# - Aide a la configuration et au developpement sur Formbuilder-

Ce document a pour objectif d’indiquer comment configurer le front de l’application Formbuilder afin de créer des contextes et d’en paramétrer entièrement les formulaires.

Il est à noter que l’on utilise backbone forms dans le formbuilder pour la génération de nos inputs, et donc, les objets JSON correspondant aux inputs que l’on va retrouver dans l’application devront être dans un format compréhensible par backbone forms.

### Définition d’un « topcontext »

Le topcontext correspond au « super contexte » dans lequel s’inscrit l’application, celui-ci définit certains comportements de base du formbuilder qui sont généraux à tous les contextes que celui-ci propose.

Afin de le modifier, il est nécessaire d’éditer le fichier « app-config.js »  se trouvant dans le dossier « assets/js ».

Dans l’objet JSON « AppConfiguration » contenu dans ce fichier, modifier la valeur de la propriété « topcontext » afin de définir un nouveau topcontext.

### Création d’un nouveau contexte

Lorsque l’on parle de contexte dans le Formbuilder, on doit comprendre « contexte applicatif ».

Afin de créer un nouveau contexte, il est nécessaire d’éditer le fichier « app-config.js »  se trouvant dans le dossier « assets/js ».

Dans l’objet JSON « AppConfiguration » contenu dans ce fichier, il faudra rajouter une nouvelle propriété dans la propriété « contexts » pour créer un nouveau contexte, la valeur de cette propriété sera le nom du nouveau contexte.

Un nouveau contexte ne doit pas obligatoirement définir de propriétés, s’il est laissé vide, alors il aura les propriétés du contexte « defaults ».

On peut définir un certain nombre de propriétés spécifiques à un contexte directement depuis ce fichier de configuration :

* La liste des types d’inputs qu’il est possible d’ajouter dans un formulaire : ajout d’une propriété « inputTypes» qui sera un tableau contenant une liste d’inputs existants dans le formbuilder (la liste des inputs disponible est indiquée dans le fichier)
* La liste des propriétés (colonnes) affichées par défaut dans la grille des inputs lors de la création d’un formulaire : ajout d’une propriété « editColumns» qui sera un tableau contenant une liste de propriétés qui seront configurées dans le contexte (voir section \*\*\*)
* Un tableau de conversion entre types (non réimplémenté par RK de puis le rework), qui doit permettre de transformer un type d’input en un autre type d’input : ajout d’une propriété « allowedConvert » qui sera un tableau contenant une liste de tableaux contenant une liste d’inputs qu’il est possible de convertir de l’un vers l’autre (voir exemple dans le fichier)

La création d’un nouveau contexte suppose la création d’un certain nombre de fichiers de configuration qui vont être listés ci-après.

#### Les collections, propriétés statiques et contextuelles de formulaires

Dans le formbuilder, la liste des propriétés statiques pour un formulaire est définie dans le fichier « FieldCollection.js » se trouvant dans le dossier « assets/js/editionPageModule/collection », celui-ci n’est pas voué à évoluer, mais en cas de besoin, il contient suffisamment d’exemples qui vous permettront de rajouter du contenu.

Toujours dans ce dossier, on trouvera la liste des propriétés spécifiques à un contexte pour un formulaire, définie dans le fichier « collection » du contexte sélectionné, avec pour nom [contexte]Collection.js .

*NB : Ce fichier sera donc une « extension » de la collection de base existante « FieldCollection.js »*

Cette collection doit retourner un objet contenant les propriétés suivantes :

* schemaExtention : Cette propriété doit contenir la liste des propriétés spécifiques au formulaire que l’on désire retrouver pour ce contexte, ces propriétés spécifiques doivent être compréhensibles par le moteur backbone qui va générer des inputs à partir de celles-ci.

*NB : une propriété spécifique au Formbuilder peut être rajoutée dans un objet backbone pour un input : « searchable » qui devra être à « true » si l’on veut qu’il soit possible de filtrer dans la homepage un formulaire de ce contexte à partir de cette propriété*

* propertiesDefaultValues : Cette propriété doit contenir la liste des valeurs par défaut pour toutes les propriétés spécifiques, si une propriété spécifique ne devait avoir aucune valeur par défaut, il sera tout de même indispensable de la faire figurer dans cette propriété avec une valeur vide. Le format des propriétés pour propertiesDefaultValues sera tout simplement « [inputname] : [defaultValue] »

La collection doit également obligatoirement implémenter deux fonctions :

* getSchemaExtention : qui doit retourner « this.schemaExtention »
* initializeExtention : qui doit retourner une valeur booléenne correspondant à la réussite ou non de l’initialisation de l’objet

Pour finir, le fichier « CollectionExtention.js » référence la liste des collections existantes, il sera nécessaire de rajouter la référence à la nouvelle collection créée dans ce fichier.

