Gestion des formulaires

https://code-industry.net/masterpdfeditor-help/digital_signatures/

Esup-signature propose de mettre rapidement en ligne des formulaires PDF simples. Cette fonction s'appuie sur les PDF Forms (formulaires présents dans les fichiers PDF).

D'une manière générale, esup-signature est capable d'analyser les formulaires PDF, d'en effectuer le rendu (via PDF.js) et de "fusionner" les données saisies lors de la signature. Pour aller un peu plus loin vous avez la possibilité d'utiliser un PDF Form comme modèle au moment de démarrer un circuit. Dans ce cas on utilise le module d'administration pour déclarer un **formulaire**. Les formulaires sont accompagnés de plusieurs fonctions additionnelles :

- stockage des données saisies dans la base de données d'esup-signature
- export des données aux formats csv et json
- pré-remplissage des champs en fonction de l'utilisateur courant et des sources de données
- verrouillage/libération des champs en fonction de l'étape courante

La mise en place du formulaire basé sur un PDF se fait en 3 étapes :

- création du circuit des signatures (voir Gestion des circuits)
- création du PDF avec formulaire
- import du PDF avec formulaire dans esup-signature

Création d'un PDF Form

Pour créer un formulaire et générer le fichier PDF, il est possible d'utiliser Libre Office. Il gère correctement la création de champs de formulaires (texte, cases à cocher,...) ainsi que la conversion au format PDF. La seule restriction concerne les champs signature. Pour palier au problème, l'université de Rouen utilise Master PDF Editor (payant), disponible pour Windows, Linux et Mac.

Ajout d'un champ signature avec Master PDF Editor : https://code-industry.net/masterpdfeditor-help/digital_signatures/

L'opération consiste donc à placer les champs de formulaire puis à les configurer de la bonne façon.

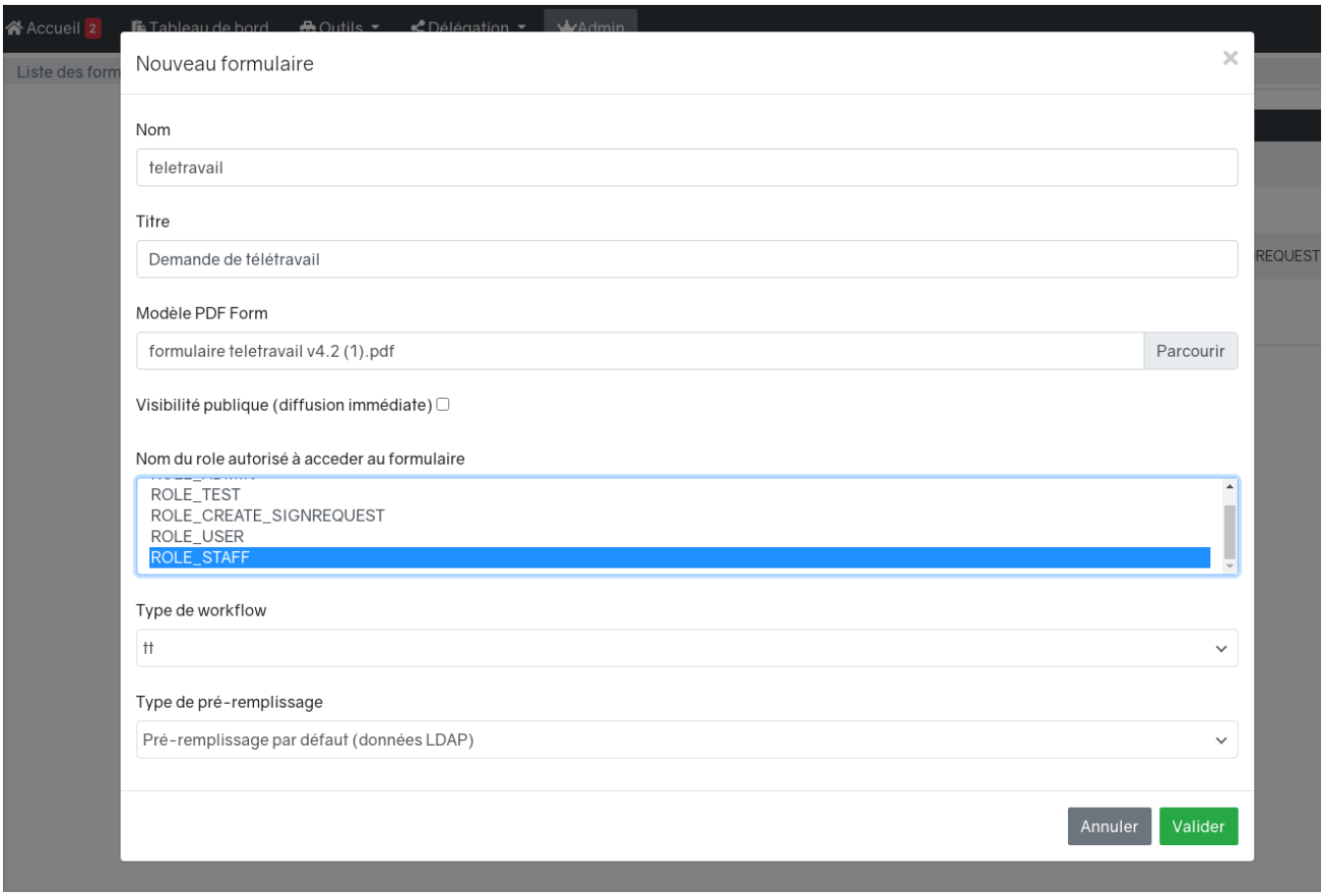
Les champs doivent être configurés comme suit:

