**FORM BUILDER QUICK START GUIDE**

**1. INTRO**

The form builder is a tool to style and structure your dynamic forms without any HTML, CSS or Javascript knowledge. It also allows you to set up interactive logic, for example hiding/showing fields, validation, or something like adding a date picker etc. The following document will provide you a guide on how to use this tool.

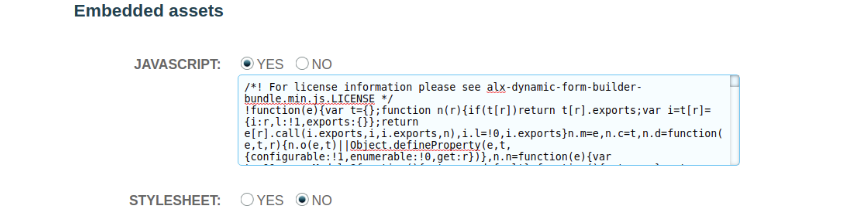
**2. INITIALIZATION**

The form builder is bundled and minified javascript code that takes in a configuration (provided by you) and uses it to do it’s “magic” under the hood. In order to make it part of the form you have to navigate to the dynamic form configuration and copy-paste it to the “Embedded assets – Javascript” section.

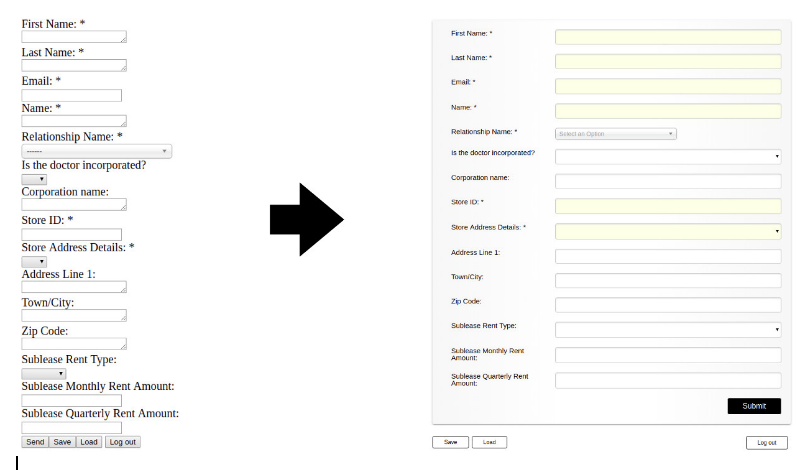
In the example shown in this quick tutorial the form is a skinny version of a sublease submission, where there are three Targets. Doctors, Stores and the Sublease (which is the Relational Target).

**IMPORTANT!**

Make sure you copy-paste the entire code. If anything is missing or you modify the code the form builder will not work.



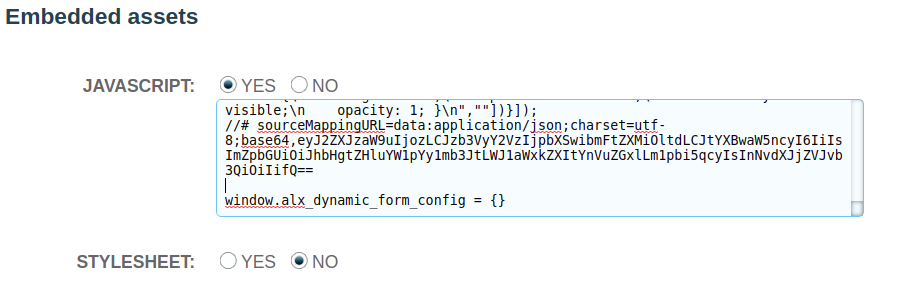
Save the settings and if you open the dynamic form you should see the form builder being initialized and applying the default styling and layout.



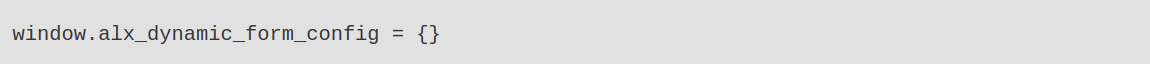
**3. CONFIGURATION BASICS**

The configuration tells the form builder what to do. It is a set of rules that gets interpreted by the form builder to set up the structure, the styling and the form’s internal logic.

It also gets inserted into the “Embedded assets – Javascript” section, underneath the code that you added in the previous step.



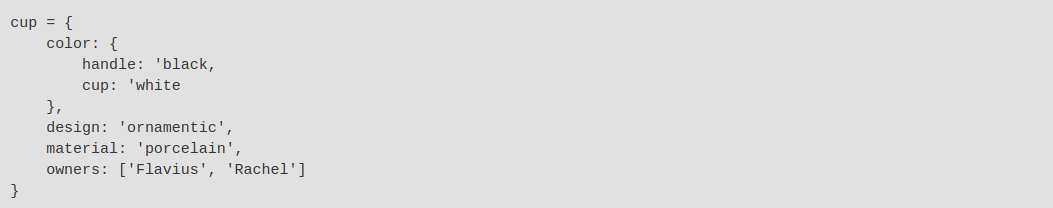
An empty configuration (the equivalent of providing no configuration) would look like this.

****

This is the Javascript Object which we are going to gradually extend to configure our form.

**What is a Javascript Object?**

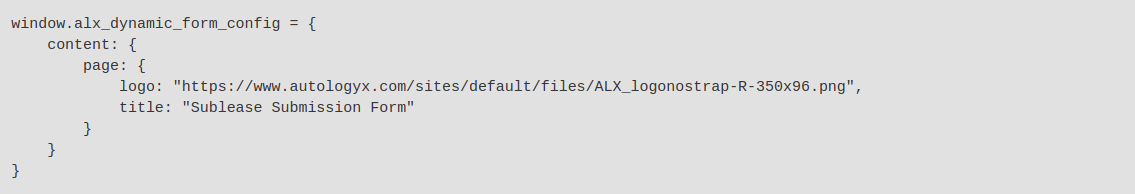
Objects in JavaScript can be compared to objects in real life. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. A cup described as a Javascript Object could look something like this.



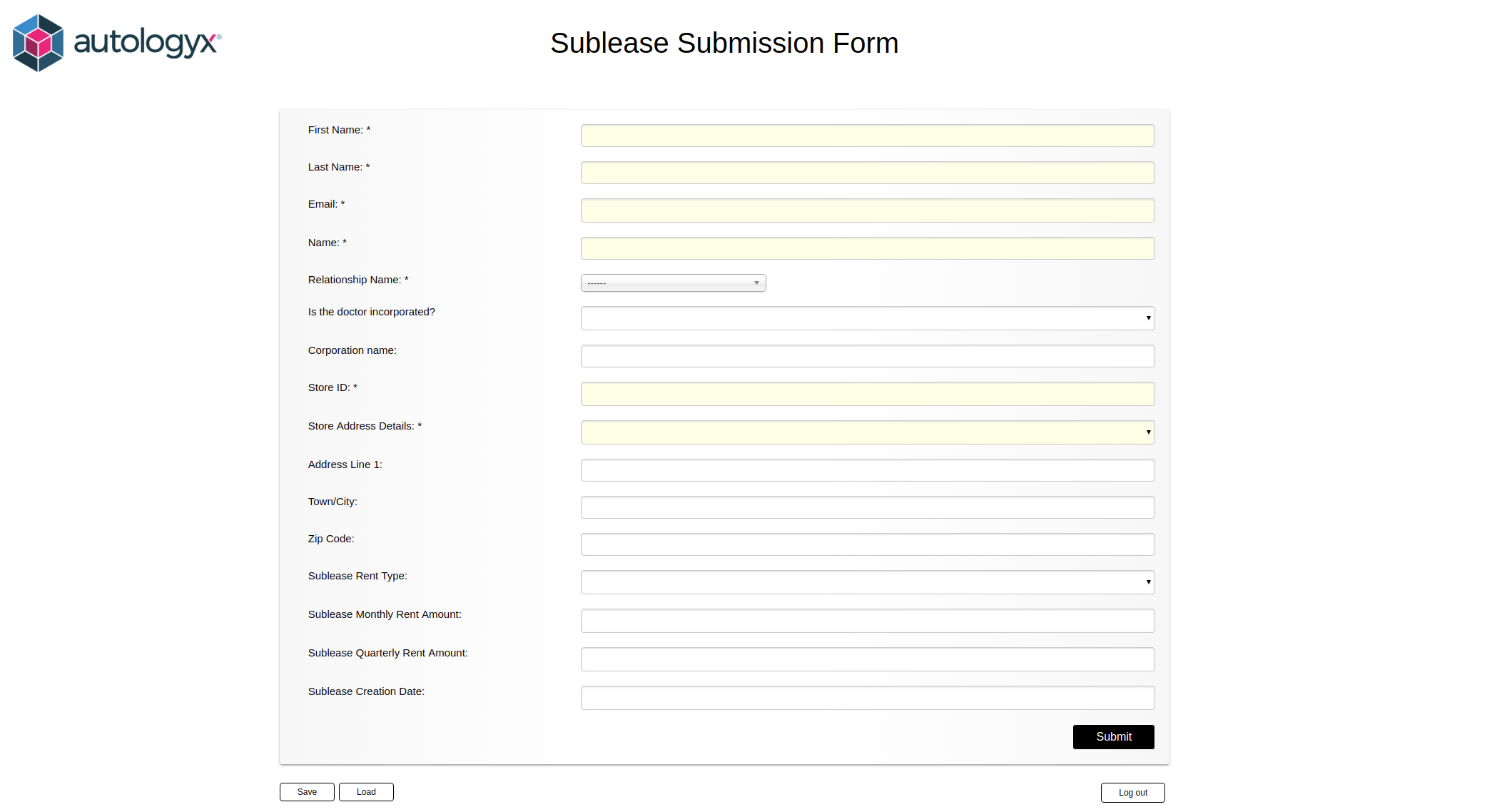
**4.0. BASIC FORM EDITING**

**4.1. ADDING LOGO AND TITLE**

As a first step let’s enhance our form by adding a logo and a title to it by changing the configuration.



