule?: false;
6  };
7
8  export type ModuleFormField = NestedModuleFormField | SelectModuleFormField;
9
10 export type NestedModuleFormField = BaseModuleFormField & {
11   isNested: true;
12   fields: FormField[];
13 };
14
15 export type SelectModuleFormField = BaseModuleFormField & {
16   isNested?: false;
17   selectFields: { label: string; value: string };
18   modulePlural: string;
19 };
20
21 export type BaseModuleFormField = BaseFormField & {
22   type: string; //any module name
23   isModule: true;
24 };
25
26 export type BaseFormField = {
27   name: string;
28   priority: number;
29   isArray?: boolean;
30   validation?: Partial<ValidationObject>;
31   conditionalFieldName?: string; //The field name that must be defined in order for this field to be visible (if
32   styles?: {
33     occupies?: Occupation;
34   };
35 };

```

This widget is used to display the form of a module based on the fields provided.

#### Configs attributes


Attribute	Tags	Description
fields	MENDATORY	An array of fields that should be in the form. Each field should have a <code>FormField</code> type and each attribute in the <code>FormField</code> is explained below in the "Field attributes" section.

button	OPTIONAL	Used to provide some configs for the submit button. Each attribute this object takes is explained in details in the “Button attributes” section.
Field attributes		
Attribute	Tags	Description
name	MENDATORY	The name of the form field that is going to be sent in the mutation request.
priority	MENDATORY	<p>The priority of this field. Fields are sorted based on this priority. Lower numbers indicate higher priorities. Note also that the given priority is relative to the level this field is in, so in case of nested fields, priorities are sorted based on each nesting level.</p> <p>We could remove this attribute and depend only on the order of the array directly without additional sorting from the client side.</p>
type	MENDATORY	<p>The type of the field. It could be a <code>FormPrimitiveTypes</code> from the following:</p> <pre> 1 export enum FormPrimitiveTypes { 2   STRING = 'string', 3   NUMBER = 'number', 4   BOOLEAN = 'boolean', 5   DATE = 'date', 6   TIME = 'time', 7   FILE = 'file' 8 }</pre> <p>and in this case, <code>isModule</code> could be set to false or left undefined.</p> <p>Or it could be a <code>string</code> of the name of the module. In this case <code>isModule</code> must be set to true.</p> <p>Notice that the given type should not contain <code>[]</code> to indicate an array type as we already have a flag <code>isArray</code> for that and they should only be a <code>FormPrimitiveTypes</code> or a <code>string</code> of the name of a module.</p>
isArray	OPTIONAL DEFAULTS TO false	Whether this field is an array or not.
isModule	OPTIONAL DEFAULTS TO false	<p>Whether the given type is a module or not. If <code>isModule</code> is set to true, then additional attributes must be set:</p> <ul style="list-style-type: none"> <li>In case <code>isNested</code> is false or undefined <ul style="list-style-type: none"> <li><code>selectFields</code></li> <li><code>modulePlural</code></li> </ul> </li> <li>In case <code>isNested</code> is true <ul style="list-style-type: none"> <li><code>fields</code></li> </ul> </li> </ul>

isNested	OPTIONAL DEFAULTS TO false	<p>Whether the module is nested or not.</p> <ul style="list-style-type: none"> <li>If the module is not nested then a dropdown is displayed to choose from multiple options of the module. And we provide a <code>selectFields</code> attribute to determine the label and value for the dropdown. and a <code>modulePlural</code> to determine the plural name of the module we are going to fetch the dropdown options from.</li> <li>If the module is nested then we provide an additional <code>fields</code> attribute and display the fields inside this array in the form.</li> </ul>
selectFields	OPTIONAL but MENDATORY if isModule=true and isNested=false undefined	<p>An object that has the following type</p> <pre>1 { label: string; value: string }</pre> <p>The <code>label</code> attribute indicates the name of the field that is going to be used as a label for the dropdown.</p> <p>The <code>value</code> attribute indicates the name of the field that is going to be used as a unique value for the dropdown. And this is the value that is going to be submitted in the mutation request to represent the user's choice. In most cases this should be the <code>id</code> field.</p>
modulePlural	OPTIONAL but MENDATORY if isModule=true and isNested=false undefined	<p>The plural name of the module. It is used to fetch all the options used to be displayed in the dropdown.</p>
fields	OPTIONAL but MENDATORY if isModule=true and isNested=true	<p>An array of nested fields inside another field. Each item in that array has the same type <code>FormField</code> of its parent field normally to allow any number of nesting levels.</p> <p>For example, if a parent field has <code>name="address"</code> and this field has an array of child <code>fields</code> and that array has fields with names <code>name="city"</code> and <code>name="country"</code>, then the object submitted could be:</p> <pre>1 { 2   address:{ 3     city:"Any City", 4     country: "Any Country" 5   } 6 }</pre>
validation	OPTIONAL	<p>You could provide any of the following attributes to add validation to the field.</p> <pre>1 export type ValidationObject = { 2   required: boolean;</pre>

		<pre> 3   pattern: string; //Regex 4   //For numbers 5   min: number; 6   max: number; 7   //For strings 8   minLength: number; 9   maxLength: number; 10  //For files 11  maxNumberOfFiles: number; 12  maxFileSizeInBytes: number; 13  }; </pre>
styles	OPTIONAL	Currently, it has an <code>occupies</code> attribute that determines the width this field occupies just like how it is explained previously.
conditionalFieldName	OPTIONAL	If provided, the field will only be shown under the condition that the field with this name is filled out.

### Button attributes

Attribute	Tags	Description
text	OPTIONAL LOCALIZABLE DEFAULTS TO "Save"	The button text. It could be a <code>string</code> or a <code>LocalizationObject</code>
icon	OPTIONAL DEFAULTS TO 	The button icon as a string <code>svg</code> element.
align	OPTIONAL DEFAULTS TO end	Determines how the button should be aligned. It could take four values: <ul style="list-style-type: none"> <li><code>full</code> occupies the full width.</li> <li><code>start</code> aligned at the start with a fixed width.</li> <li><code>center</code> aligned at the center with a fixed width.</li> <li><code>end</code> aligned at the end with a fixed width.</li> </ul>

### ☐ List Widget

