Integrated

Form configurations

Technical Design

Jeroen van Leeuwen
19-7-2018

Introduction

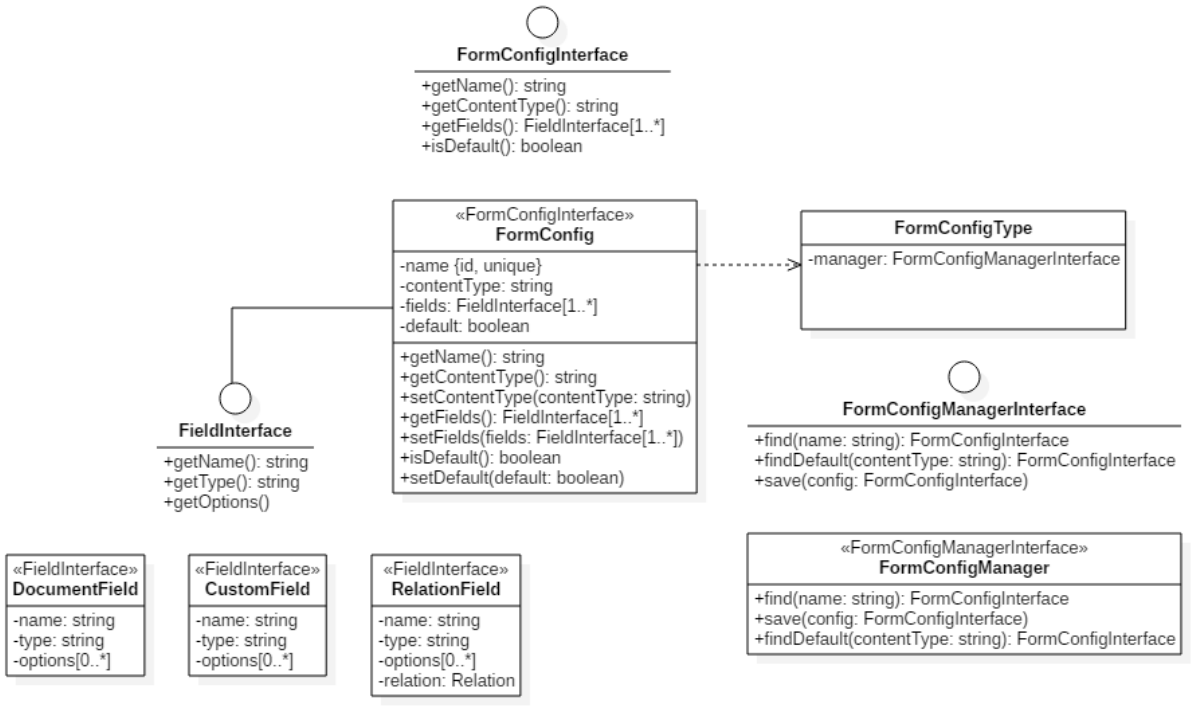
This document contains a technical design of the Integrated form configurations. With the form configurations it will be possible to make a configuration for a content type. With this configuration a form can be generated by which a content item can be saved.

Requirements

In this chapter the requirements of the form configurations will be listed.

- A content type can have one or multiple form configurations
- Per content type one form configuration can/should be marked as default
- A form configuration can contains three different fields:
    - Document type fields
    - Custom fields
    - Relation fields
- The order of the mixed fields can be changed per form configuration
- Fields can be marked as required/optional per form configuration
- A form configuration should generate a Symfony form type which can generate/populate a content item
- A form configuration should have an unique identifier preferably the name of the FormConfig.

Design



So the basic idea is to create a FormConfig document and a FormConfigType which is a Symfony form type. Based on the FormConfig document which contains the fields a form can be created with the FormConfigType.

FormConfig

The FormConfig is a Mongo document and will contain the configuration for the form. It contains an unique identifier (name for now), the content type, the fields and a default property.

So what is needed in order to create this FormConfig document:

FieldProvider

In order to create the configuration the available fields of the content type needs to be provided. This provider can be divided into different providers:

- Document fields
  The document fields are the fields that are annotated in the Content document.

- Relation fields
  The relation fields are the relation documents which have the content type selected.

- Custom fields
  The custom fields are fields that can be added to the FormConfig.

Controller

Besides a provider for fetching the available fields a controller needs to be created for providing the user interface for managing a FormConfig.

With this controller a FormConfig can be created, edited and deleted.

Also the order of the fields needs to be editable. In order to change the order the current relation field needs to be separated into separate form fields.

For saving and fetching the FormConfigManager will be used.

FormConfigType

The FormConfigType is a Symfony form type which needs a FormConfigManagerInterface for fetching the FormConfigInterface which contains the fields.

OptionResolver

The OptionResolver should validate against a valid FormConfig option. The FormConfigType can handle a FormConfig identifier or an identifier for a ContentType for the default configuration. The resolver will translate this into a FormConfig document.

Events

The current ContentFormType dispatches several events in order to expand and change the form. The FormConfigType will not dispatch any events. Symfony provides out of the box solutions for the current events: http://symfony.com/doc/3.4/form/events.html. In the backwards compatibility chapter these events will be described.

Backwards compatibility

In order to keep the current structure compatible with the new structure the following functions needs to be added:

Relations expander

When a relation is added or edited the relations will be added / removed to the FormConfig documents from that Content Type.

Please note that it can be confusing that a relation can be activated / deactivated on multiple places.

Configuration fallback

It is preferred that a FormConfig is mandatory and that a ContentType without a FormConfig cannot be edited.

Therefore no fallback will be provided but a migration / conversion script will be created.

Form event listeners

The current event listeners / subscribers should be changed in order to work with the new FormConfig.

Integrated\Bundle\CommentBundle\EventListener\CommentFormFieldsSubscriber

This class adds a comment functionality for placing a comment by a field. This will be transformed into an extension.

Integrated\Bundle\ContentBundle\Form\EventListener\FormFieldSubscriber

This class disables fields when the user has no edit access. This will be transformed into an extension.

Integrated\Bundle\ContentBundle\EventListener\ContentChannelIntegrationListener

This class adds a channel form. This will be transformed into an extension.

Integrated\Bundle\ContentBundle\EventListener\ContentRelationsIntegrationListener

This class will be removed, relations are configured within the FormConfig.

Breaks

There will also be a few backwards compatibility breaks introduced: The current ContentFormType will be removed and also the custom events will be removed. Any custom code that uses this ContentFormType or that listens to the events will not work anymore.