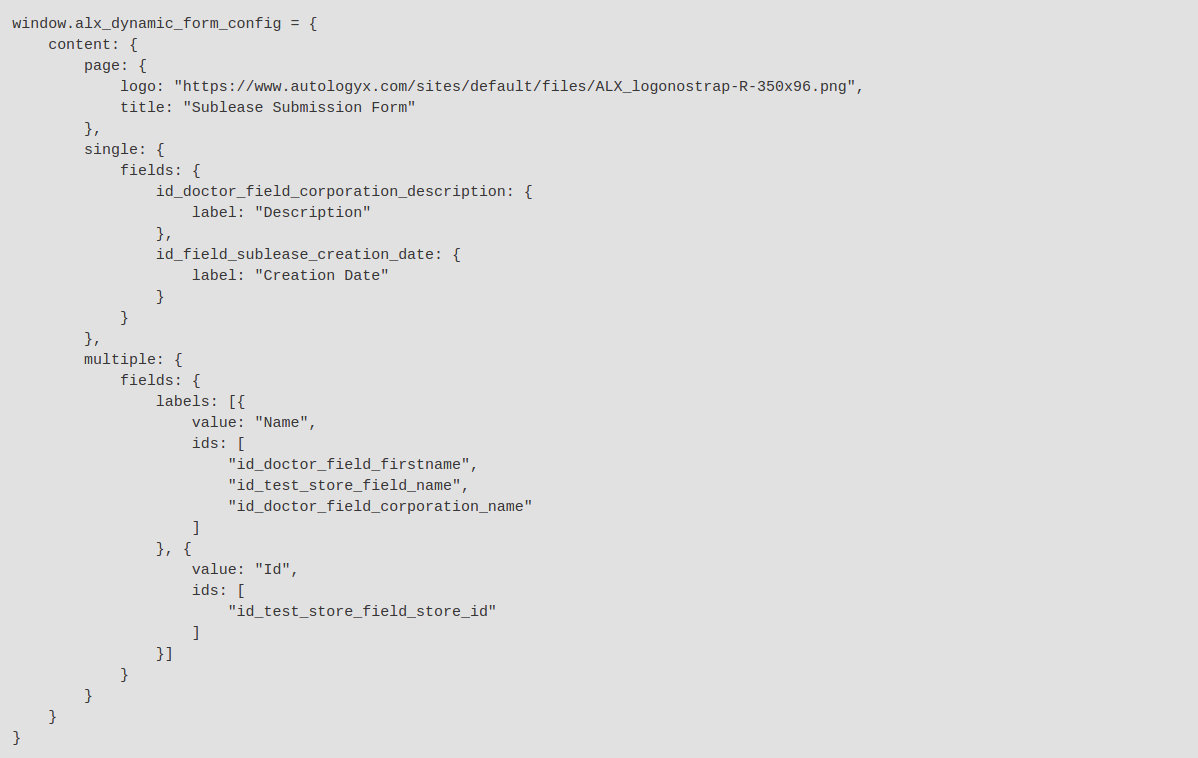
The above configuration will yield the following result:



**4.2. CHANGING FIELD LABELS**

Modifying fields, like changing the label, adding a placeholder, or just changing the field type is quite a common task. The form builder gives you quick access through the configuration to do all of these. First, let’s change the label for individual fields.



And if you want to add the same label to several fields at once you only have to do the following.

**IMPORTANT!**

As you can see you target single fields by using the “single” property and multiple fields at once with the “multiple” property.

Single fields:

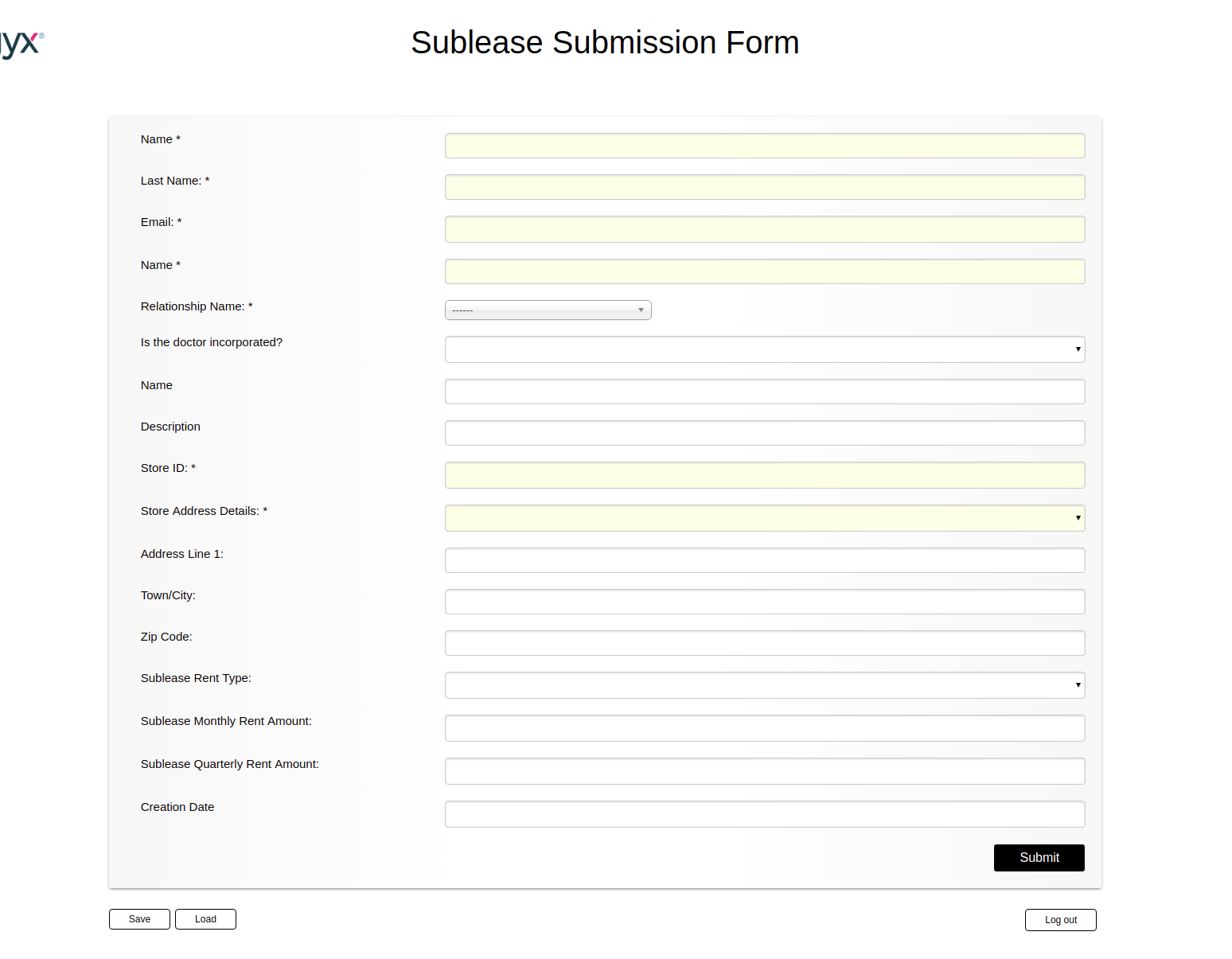
First you tell the form builder which field you want to modify (using its id) and then you tell it what logic apply to the field.

Multiple fields:

First you tell the form builder what is the logic you want to apply, then you list the fields you want the logic to be applied to.

Also notice, that when you are defining multiple labels the data structure the form builder is expecting is a list of items (array).

Now we got some of the labels changed.



**4.3. CHANGING FIELD PLACEHOLDERS**

Changing placeholders of fields is almost identical to changing the labels. All you have to do is use the “placeholder” keyword instead of “label”. First let’s add a placeholder to one of the fields.



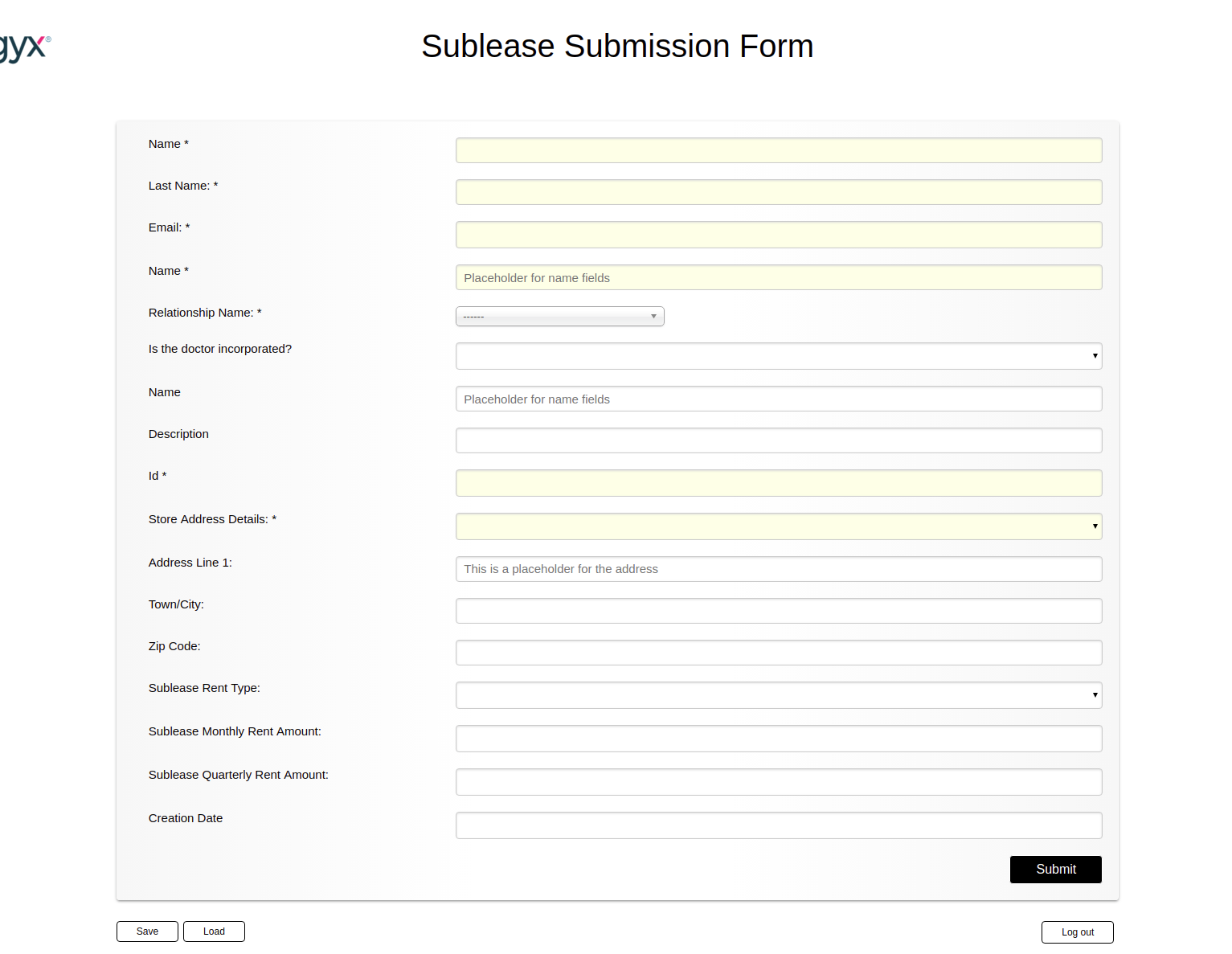
And then change the placeholder for some of the name fields where we changed the label.