```

1 export type ListWidgetType = {
2   type: WidgetTypes.LIST;
3   configs: {
4     initialFieldsLimit: number;
5     supportedViews: SupportedView[];
6     access: Access[];
7     fields: ListField[];
8     defaultView?: SupportedView; //defaults to SupportedView.LIST
9     defaultSearch?: string; //defaults to an empty string
10    hideControllerBar?: boolean; //defaults to false
11    customControllers?: ListControllers[];
12    modulePlural?: string; //if not provided then it is obtained from the url
13    kanbanOptions?: KanbanOptionsType; //used if the supported views include the kanban view
14  };

```

```

15 };
16
17 export enum SupportedView {
18     CALENDAR = 'calendar',
19     KANBAN = 'kanban',
20     LIST = 'table',
21     GRID = 'grid'
22 }
23
24 export enum Access {
25     READ = 'read',
26     CREATE = 'create',
27     EDIT = 'update',
28     DELETE = 'delete',
29     DOWNLOAD = 'download'
30 }
31
32 export enum ListControllers {
33     PAGINATION = 'pagination',
34     SEARCH = 'search',
35     VIEW_CHANGER = 'view-changer',
36     ACTION_BUTTONS = 'action-buttons'
37 }
38
39 export type ListField = {
40     name: string; //name of the fields that should be displayed (if nested fields, use the dot notation)
41     type: string;
42     priority: number;
43 };
44
45 export type KanbanOptionsType = {
46     disableDraggingCards?: boolean;
47     disableSortingLanes?: boolean;
48     fieldKeys: {
49         groupBy: string;
50         cardName: string;
51         cardTag?: string;
52         cardPhoto?: string;
53         comments?: string;
54         attachments?: string;
55     };
56     //Colors
57     tagsBackgroundColors?: Record<string, string>;
58     lanesBackgroundGradients?: Record<string, string | string[]>;
59 };

```

The widget used to display the multiple paginated records of a module. It consists of:

1. Control bar
  - a. Action buttons (Add, edit, delete...etc.)
  - b. Search bar
  - c. Pagination
  - d. View Selector (from the supported views)
2. Records View
  - a. List View
  - b. Grid View
  - c. Kanban View

#### d. Calendar View

#### Configs attributes

Attribute	Tags	Description
<code>supportedViews</code>	<b>MENDATORY</b>	<p>An array of the supported views that the current module supports. The array elements could be any of the enumeration below:</p> <pre>1 export enum SupportedView { 2   CALENDAR = 'calendar', 3   KANBAN = 'kanban', 4   LIST = 'table', 5   GRID = 'grid' 6 }</pre>
<code>access</code>	<b>MENDATORY</b>	<p>An array of the access rights that the current user has for the current module. The array elements could be any of the enumeration below:</p> <pre>1 export enum Access { 2   READ = 'read', 3   CREATE = 'create', 4   EDIT = 'update', 5   DELETE = 'delete', 6   DOWNLOAD = 'download' 7 }</pre>
<code>fields</code>	<b>MENDATORY</b>	<p>Array of all the fields that could be displayed. Each element in the array should have a <code>name</code> , <code>type</code> and <code>priority</code>. Each attribute is discussed in details below.</p>
<code>initialFieldsLimit</code>	<b>OPTIONAL</b>  <code>DEFAULTS TO</code> <code>min( fields.length ,4)</code>	<p>The number of the initially shown fields before the user could select and unselect the fields to be shown. The fields that are initially shown are dependent on this attribute and the priory of the fields. If this attribute is set to 4, then the fields with the highest 4 priorities.</p>
<code>defaultView</code>	<b>OPTIONAL</b>  <code>DEFAULTS TO</code> <code>SupportedView.LIST</code>	<p>The default view of the module. It should be one of the supported views:</p> <pre>1 export enum SupportedView { 2   CALENDAR = 'calendar', 3   KANBAN = 'kanban', 4   LIST = 'table', 5   GRID = 'grid' 6 }</pre>

<code>defaultSearch</code>	<b>OPTIONAL</b> <b>DEFAULTS TO</b> <code>""</code>	The default search query to fill the search bar with initially.
<code>hideControllerBar</code>	<b>OPTIONAL</b> <b>DEFAULTS TO</b> <code>false</code>	Whether to hide the control bar all together and only show the records or display the control bar at the top of the records.
<code>customControllers</code>	<b>OPTIONAL</b> <b>DEFAULTS TO</b> <code>all controllers</code>	<p>If you want to customize the controllers visible in the control bar. You can provide a list with all the controllers you want to show.</p> <p>Here are the available controllers:</p> <pre> 1 export enum ListControllers { 2   PAGINATION = 'pagination', 3   SEARCH = 'search', 4   VIEW_CHANGER = 'view-changer', 5   ACTION_BUTTONS = 'action-buttons' 6 }</pre>