- Le nom (esup-signature utilisera ce nom comme identifiant du champ)
- Cocher ou non "Obligatoire"
- Esup-signature gère les champs texte, texte multi-lignes, cases à cocher, radio boutons et liste déroulantes
- Dans la partie "Format" des champs texte, esup-signature reconnait les formats : Nombre et Date/Heure

- **Toutes autres options n'auront aucun effet sur le comportement d' esup-signature**

Import du formulaire

Lorsque le formulaire PDF est terminé, il faut l'importer dans esup-signature. Pour cela il faut aller sur "Admin" puis "Formulaire" puis cliquer sur le bouton bleu "+" et enfin sur l'icône PDF.

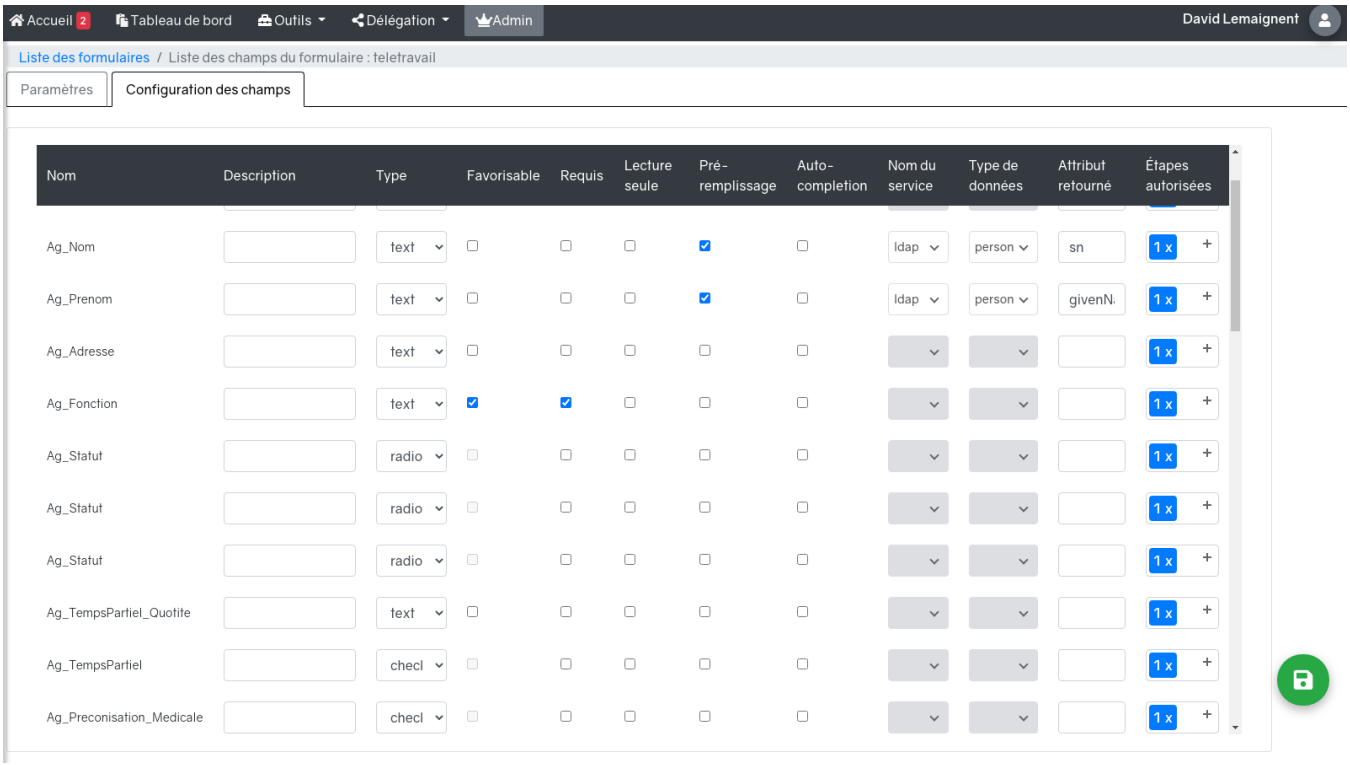


Vous devez saisir les informations suivantes :