**IMPORTANT!**

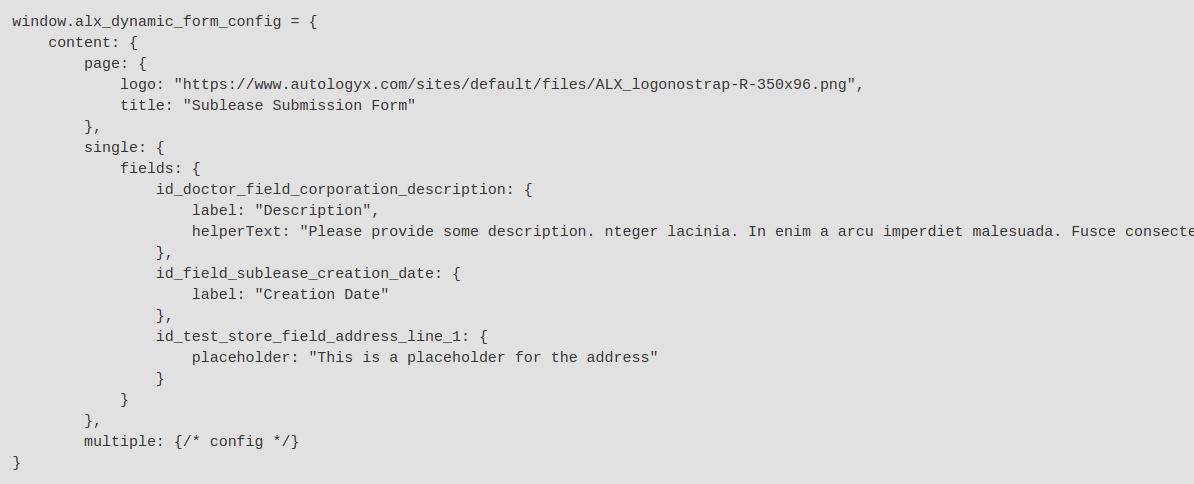
Placeholders are only applied to fields that accept a text as an input.

Now with these changes we added some placeholder texts too.



**4.4. ADDING HELPER TEXTS TO FIELDS**

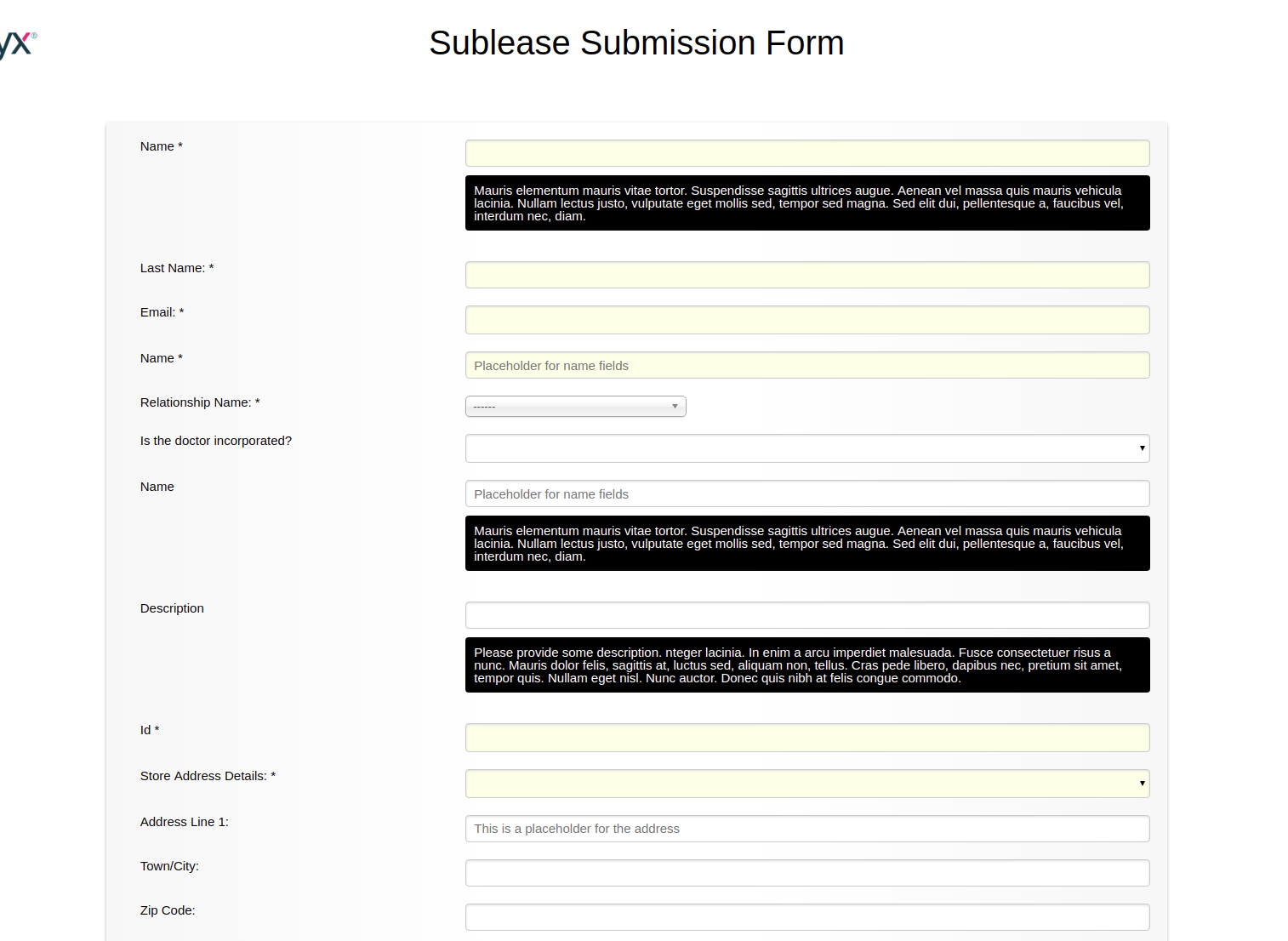
One fairly common task whie modifying the forms is to add helper texts underneath the fields that guide the user how to fill the form. Using the same principle we just learned with the labels and placeholders we can quickly add helper texts. Let’s add some helper text to the description field so we can see how applying multiple logic to an individual field looks like.



Same logic as we have seen it before for adding multiple helper texts.

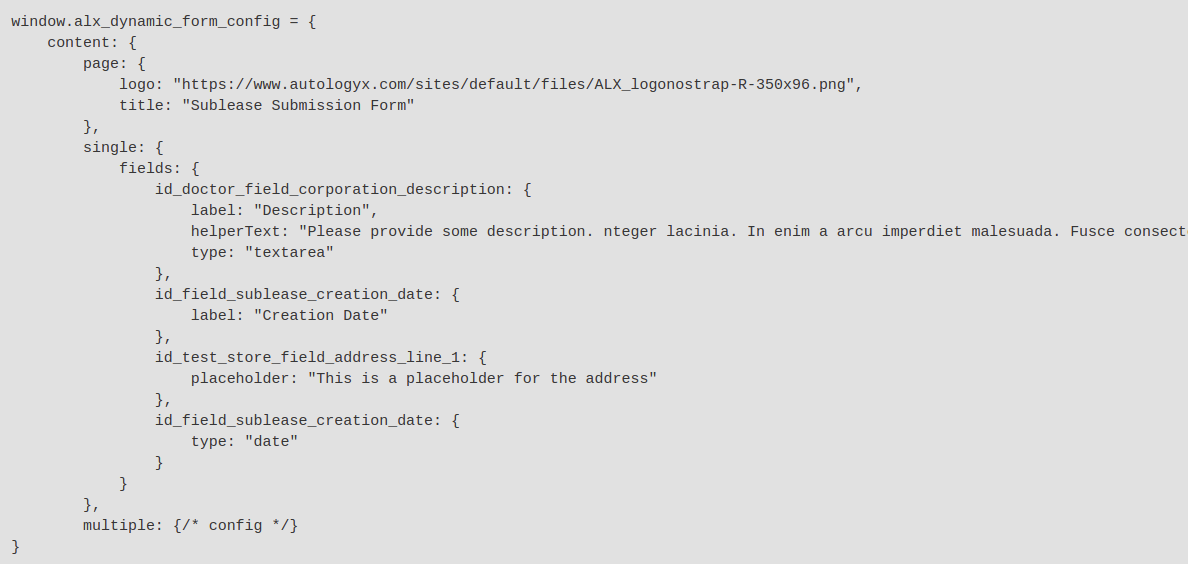


With the above changes this is the result we get.

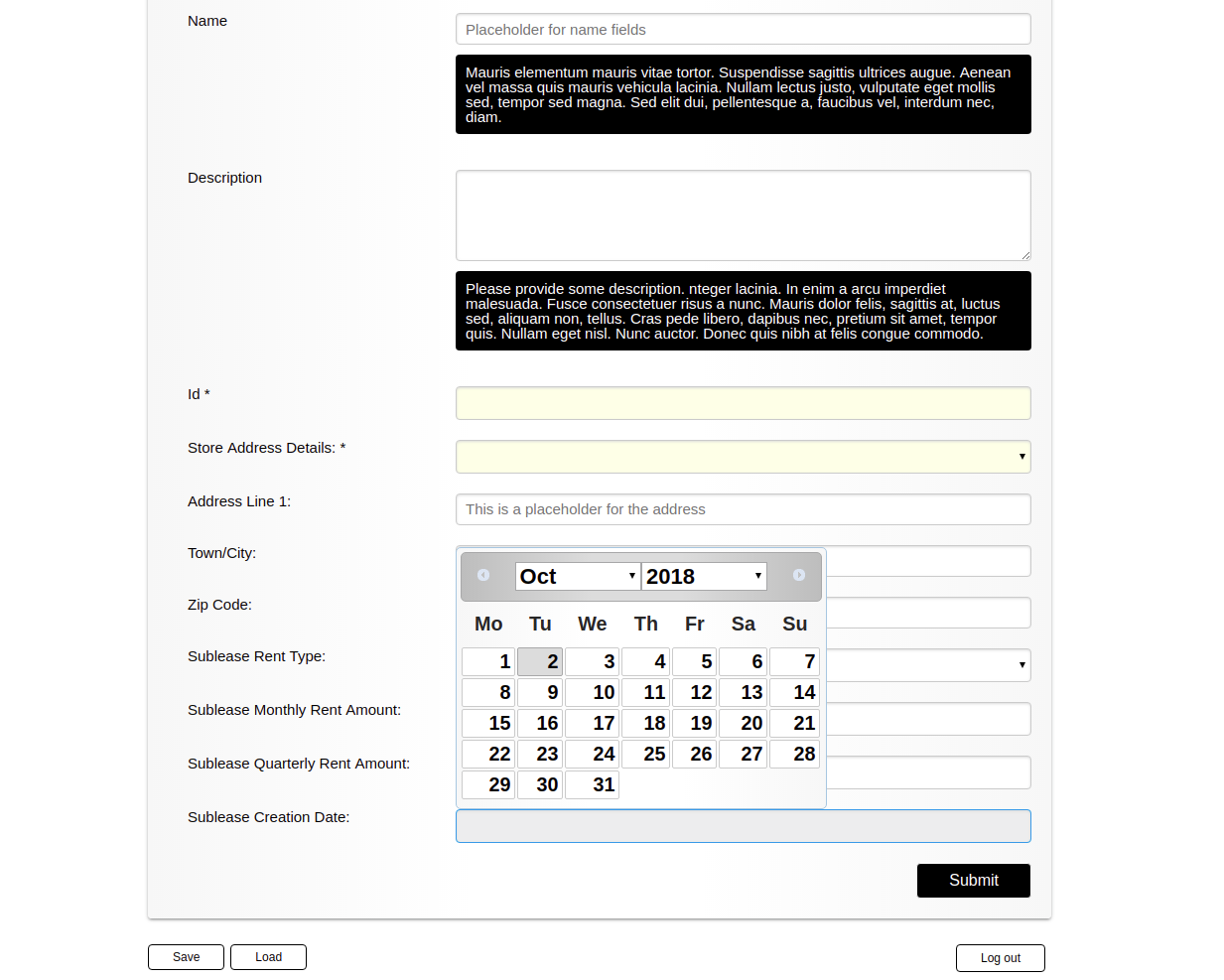


**4.5 CHANING FIELD TYPE**

By default a dynamic form has a limited set of field types, but what if we need a multi-line textarea or a date picker for a field? First let’s add the changes.