<code>modulePlural</code>	<b>OPTIONAL</b> <b>DEFAULTS TO</b> The module in the URL of the current page if present.	This is useful if you want to display a list of records of a specific module and you are in a different module. As by default the <code>modulePlural</code> is assumed to be the module in the URL of the current page.
<code>kanbanOptions</code>	<b>OPTIONAL</b>	Configurations used if the supported views include <code>SupportedView.KANBAN</code> . Each attribute in this object is explained in the Kanban Options attributes Section.

#### Field attributes

Attribute	Tags	Description
<code>name</code>	<b>MENDATORY</b>	The name of the field key that is used to access the data and construct the <code>gql</code> query. If the data to be accessed is nested use the dot notation.
<code>type</code>	<b>MENDATORY</b>	The type of that field. Should only contain primitives like string, number, boolean...etc. As that field is going to be displayed in the table. If there are nested modules then the dot notation is used till we reach a primitive.
<code>priority</code>	<b>MENDATORY</b>	Based on that attributes the fields are sorted to be displayed by the order of highest priority. Lower numbers indicate higher priorities.

#### Kanban Options attributes

Attribute	Tags	Description
<code>disableDraggingCards</code>	<b>OPTIONAL</b>	The cards are draggable by default. Meaning they could be sorted within their own lane or

	<p>DEFAULTS TO <code>false</code></p>	<p>moved around between different lanes. If you want to disable this feature set this attribute to <code>true</code>.</p>
<p><code>disableSortingLanes</code></p>	<p>OPTIONAL</p> <p>DEFAULTS TO <code>false</code></p>	<p>You can sort the lanes by default, but if you do not want them to be sortable set this attribute to <code>true</code>.</p>
<p><code>fieldKeys</code></p>	<p>MENDATORY</p>	<p>The name of the field keys used to access the data that is going to be displayed on different parts of the card. For example a card could look like that:</p> <div data-bbox="1055 514 1437 714"> </div> <p>You can determine the following field keys:</p> <ul style="list-style-type: none"> <li>• <code>groupBy</code> the key of the field to group the data by into lanes. MENDATORY</li> <li>• <code>cardName</code> The key used to access the text shown on the card from the outside. MENDATORY In the above example it represents this part: <div data-bbox="1079 1018 1258 1050"> <p>End User Flow Charts</p> </div> </li> <li>• <code>cardTag</code> the key used to access the tag shown in the card. OPTIONAL if it is not provided, no tag is shown. In the above example it represents this part: <div data-bbox="1079 1239 1169 1270"> <p>Medium</p> </div> </li> <li>• <code>cardPhoto</code> the key used to access the photo shown in the card. OPTIONAL if it is not provided, no photo is shown. In the above example it represents this part: <div data-bbox="1079 1459 1128 1512"> </div> </li> <li>• <code>comments</code> the key used to access the comments of this card. OPTIONAL if it is not provided no comments icon and count is shown. <div data-bbox="1071 1701 1128 1732">  0 </div> </li> <li>• <code>attachments</code> the key used to access the attachments of this card. OPTIONAL if it is not provided no attachments icon and count is shown. <div data-bbox="1071 1921 1128 1963">  0 </div> </li> </ul>



<code>tagsBackgroundColors</code>	<div>OPTIONAL</div>	<p>The colors used for the background of the tags. You provide an object where the key is the name of the tag and the value is the color you want.</p> <p>For example, the following object is provided for the following display:</p> <pre>1 { 2   [PriorityType.LOW]: '#ffc700', 3   [PriorityType.MEDIUM]: '#4f58a', 4   [PriorityType.HIGH]: '#fb5176' 5 }</pre> <div><div>Low</div><div>Dashboard Layout Design.</div><div><div></div>0<div></div>0<div></div></div></div> <div><div>Medium</div><div>End User Flow Charts</div><div><div></div>0<div></div>0<div></div></div></div> <div><div>High</div><div>Social Media Posts</div><div><div></div>0<div></div>0<div></div></div></div>
-----------------------------------	---------------------	---

## Addresses Widget