1. un nom (technique)
2. un titre (affiché sur le bouton de création d'un nouveau document)
3. sélectionner votre modèle PDF Form
4. choisir la visibilité pour tous ou saisir les rôles pour définir qui peut accéder à ce formulaire
5. choisir le type de pré-remplissage
6. choisir un circuit qui aura été créé au préalable

Lorsque vous validez le formulaire, esup-signature analyse le PDF Form et constitue la structure du formulaire. Vous aurez ensuite accès à la configuration des champs.

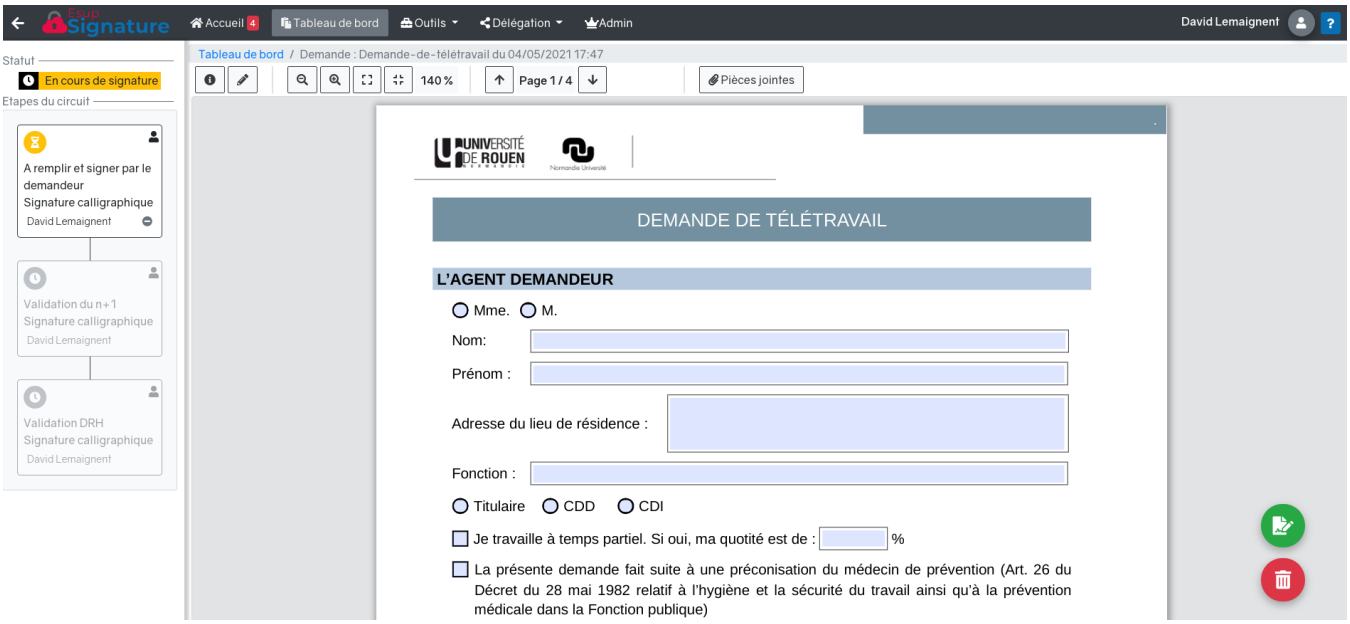
Voici un exemple de l'édition des champs d'un formulaire dans l'interface administrateur d'esup-signature ("Admin" "Formulaire" "Modifier les champs" :