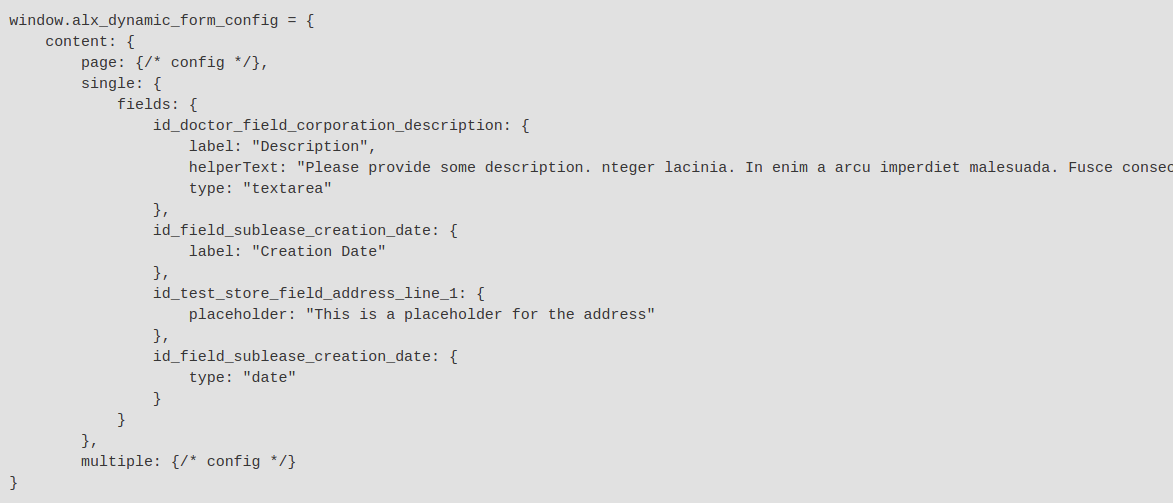
And below you can see how the end results look like.

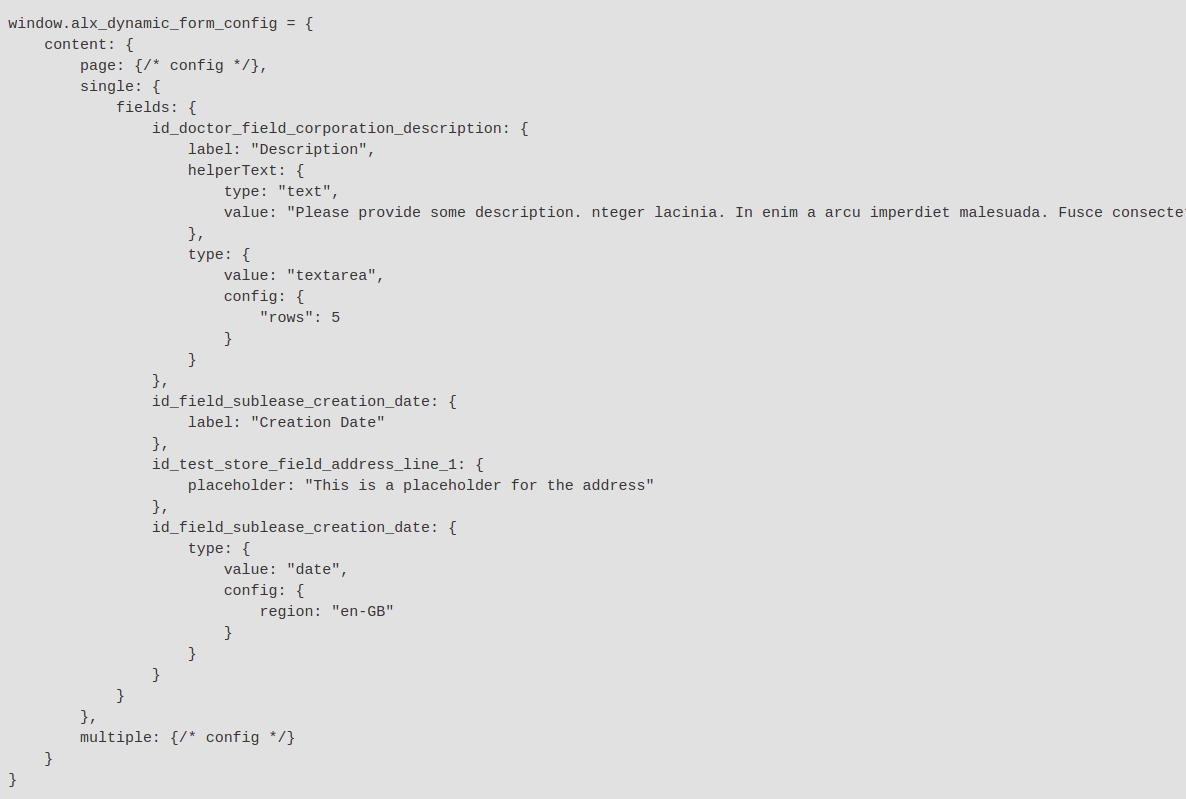


**4.6 ADVANCED FIELD CONFIGURATION**

Now lets stop here for a moment. In the above examples we were able to add quick changes, but soon we would run into certain limitations. What if we wanted to add more rows to textarea fields, what if we wanted to date field to use US rather than UK locale date. What about adding hyperlinks to helper texts?

The form builder allows you to do “finer” configuration. Take a look at the configurations below.



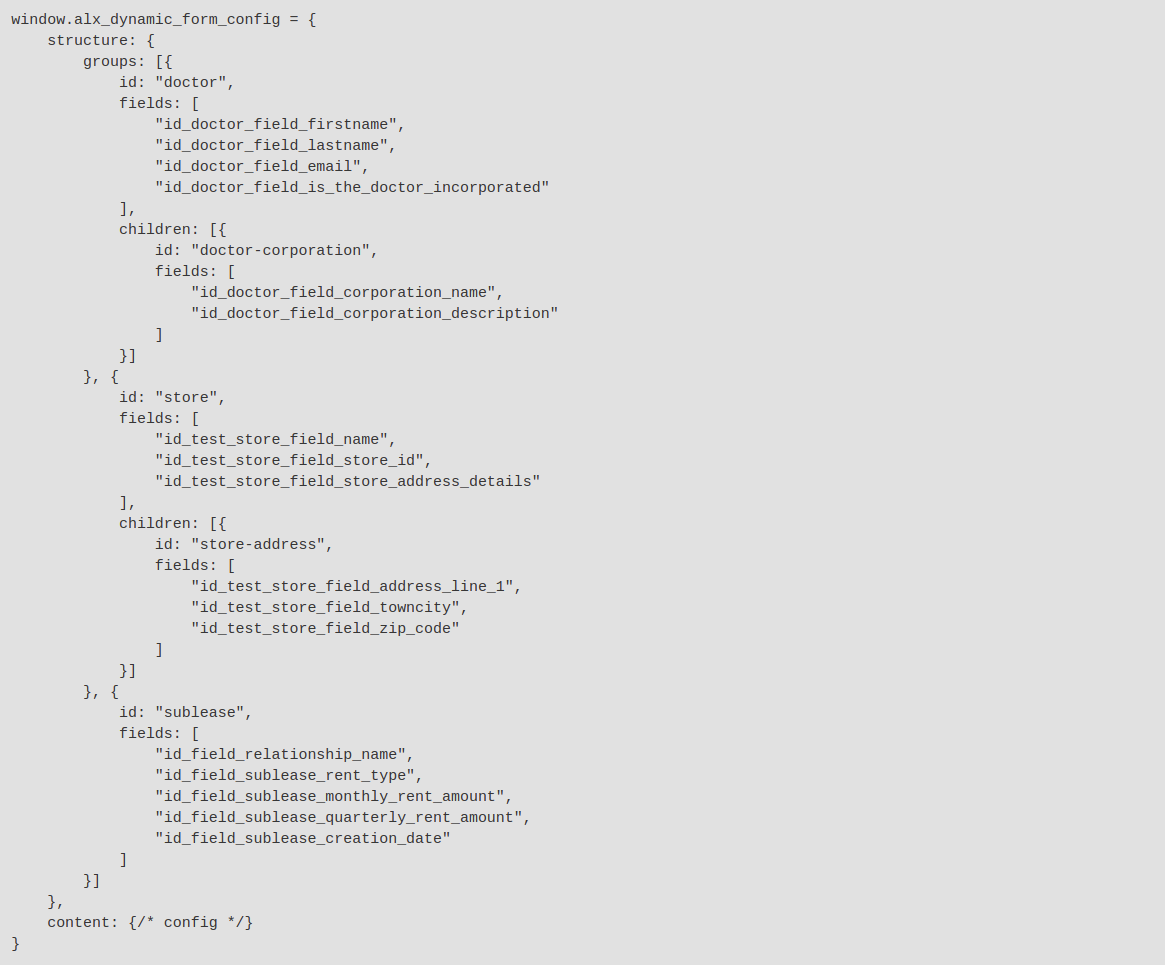


These two configurations provide the same result. The difference is that this time we provided “sub-configuration objects” instead of simple texts. You will learn more about that in the advanced section, but for now all you have to know that this is the way to do things, like injecting HTML content to the form.

**5.0 GROUPING FIELDS TOGETHER**

Now let’s do something a little bit more advanced. The form builder allows us to put fields into logical groups (or nest groups within groups). This will give us access to powerful features like pagination, or showing/hiding multiple fields at once.

If you remember here in this tutorial we have three Targets. Doctors, Stores and the Sublease (which is the Relational Target).



This setting on the surface will not change the form, but under the hood quite a few things happened. We told the form builder that we have five groups, “sublease”, “doctor”, “store”, “store-address” and “doctor-corporation”. Two of which are nested groups.

We also defined which fields belong to which group by using their ids.