```
1 export type AddressesWidgetType = {  
2   type: WidgetTypes.ADDRESSES;  
3 };
```

The widget is used to display a list of addresses categorized by their type (billing or shipping.) It currently has no `configs` object, but we can add one later on to make it more customizable. The widget should get its data from an agreed upon back-end request.

It supports the following features:

- Adding a new address.
- Deleting or Editing an existing address.
- Selecting addresses.
- Map view.

### Shipping Address

☐ Customer: [Hossam Hemaya](#)

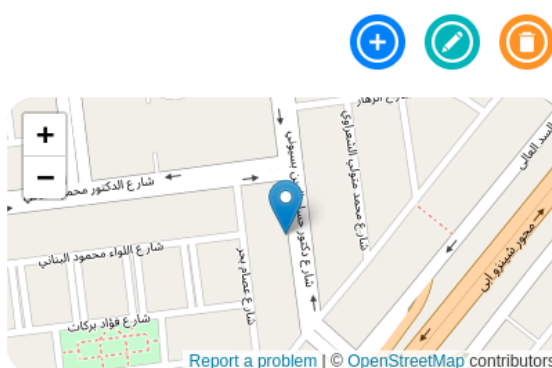
Address: Nasr City

Building: 445

City: Cairo

Country: Egypt

→ Direction



☐ Customer: [Hossam Hemaya](#)

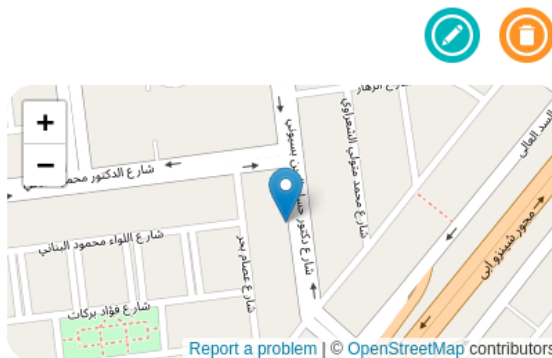
Address: Nasr City

Building: 445

City: Cairo

Country: Egypt

→ Direction



### Billing Address

☐ Customer: [Hossam Hemaya](#)

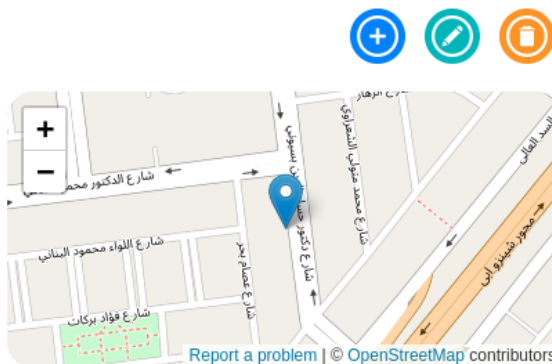
Address: Nasr City

Building: 445

City: Cairo

Country: Egypt

→ Direction



## ❏ Log Note Widget