#### Ajout d’inputs statiques contextuels

Il est possible de définir une liste d’inputs que l’on veut pouvoir retrouver de façon constante dans tous les formulaires que l’on aura créés pour un contexte.

*NB : Dans l’application, ces inputs n’apparaissent pas à la création d’un formulaire, ils ne seront visibles qu’après une première sauvegarde.*

Pour cela, depuis le dossier des collections, il faudra se rendre dans le sous-dossier « staticInputs » (chemin complet « assets/js/editionPageModule/collection/staticInputs »).

Dans ce dossier, créez un fichier avec la nomenclature suivante « [Context]Statics.js ».

Ce fichier doit retourner un objet proposant les propriétés suivantes :

* staticInputs : Cette propriété doit contenir une liste de propriétés qui représente la liste des inputs statiques que l’on désire retrouver pour le contexte, avec le format suivant : « [inputName] : {prop1 : [value], prop2 :[value], …} ». Comme l’exemple l’indique, **toutes** les propriétés d’inputs doivent être renseignées à cet endroit.
* compulsoryInputs : Cette propriété est un tableau qui doit à priori contenir la liste des noms d’inputs que l’on a défini dans la propriété précédente.

Ainsi que les fonctions suivantes :

* getStaticInputs : doit renvoyer la propriété « staticInputs »
* getCompulsoryInputs : doit renvoyer la propriété « compulsoryInputs »
* applyRules : doit renvoyer la propriété « staticInputs », elle implémente communément des règles permettant de filtrer la liste des statsInputs en fonction de certains critères.
* initializeStatics : doit retourner une valeur booléenne corrrespondant à la réussite ou non de l’initialisation de l’objet

Pour finir, le fichier « ContextStaticInputs.js » référence la liste des fichiers d’inputs statiques contextuels existants, il sera nécessaire de rajouter la référence au nouveau fichier d’inputs statiques créé dans ce fichier.

### Création d’un nouveau type d’input

#### Dans le formbuilder, la liste des inputs qu’il est possible de créer est définie dans le fichier « Fields.js » se trouvant dans le dossier « assets/js/editionPageModule/models».

Pour créer un nouveau type d’input, il suffit de rajouter une nouvelle propriété à notre objet « models ».

*NB : Votre nouvel input dérivera probablement de models.BaseField, sinon d’un autre input existant.*

Cette nouvelle propriété, dont le nom devra être [InputType]Field, devra implémenter 3 fonctions :

* initialize : initialise votre nouvel input
* schema : qui retournera un objet JSON contenant la liste des propriétés dynamiques de votre input
* defaults : qui retournera un objet JSON contenant la liste des valeurs par défaut des propriétés dynamiques que vous avez listés dans le retour de votre fonction « schema »

Et elle devra également implémenter 3 propriétés :

* type : nom de l’input
* i18n : nom de l’input
* section : j’avoue là je ne sais pas trop … les autres inputs mettent un peu n’importe quoi, peut-être que ce n’est pas très important …

*EDIT : La partie « section » est peut-être là pour pouvoir spécifier le « contexte global » dans lequel s’inscrivent les inputs (pour l’instant, il n’existe que « RENECO » comme top context), et si c’est effectivement le cas, la feature est à priori non fonctionelle pour le moment.*

#### Propriétés contextuelles d’inputs

Afin de rajouter des propriétés contextuelles à un input, ouvrir dans le dossier « assets/js/editionPageModule/models/ExtraContextProperties » le fichier correspondant à votre contexte, son nom sera [Context]Properties.js .

Dans ce fichier, rajoutez (si il n’existe pas déjà) dans la propriété « extraProperties » de l’objet JSON retourné, une propriété au nom de votre type input. Cette nouvelle propriété renseignera 2 sous-propriétés :

* schema : qui correspond à la liste des propriétés dynamiques d’input spécifique à votre contexte
* defaults : qui correspond à la liste des valeurs par défaut des propriétés que vous avez listés dans la sous-propriété « schema »

*NB : Si vous désirez rajouter une propriété dynamique contextuelle pour TOUS les types d’inputs de votre contexte, au lieu de rajouter une propriété du nom de votre type d’input, nommez votre propriété « all ».*

#### Chargement de données pour les propriétés contextuelles d’inputs

Si vous voulez charger des données externes au formbuilder pour, par exemple, remplir un SELECT, c’est possible. Pour ce faire, ouvrez le fichier correspondant à votre contexte dans le dossier « assets/js/editionPageModule/views/loaders ».

Dans ce fichier, vous trouverez une fonction « loadFormData » qui est appelée suite à chaque création d’input, à partir de celle-ci, vous pouvez mettre en place du code pour par exemple appeler une API externe et récupérer des données qui seront utiles à votre input.