**IMPORTANT!**

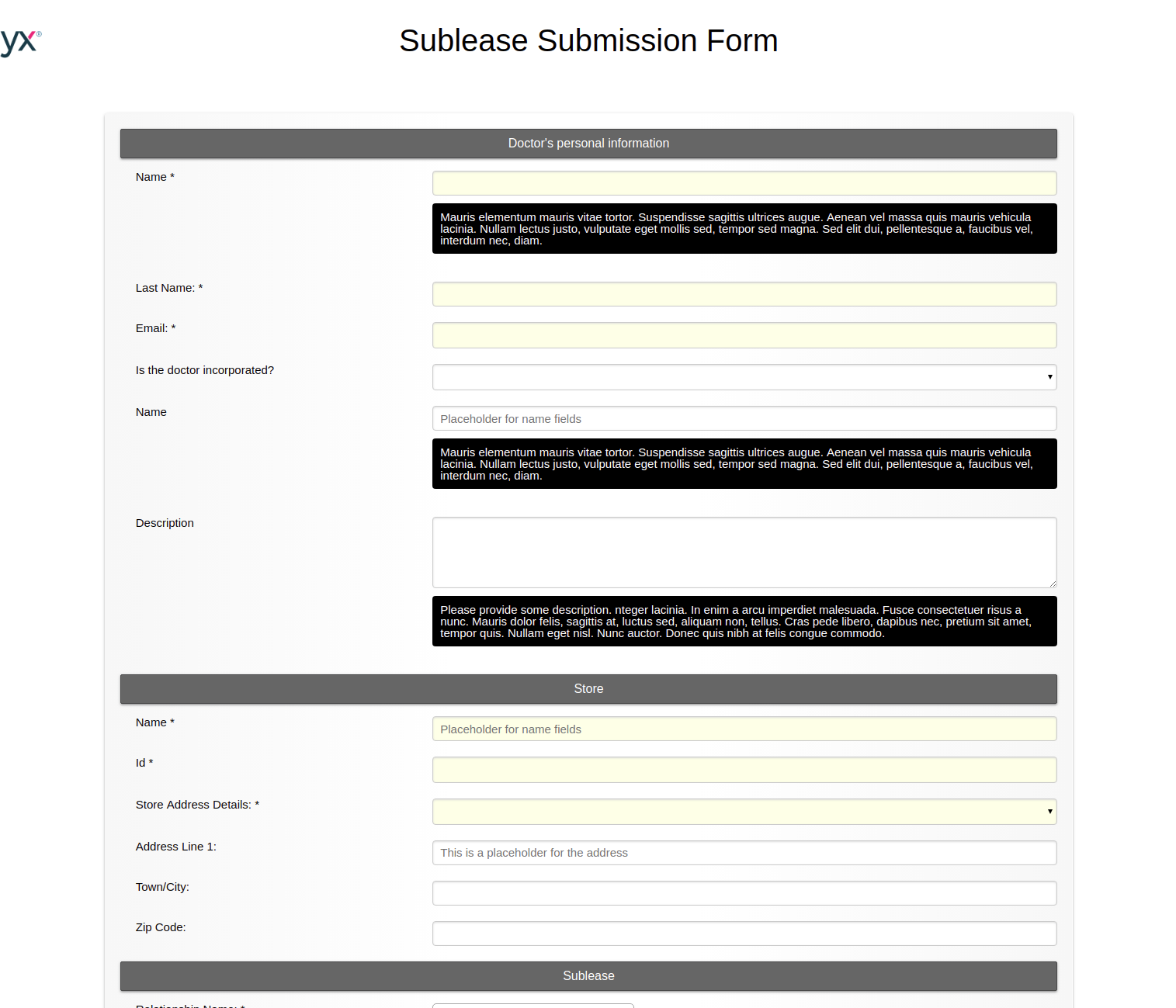
When you define groups you have to allocate all the fields, otherwise they will “fall out” from the form.

**5.1. ADDING HEADINGS TO GROUPS**

Now that we have groups of fields we can label them appropriately. Let’s do that by adding some headings.

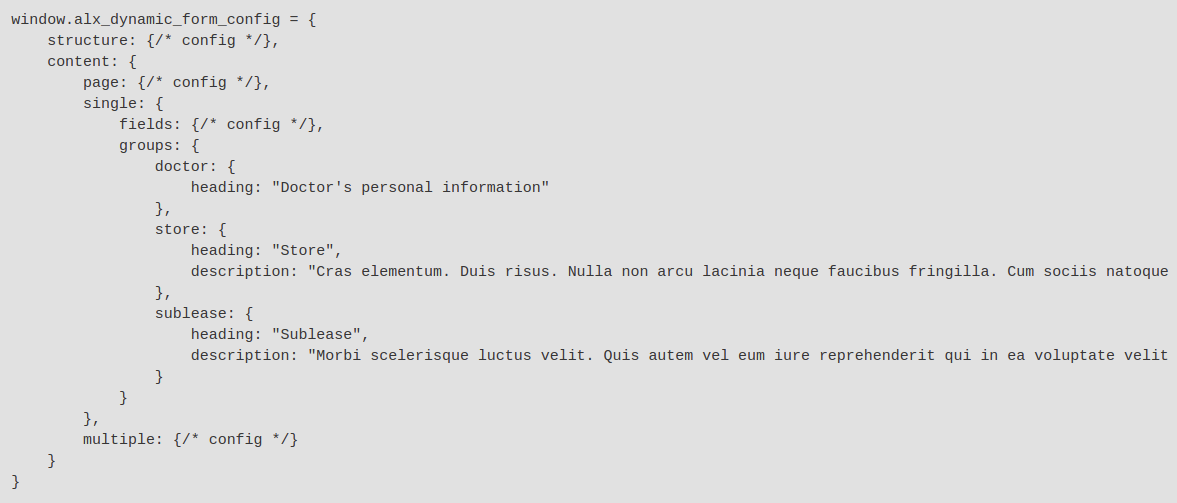


As you can see the logic is pretty similar to how we would configure our fields if we wanted to inject related content.

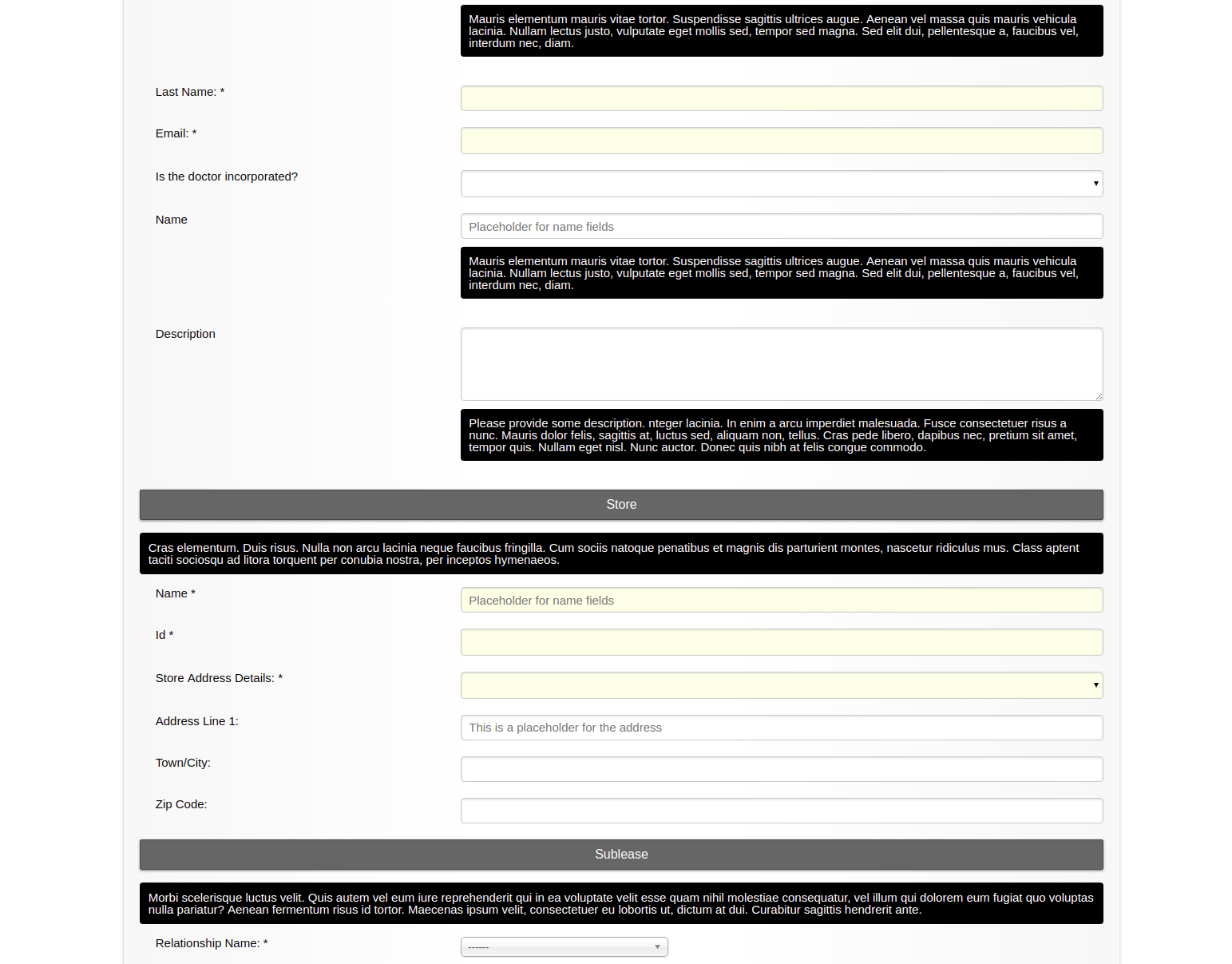


**5.2. ADDING DESCRIPTIONS TO GROUPS**

Now let’s add some descriptions to some of the groups.

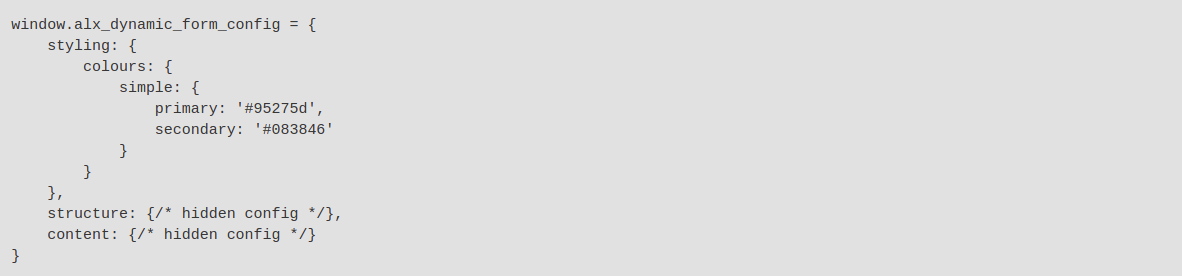


Now we managed to add some description to the groups.