```
1 export type LogNoteWidgetType = {  
2   type: WidgetTypes.LOG_NOTE;  
3 };
```

The widget used to log a new note and display the logged notes. It currently has no `configs` object, but we can add one later on to make it more customizable.

Write Something

😊 📎

Send

Today ^

Note By Hossam Hemaya  
hey 🌟 @MOSTAFA  
Attachment 1 Attachment 2

Apr 26, 2023 ^

Note By Hossam Hemaya  
hello

Feb, 2023 v

Mar, 2023 v

The widget has the following features:

- Add a new note
  - Normal Text
  - Attach files
  - Mention users
  - Add Emojis
- Displays the logged notes
  - Categorized by date
    - By day if in the current month
    - By month for previous months (Collapsed by default)
  - View text and mentions
  - Download attachments

On submission the following data is going to be sent to an agreed upon endpoint:

Attribute	Description
<code>text</code>	The text inside the <code>textarea</code> as a normal <code>string</code> including the mentions and emojis.
<code>mentionedUsersIds</code>	<code>string[]</code> of the IDs of the mentioned users in the text to be able to notify those users from the server.

attachments

Depending on the requirements this could either be:

- The files paths obtained from the document manager integrated with AWS.
- The files as binary sent through `FormData`.

The widget should get its data from an agreed upon back-end request. The following is the expected type of each note in the displayed logged notes:


```
1 export type LoggedNoteType = {
2   id: string;
3   text: string;
4   writtenBy: User;
5   createdAt: MomentInput;
6   updatedAt: MomentInput;
7   attachments?: { path: string; name: string }[]; //To be able to download the attachments.
8   mentions?: { name: string; id: string }[]; //To be able to highlight the mentions in the text.
9 };
```



## ☐ Branches Widget