On voit ici les propriétés modifiables des champs, elle sont les suivantes :

Propriété	Fonction
Nom du champ	(non modifiable)
Description	apparaît au survol et lors du contrôle des champs requis
Type	text, checkbox, radio, date, time, number, select
Favorisable	si cette case est cochée, les données saisies dans ce champ seront proposées à l'utilisateur lors de la prochaine saisis
Requis	aux étapes concernées, les champs seront obligatoires
Lecture seule	verrouille le champ
Pré-remplissage	active la fonction de pré-remplissage du champ en fonction de l'utilisateur courant
Auto-complétion	active la fonction d'auto-complétion du champ en fonction de la saisie courante dans le champ concerné
Nom du service	choix du service utilisé lors du pré-remplissage ou de l'auto-complétion (esup-signature propose default et ldap nativement voir ci-après)
Type de donnée	sélection du "sous-service" utilisé lors du pré-remplissage ou de l'auto-complétion
Attribut retourné	nom de l'attribut retourné lors du pré-remplissage ou de l'auto-complétion
Étapes autorisées	numéro des étapes pour lesquelles les champs sont à remplir (le champ sera verrouillé à toutes les autres étapes)

Exemple de rendu d'un formulaire :



Pré-remplissage et auto-complétion

Comme vu précédemment, esup-signature permet le pré-remplissage des champs. Les services disponibles sont définis à l'aide de classes de type "ExtValue" (valeur externes)

Esup-signature est fourni avec les classes DefaulExtValue et LdapExtValue. La classe DefaultExtValue est utilisable en mettant "default" dans le nom du service et "system" au niveau du type. Elle propose les attributs (calculés) suivant :

nom de l'attribut	description
day	numéro du jour
month	numéro du mois
year	numéro du l'année
date	date du jour
time	heure
dateTime	date et heure
currentUser	nom prénom de l'utilisateur courant
stepUsers	liste des mails des participants à l'étape courante
currentStepNumber	numéro de l'étape courante
id	id de la demande de signature

Vous pouvez créer vos propres classes de données externes en implémentant le type "ExtValue" en reprenant DefaultExtValue.java par exemple.

Classe de pré-remplissage

Pour pré-remplir ou auto-compléter un formulaire, esup-signature se base une une classe de "pré-remplissage" du type "PreFill". D'origine esup-signature est fourni avec la classe DefaultPreFill qui prend en charge les données externes de type DefaultExtValue et LdapExtValue. Ceci répond à une grande partie des besoins.

Cependant, tout comme cela peut être le cas pour les workflows, il se peut qu'il soit nécessaire de calculer certaines données à pré-remplir spécifiquement pour un formulaire (donnée calculée en fonction de l'utilisateur courant par exemple). Dans ce cas il y a la possibilité d'implémenter votre propre classe de type PreFill (voir DefaultPreFill.java pour l'exemple).