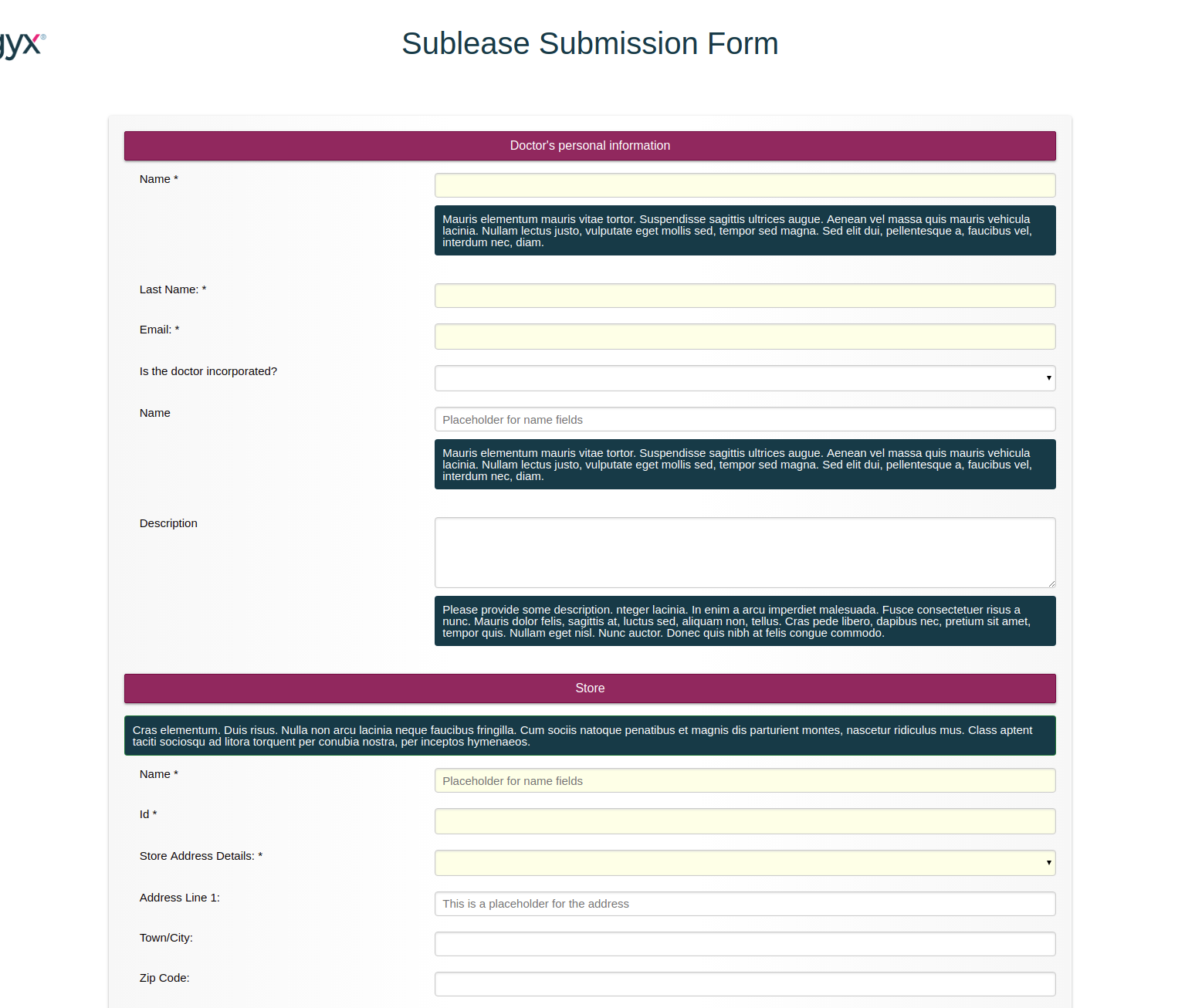


**6****.0. CHANGING COLOURS**

Now that we added some content and our form started taking shape we can see that by default ther form builder uses variations of grey and black to colour the form. The grey is the primary and the black is the secondary by default. If we want to change that we only have to do the following.



After defining the primary and secondary colours all of that gets applied to all the elements throughout the form to get the following result.

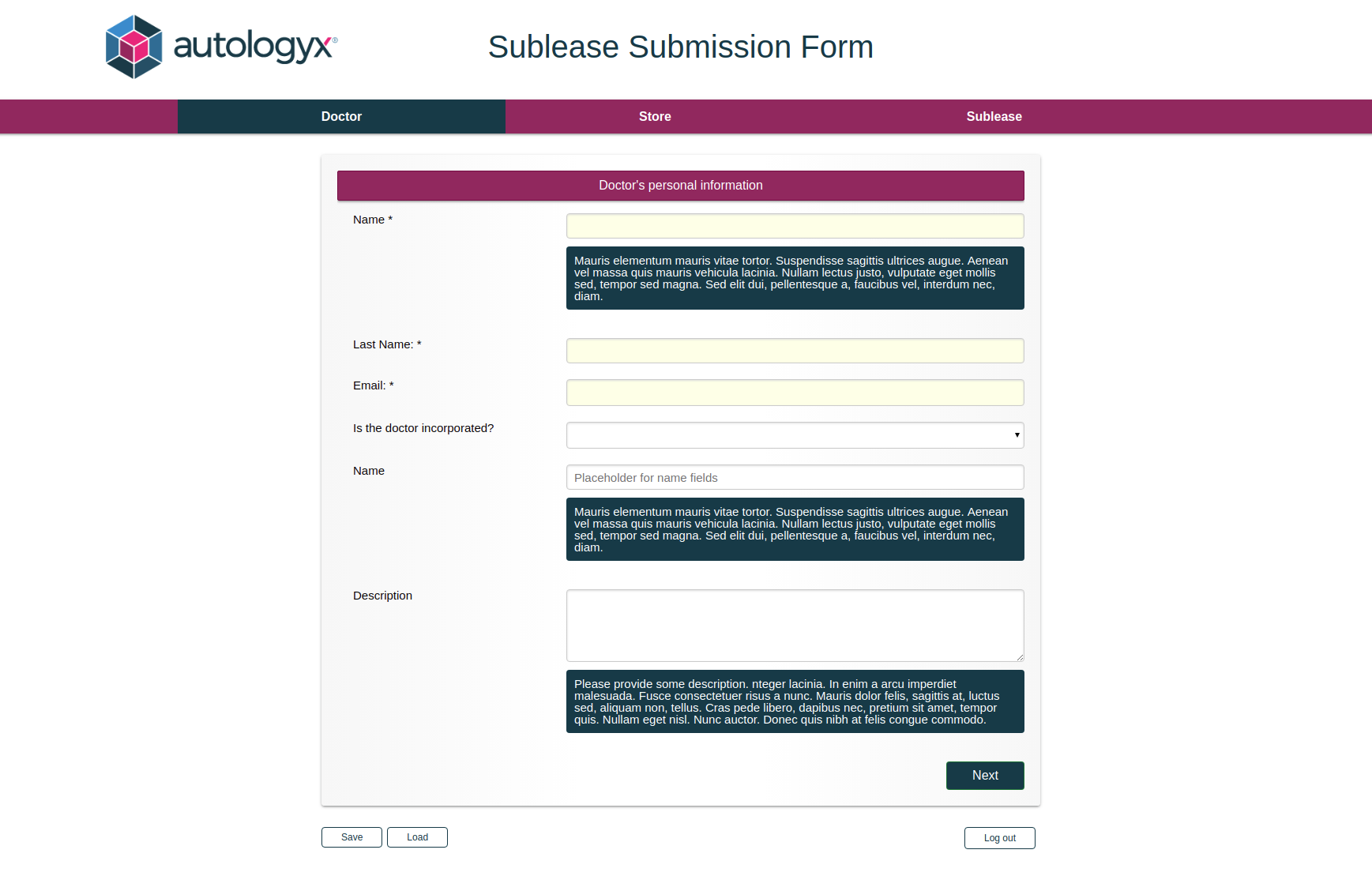


**IMPORTANT!**

When it comes to text the form builder runs a check against the background colour and picks white or black dependent on which one is the most contrasting.

**7.0. PAGINATION**

An other really powerful feature of the form builder is to have the ability to split long forms into several pages. After generating the groups adding pagination is very simple.



**IMPORTANT!**

When you enable pagination you have to include all the top-level groups in your configuration when you label them.

**8.0. STATE MANAGEMENT**

**8.1. FIELD STATE BASICS**

So far the only things we have been changing were related to layout and styling. An other important aspect of forms is hiding/showing fields at certain conditions, validating them or setting their values based on certain rules.

The most important part about state management is understanding the difference between “default” and “other” state. Every field (or group) has a default state (for example is it visible or not). Then these default states can change dependent on “conditions” that you define in your configuration (for example hide field X if field Y is set to “Yes”).

When it comes to state management you can target both fields and groups. These are the things you can modify via state management:

**Fields:**

- visibility

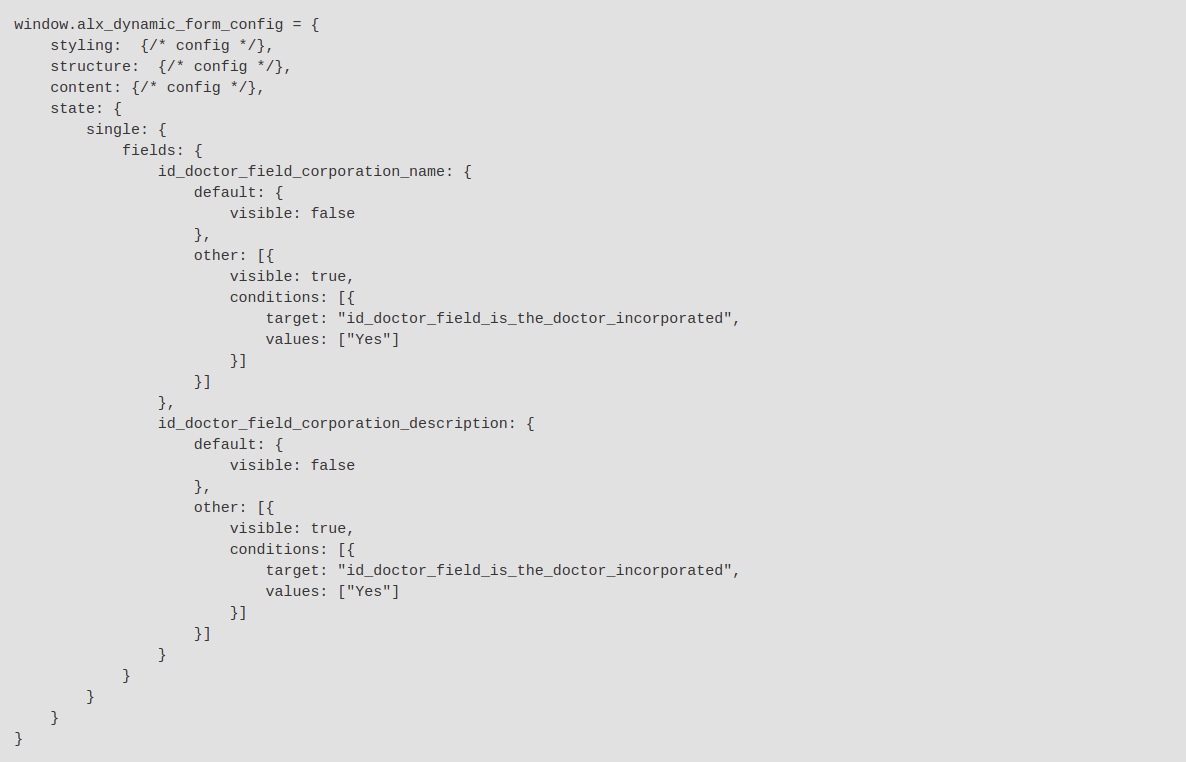
- value

- validation

**Groups:**

- visibility

Now let’s take a look how state configuration looks like in the form that we have been working on. We have two fields named “id\_doctor\_field\_corporation\_name” and “id\_doctor\_field\_corporation\_description” that we want to show only when the field “id\_doctor\_field\_is\_the\_doctor\_incorporated” is set to “Yes”.



If you take a look at the above configuration you can see that it has similar features to the other sub-configurations that we were using while adding descriptions, headings etc. First you tell which elements (“id\_doctor\_field\_corporation\_name” and “id\_doctor\_field\_corporation\_description”) are targetted and then you set up the logic.