```
1 export type BranchesWidgetType = {
2   type: WidgetTypes.BRANCHES;
3 };
```

The widget used to display all the branches with their schedules with the ability to add/edit a branch schedule. It currently has no `configs` object, but we can add one later on to make it more customizable.

Polo Clinic - Nasr City Branch







Working days

Sa Su Mo Tu We Th Fr

Working hours

06:05 : 08:05 12:05 : 14:05 1 >



Working days

Sa Su Mo Tu We Th Fr

Working hours

06:05 : 08:05 12:05 : 14:05

Polo Clinic - Nasr City Branch





Working days

Sa Su Mo Tu We Th Fr

Working hours

06:05 : 08:05 12:05 : 14:05

Edit Schedule

Add Schedule



Action		Attribute	Description
Add		branchId	The string id of the branch the schedule is being added to.
		schedule	It is of type <code>Omit&lt;ScheduleType, 'id'&gt;</code> .
Edit		schedule	It is of type <code>ScheduleType</code> .

Attribute	Description
branch	It is of type <code>Omit&lt;BranchType, 'id'   'schedules'&gt; &amp; {schedules : Pick&lt;ScheduleType, 'workingDays'   'workingHours'&gt;}</code> .

```

1  export type BranchType = {
2    id: string;
3    name: string;
4    schedules: ScheduleType[];
5    location?: Location;
6  };
7
8  export type ScheduleType = {
9    id: string;
10   workingDays: WeekDaysEnum[];
11   workingHours: PeriodTime[];
12   periodTime: PeriodTime;
13   sessionPeriodInMinutes: number;
14   spaceTimeInMinutes: number;
15   breakTime?: PeriodTime;

```

```

16 };
17
18 export enum WeekDaysEnum {
19     SAT = 'Sa',
20     SUN = 'Su',
21     MON = 'Mo',
22     TUE = 'Tu',
23     WED = 'We',
24     THU = 'Th',
25     FRI = 'Fr'
26 }
27
28 export type PeriodTime = { from: MomentInput; to: MomentInput };
29
30 export type Location = {
31     latitude: number;
32     longitude: number;
33 };

```

## Insurance Widget

```

1 export type InsuranceWidgetType = {
2     type: WidgetTypes.INSURANCE;
3     configs: {
4         companies: InsuranceCompany[];
5     };
6 };
7
8 export type InsuranceCompany = {
9     id: string;
10    name: string;
11    colors?: {
12        card?: string; //defaults to white
13        text?: string; //defaults to black
14        label?: string; //defaults to grey
15    };
16    svgIcon?: string; //defaults to no icon
17 };

```

## Free Form Widget

```

1 export type FreeFormWidgetType = {
2     type: WidgetTypes.FREE_FORM;
3     configs: {
4         sections: FreeFormSection[];
5     };
6 };
7
8 export type FreeFormSection = {
9     name: string;
10    label: string | LocalizationObject | null;
11    occupies?: Occupation;
12    fields?: FreeFormField[]; //Incase it is not nested inside another subsection
13    sections?: FreeFormSection[]; //In case it is nested inside another subsection

```