If we break down the sub-configurations this is how they looks like:

- We have a “default” state for the fields

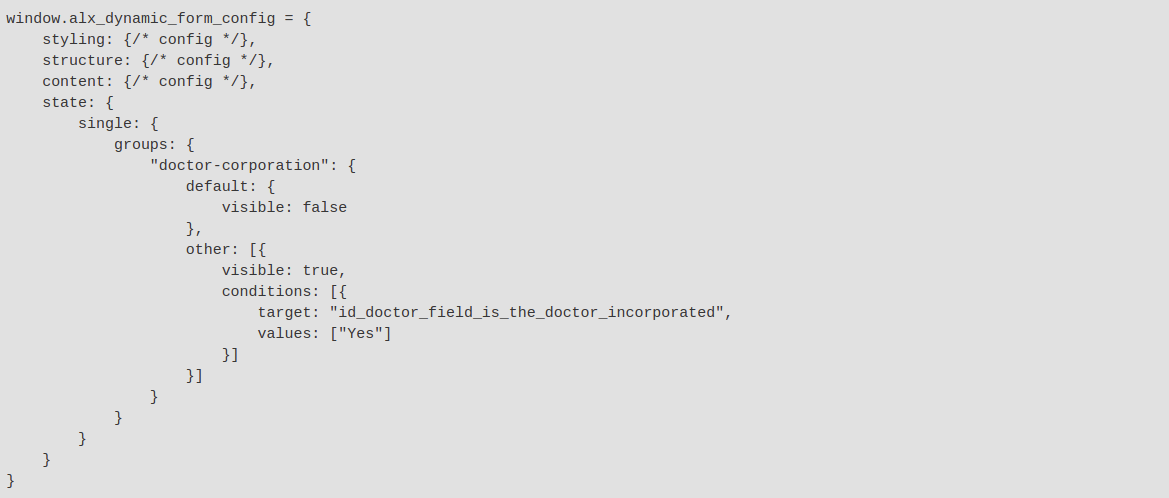
- We have a list of “other” states for the fields that are going to be used if certain conditions are met

In short, the form builder sets up an interconnected map between the fields and the groups under the hood and every time something changes it checks if it has to modify the state.

**8.2 GROUP STATE BASICS**

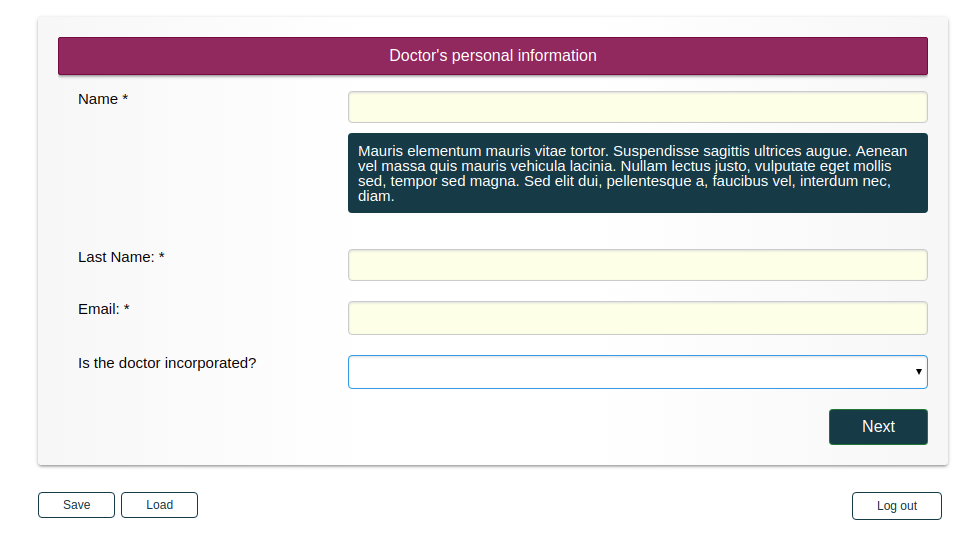
Now if you remember while we were configuring the groups earlier the two fields “id\_doctor\_field\_corporation\_name” and “id\_doctor\_field\_corporation\_description” were grouped together under the name “doctor-corporation”.

That means the previous task could have been achieved with the below configuration:



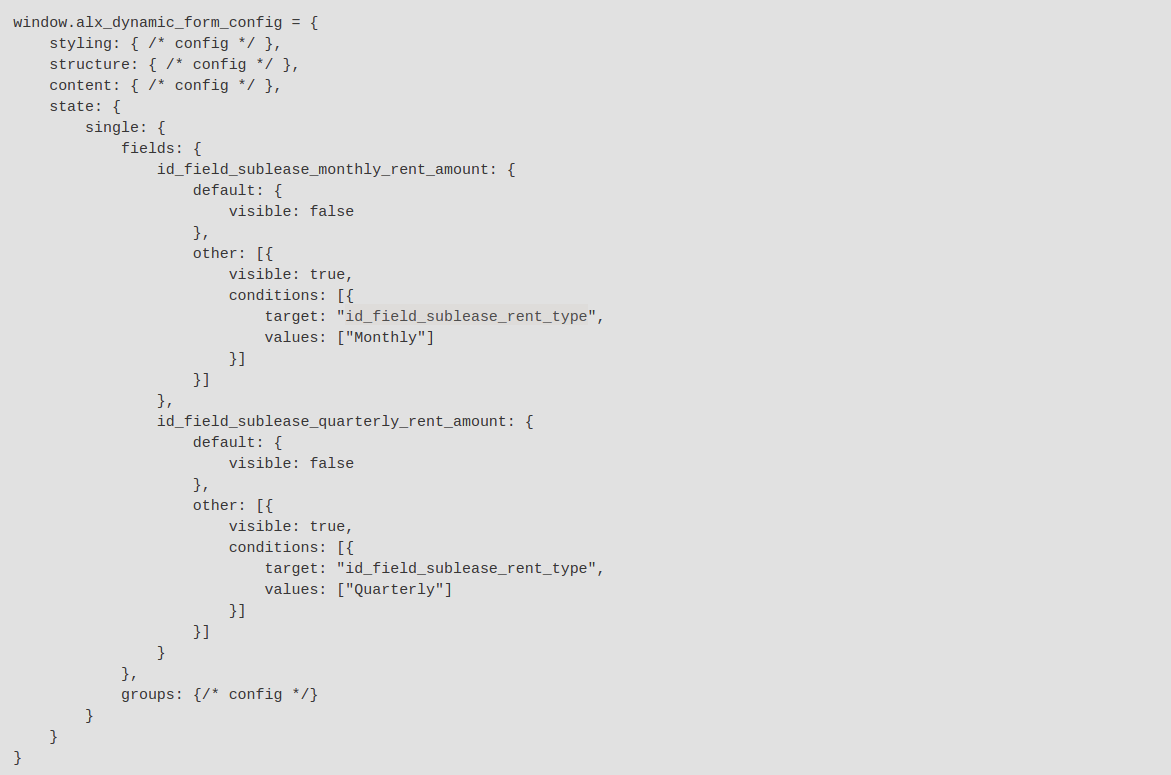
**IMPORTANT!**

In a Javascript Object you cannot use certain reserved characters as keys. In the above example in “doctor-corporation” the dash would be interpreted as a minus, therefore we had to put quotes around the word.

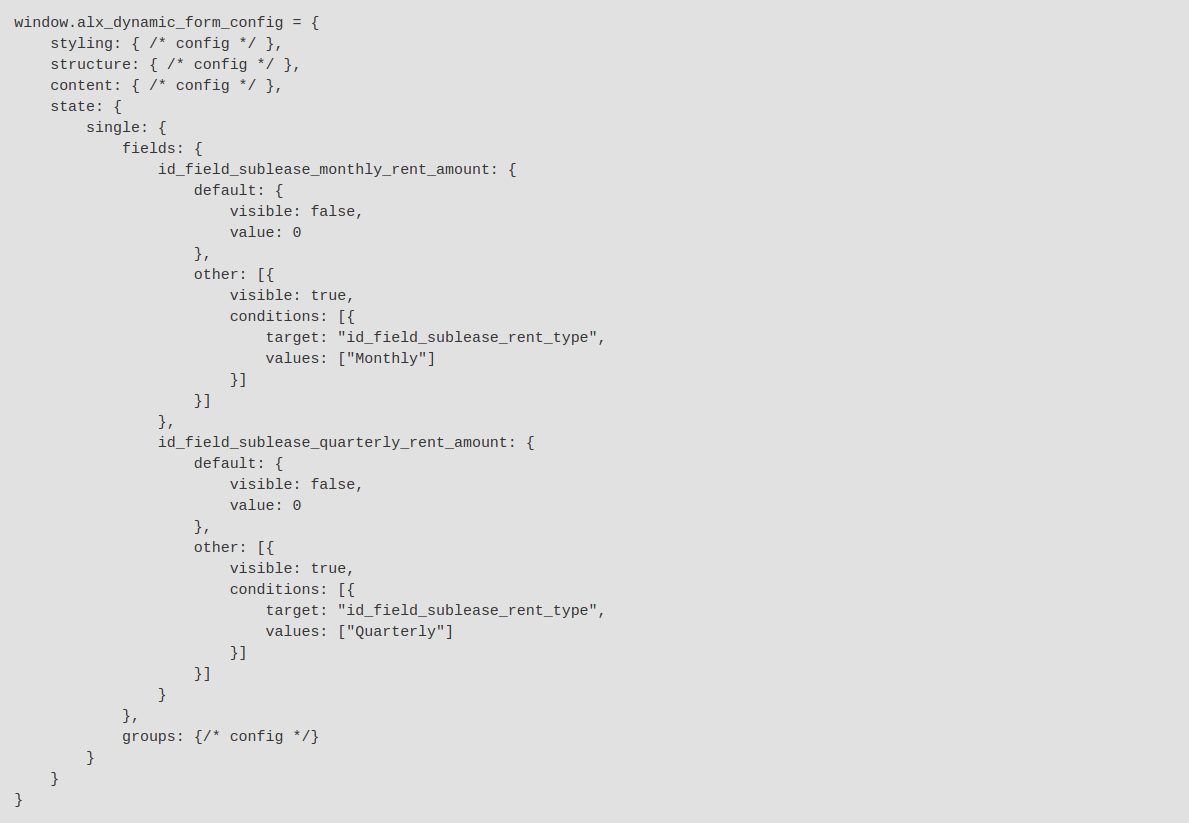


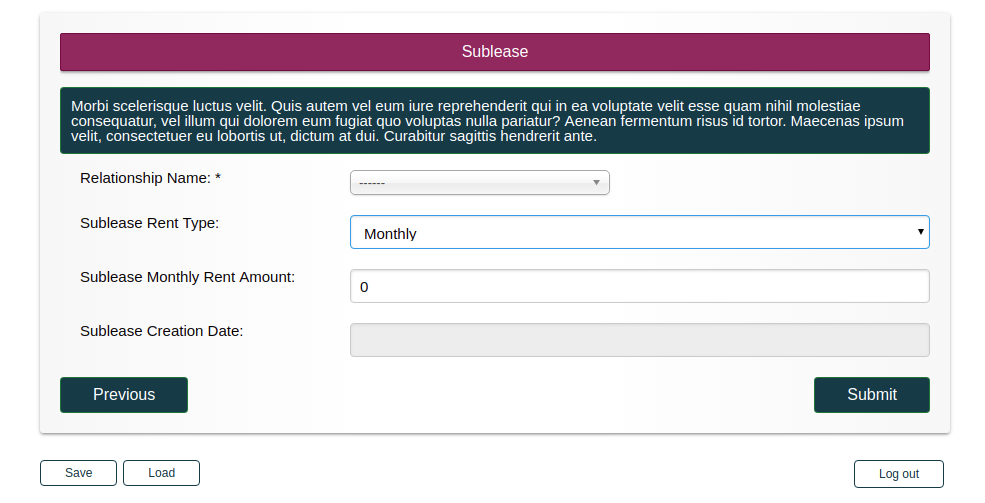
**8.3. SETTING FIELD VALUE**

The form used in this quick tutorial also has some fields called “id\_field\_sublease\_monthly\_rent\_amount” and “id\_field\_sublease\_quarterly\_rent\_amount” that are dependent on “id\_field\_sublease\_rent\_type”. Let’s set that up quickly.