```

14 //If neither sections or fields is provided an empty widget is rendered
15 };
16
17 export type FreeFormField = {
18   name: string;
19   components: FreeFormFieldComponent[];
20 };
21
22 export enum FreeFormFieldComponentTypes {
23   TITLE = 'title',
24   CHECKBOX = 'checkbox',
25   TEXT_INPUT = 'text-input',
26   DROPDOWN_INPUT = 'dropdown-input'
27 }
28
29 export type FreeFormFieldComponent =
30   | FreeFormTitle
31   | FreeFormCheckbox
32   | FreeFormTextInput
33   | FreeFormDropdownInput;
34
35 export type FreeFormTitle = {
36   type: FreeFormFieldComponentTypes.TITLE;
37   configs: {
38     name: string | LocalizationObject;
39   };
40 };
41
42 export type FreeFormCheckbox = {
43   type: FreeFormFieldComponentTypes.CHECKBOX;
44   configs?: {
45     default?: boolean;
46   };
47 };
48
49 export type FreeFormTextInput = {
50   type: FreeFormFieldComponentTypes.TEXT_INPUT;
51   configs: {
52     placeholder: string | LocalizationObject;
53     validation?: Partial<ValidationObject>;
54   };
55 };
56
57 export type FreeFormDropdownInput = {
58   type: FreeFormFieldComponentTypes.DROPDOWN_INPUT;
59   configs: {
60     placeholder: string | LocalizationObject;
61     validation?: Partial<ValidationObject>;
62     options: { label: string | LocalizationObject; value: string }[];
63   };
64 };

```

## Objective

The aim of this widget is to create a free form widget with sections, and sub sections, where fields are placed freely according to the space they occupy and wrap to the next line. It will mainly be used to display the life style form, but could be reused for similar forms if needed.

The sections, subsections, fields, and the components inside each field are customizable in the following example.



Physical Exercise
☐ Exercise
Period / Day

Sleep
☐ Sleep at daytime
Hours of sleep

Diet

Meals Per Day
Meals Per Day
☐ Eats alone
☐ Coffee
Cups Per Day

☐ Soft Drinks (Sugar)
☐ Salt
☐ Currently On a diet
Diet Info

Lifestyle Information

Smoking
☐ Smokes
☐ Ex-smoker
Age started to smoke
Age in years
Cigarettes a day
☐ Passive Smoker
Age of quitting
Age in years

Drinking
☐ Drinks Alcohol
Age started to drink
Age in years
Beers / Day
Liquor / Day
☐ Ex alcoholic
Age quit drinking
Age in years
Wine / Day

Drugs
☐ Drug Habits
Age started drugs
Age in years
☐ Ex drug addict
☐ IV drug user
Age quit drugs
Age in years

Save

In the example of the above form, the following object structure is going to be submitted:

```

1 {
2   "physicalExercise": {
3     "exercise": {
4       "checked": false,
5       "selection": null
6     }
7   },
8   "sleep": {
9     "sleep": {
10      "checked": false,
11      "selection": {
12        "label": "1 hour of sleep",
13        "value": "1 hour of sleep"
14      }
15    }
16  },
17  "diet": {
18    "mealsPerDay": {
19      "selection": null

```

```
20     },
21     "eatsAlone": {
22         "checked": false
23     },
24     "coffee": {
25         "checked": false,
26         "selection": null
27     },
28     "softDrinks": {
29         "checked": false
30     },
31     "salt": {
32         "checked": false
33     },
34     "currentlyOnDiet": {
35         "checked": false,
36         "text": ""
37     }
38 },
39 "lifestyle": {
40     "smoking": {
41         "smokes": {
42             "checked": false
43         },
44         "exSmoker": {
45             "checked": false
46         },
47         "ageStartedToSmoke": {
48             "text": ""
49         },
50         "cigarettesPerDay": {
51             "text": ""
52         },
53         "passiveSmoker": {
54             "checked": false
55         },
56         "ageOfQuitting": {
57             "text": ""
58         }
59     },
60     "drinking": {
61         "drinksAlcohol": {
62             "checked": false
63         },
64         "ageStartedToDrink": {
65             "text": ""
66         },
67         "beersPerDay": {
68             "text": ""
69         },
70         "liquorPerDay": {
71             "text": ""
72         },
73         "exAlcoholic": {
74             "checked": false,
75             "text": ""
76         },
77         "ageQuitDrinking": {
```

```

78         "text": ""
79     },
80     "winePerDay": {
81         "text": ""
82     }
83 },
84 "drugs": {
85     "drugHabits": {
86         "checked": false
87     },
88     "ageStartedDrugs": {
89         "text": ""
90     },
91     "exDrugAddict": {
92         "checked": false
93     },
94     "ivDrugUser": {
95         "checked": false
96     },
97     "ageQuitDrugs": {
98         "text": ""
99     }
100 }
101 }
102 }


```

given that the following configs are used for the widget:

You can find the example configs file in the following path in the AHD dashboard repo.

```
src/components/Widgets/testWidgetBackendConfigs/freeFormWidget.ts
```

### Configs attributes

Attribute	Tags	Description
sections	MENDATORY	<p>The main sections this form has. They look like this:</p>  <p>It is an array where each element is an object with the following attributes <code>name</code>, <code>label</code>, <code>occupies</code>, <code>fields</code>, <code>sections</code>. Each one of these attributes is explained in details in the following section.</p>

### Section attributes

Attribute	Tags	Description
name	MENDATORY	The name that is going to be used in the final form submission object. I.e. the object key.
label	MENDATORY LOCALIZABLE	The label of the section to be displayed in the UI.
occupies	OPTIONAL	How much this section occupies out of the total width. It is of type <code>Occupation</code> and

		works the same way as explained previously.
fields	OPTIONAL	In case there are no sub sections. You add the fields that represent field components or titles. It as an array of objects where each object has a <code>name</code> , and <code>components</code> attribute. Each one of these attributes is explained in details in the Field attributes Section.
sections	OPTIONAL	<p>An array of nested subsections. It has the same exact type as its parent. Which allows us to have as many nested subsections as needed. The top level section looks like that:</p> <pre>Physical Exercise</pre> <p>while any subsequent sections look like that:</p> <pre>Smoking</pre> <p>and depending on the level of nesting, they get a start padding, the deeper they are nested the larger the padding.</p>

#### Field attributes

Attribute	Tags	Description
name	MENDATORY	The name that is going to be used in the final form submission object. I.e. the object key.
components	MENDATORY	<p>The different components this field renders. It is an array of objects. There are mainly 4 different components that each field could render:</p> <pre>1 export enum FreeFormFieldComponent 2   TITLE = 'title', 3   CHECKBOX = 'checkbox', 4   TEXT_INPUT = 'text-input', 5   DROPDOWN_INPUT = 'dropdown-input' 6 }</pre> <p>For example the following field:</p> <div> <input type="checkbox"/> Exercise       <div>Period / Day</div> </div> <p>This field has three components in it:</p> <ul style="list-style-type: none"> <li>checkbox</li> <li>title</li> <li>dropdown</li> </ul> <p>And they are displayed by the same order you provide them. If we wanted a text input instead of a dropdown for the most right field</p>

we would have used a text input component instead. and if we only wanted a checkbox with a title then we use the checkbox and title components...etc. Each component type could have some configs which are explained in the following section.

In the above example, if the field has a name attribute of `exercise` then the final object to be submitted would contain an object with the following object:

```
1 {
2   exercise:{
3     checked: false,
4     selection : {label: "", something:
5   }
6 }
```

So each added component maps to a key in the form object except for the title as it is just for display. The mapping of each component to its object key is illustrated in the following section.

#### Field Component configs attributes:

Type	Form Object Key	Configs
TITLE	-	<ul style="list-style-type: none"><li><code>name</code> the text of the title displayed. <a href="#">LOCALIZABLE</a></li></ul>
CHECKBOX	<code>checked</code>	<ul style="list-style-type: none"><li><code>default</code> a boolean value representing the initial value of the checkbox.</li></ul>
TEXT_INPUT	<code>text</code>	<ul style="list-style-type: none"><li><code>placeholder</code> the text used as a placeholder for the text input <a href="#">LOCALIZABLE</a></li><li><code>validation</code> It has the same type (<code>Partial&lt;ValidationObject&gt;</code>) as the validation object used in the Form widget.</li></ul>
DROPDOWN_INPUT	<code>selection</code>	<ul style="list-style-type: none"><li><code>placeholder</code> the text used as a placeholder for the dropdown <a href="#">LOCALIZABLE</a></li><li><code>validation</code> It has the same type (<code>Partial&lt;ValidationObject&gt;</code>) as the validation object used in the Form widget.</li><li><code>options</code> an array of options for the dropdown menu. Each option is an object with the the following attributes:<ul style="list-style-type: none"><li><code>label</code> The text to be displayed for each option <a href="#">LOCALIZABLE</a></li><li><code>value</code> A unique identifier for each option.</li></ul></li></ul>