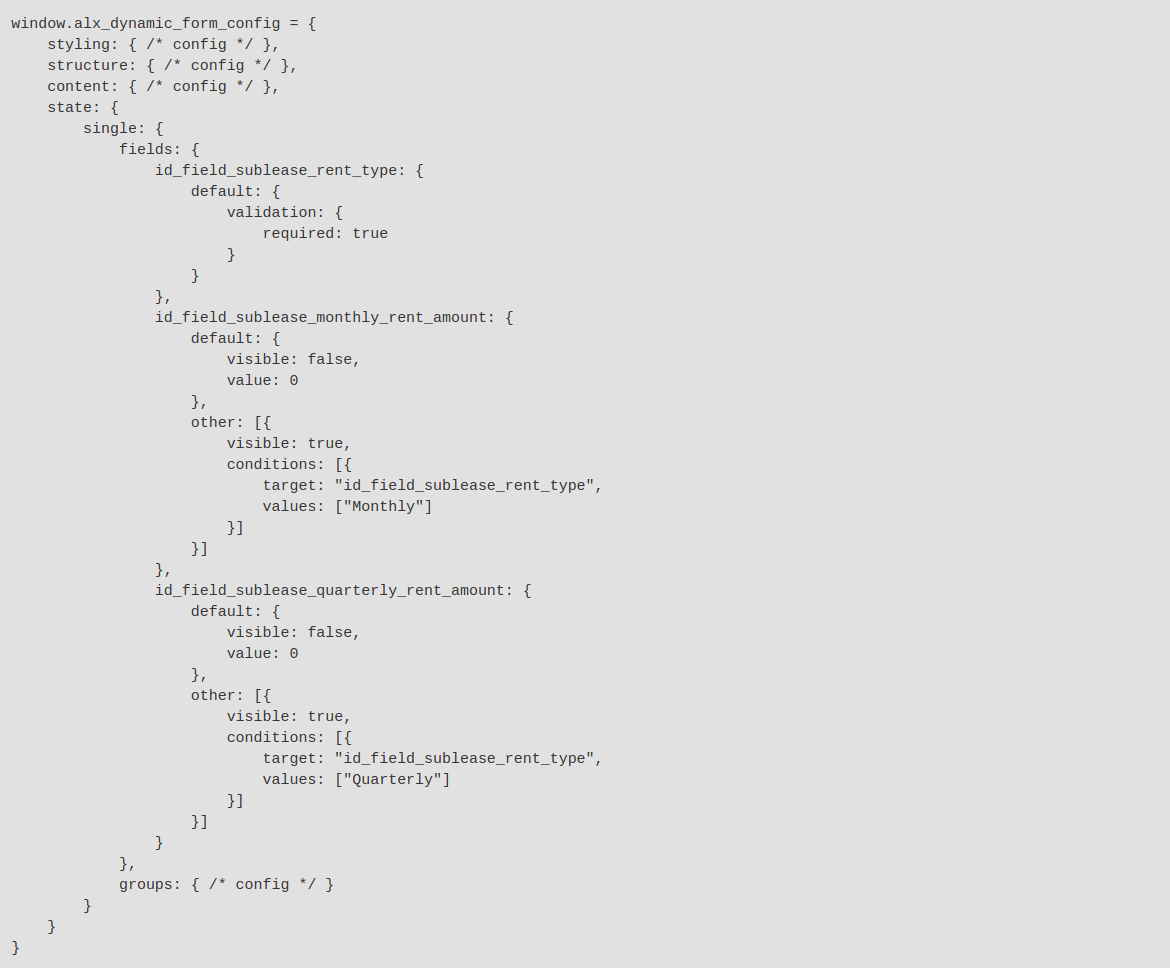


Now we set the default value 0 to the “id\_field\_sublease\_monthly\_rent\_amount” and “id\_field\_sublease\_quarterly\_rent\_amount” fields.



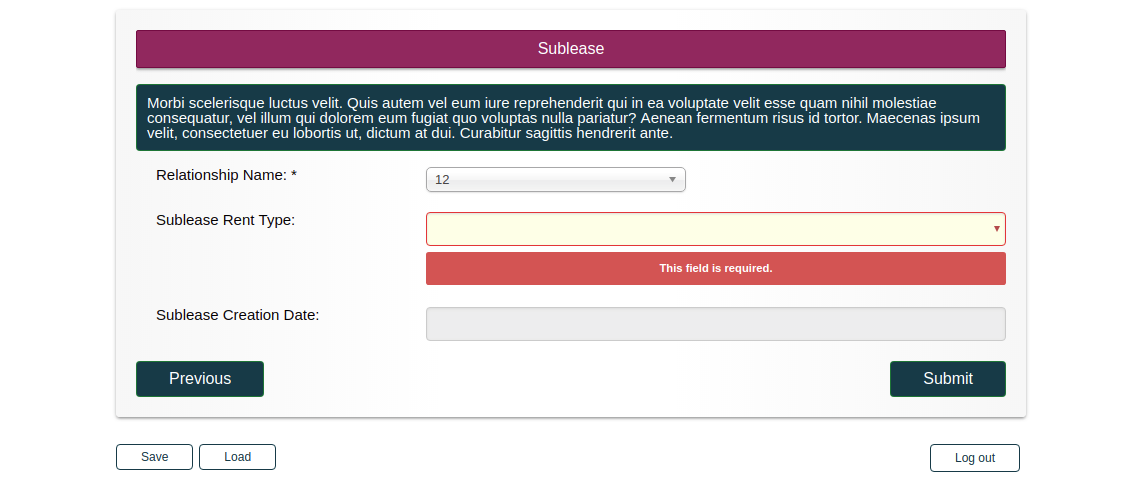


And let’s also set “id\_field\_sublease\_rent\_type” to mandatory.



Here the important takeaway, that you don’t have to define “other” states for fields or groups. This way you can set default states for fields and groups, for example making a field mandatory.

So now if we try to submit the form without giving a value to that field we will get an error.



**9.0. GENERATING UNIQUE ID**

One of the requirements when doing submissions (especially with Relational Targets) is to generate a unique identifier for the record before submitting it.

In the example used in this tutorial we have a “Relationship Name” field that we use for lookups and identifying relational records. The form builder allows you to auto-generate these IDs.

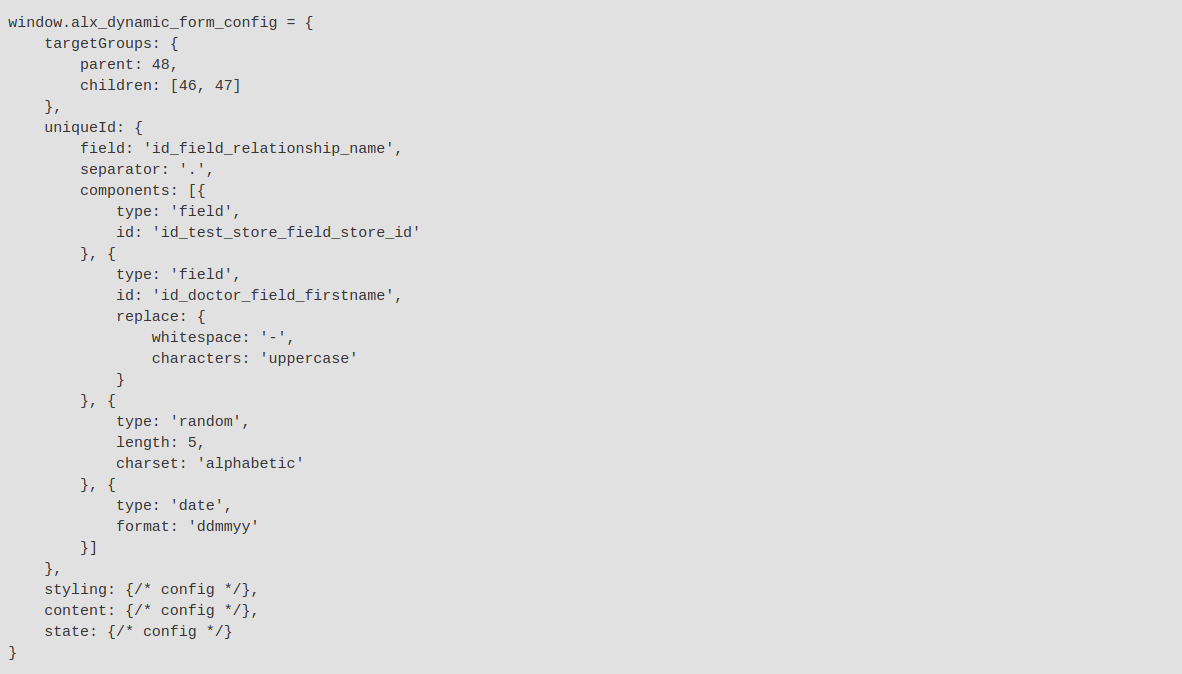
An ID is a chain of components joined together in a pre-defined order. These components can be:

- Random alphabetic or numeric string

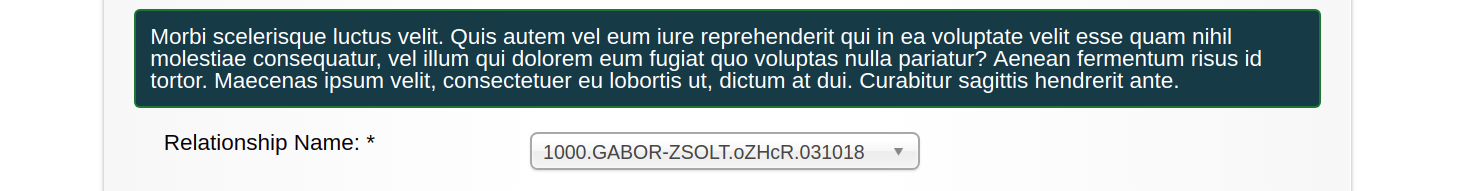
- Formatted date (e.g. ddmmyy)

- Formatted value of a field

First let’s take a look on a sample configuration:



And a sample result:



Now let’s walk through step-by step what we did here. This configuration has two important parts.

- A sub-configuration that tells the form builder what components make up the unique id

- A sub-configuration that tells the form builder which target groups it is dealing with

The generation of a unique id is fairly straightforward. You tell the form builder which field is going to be associated with the unique ID and then you define the components that will make up that ID (also the separator that is going to separate the components).

For learning more about how to configure the components take a look at the comprehensive configuration guide.

The other part of that configuration is defining the target groups that make up the form. The reason why this hasn’t been mentioned before is because the changes in thjis tutorial so far had no interaction with the already saved data.

Unique IDs on the other hand actively affect looking up already submitted data.

Therefore we need to provide a way to the form builder to determine whether we are creating a new submission or editing an already existing one.