Voici un cas pratique se basant sur une base de donnée mariaDb :

1. Il faut configurer une connexion au niveau du fichier de configuration. Ici pour une base de données nommée "tot" on ajouter un "totoDataSource". Ne pas utiliser le nom "userListDataSource" pour la source de données car il est réserver pour un autre service.

```
extdb:
    datasources:
        totoDataSource:
            driver-class-name: org.mariadb.jdbc.Driver
            password: ********
            url: jdbc:mariadb://serveur.univ-ville.fr:3306/toto
            username: consult
            name: toto
```

2. ajouter une classe TotoExtValue dans src/main/java/org/esupportail/esupsignature/service/interfaces/extvalue/impl/ . Voici un exemple :

```
package org.esupportail.esupsignature.service.interfaces.extvalue.impl;

import org.esupportail.esupsignature.entity.SignRequest;
import org.esupportail.esupsignature.entity.User;
import org.esupportail.esupsignature.service.extdb.ExtDbService;
import org.esupportail.esupsignature.service.interfaces.extvalue.ExtValue;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;
import javax.annotation.Resource;
import java.sql.ResultSet;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Component
@ConditionalOnProperty(name = "extdb.datasources.totoDataSource.name")
public class TotoExtValue implements ExtValue {

   private JdbcTemplate jdbcTemplate;

   @Resource
   private ExtDbService extDbService;

   @PostConstruct
   public void initJdbcTemplate() {
      this.jdbcTemplate = extDbService.getJdbcTemplateByName("totoDataSource");
   }

   @Override
   public String getValueByName(String name, User user, SignRequest signRequest) {
      return initValues(user, signRequest).get(name).toString();
   }

   @Override
   public List<Map<String, Object>> search(String searchType, String searchString, String searchReturn) {
      return null;
   }

   @Override
   public String getName() {
      return "toto";
   }

   @Override
   public Map<String, Object> initValues(User user, SignRequest signRequest) {
      Map<String, Object> values = new HashMap<>();
      jdbcTemplate.query("select nom_user, prenom_user from user where eppn='" + user.getEppn() + "'",
(ResultSet rs) -> {
         values.put("nom", rs.getString("nom_user"));
         values.put("prenom", rs.getString("prenom_user"));
      });
      return values;
   }

}
```

Dans ce cas précis, on suppose que que l'on souhaite récupérer les nom / prénom du l'utilisateur courant dans notre base mariaDB. On affecte les valeurs récupérées dans la une map d'attributs (on y nomme les attributs comme on le souhaite).

Les attributs disponibles au niveau de la méthode initValues sont :

- User user : l'utilisateur courant avec en particulier l'EPPN dans getEppn() et l'email dans .getEmail()
- SignRequest signRequest : le demande de siagnture courante (voir https://github.com/EsupPortail/esup-signature/blob/master/src/main/java/org/esupportail/esupsignature/entity/SignRequest.java)

3. ajouter une classe TotoPreFill dans src/main/java/org/esupportail/esupsignature/service/interfaces/prefill/impl/ . Il est possible de repartir de la classe https://github.com/EsupPortail/esup-signature/blob/master/src/main/java/org/esupportail/esupsignature/service/interfaces/prefill/impl/TestPreFill.java en la modifiant comme suit :

```
@Override
public Map<String, List<String>> getTypes() {
   Map<String, List<String>> types = new HashMap<>();
   types.put("ldap", Arrays.asList("person"));
   types.put("default", Arrays.asList("system"));
   types.put("toto", Arrays.asList
("attributs"));                                                                    #Ajout
d'un nom de type représentant le type de données
   return types;
}


.............


public List<Field> preFillFields(List<Field> fields, User user, SignRequest signRequest) {
   List<Field> filledFields = new ArrayList<>();
   PDFont font = PDType1Font.HELVETICA;
   PDResources resources = new PDResources();
   resources.put(COSName.getPDFName("Helvetica"), font);
   ExtValue extDefaultValue = extValueService.getExtValueServiceByName("default");
   Map<String, Object> defaultValues = extDefaultValue.initValues(user, signRequest);
   ExtValue extTotoValue = extValueService.getExtValueServiceByName("toto");
#déclaration du nouveau extValue
   Map<String, Object> totoValues = extTotoValue.initValues(user, signRequest);
#initialisation des valeurs
   ExtValue extLdapValue = extValueService.getExtValueServiceByName("ldap");
   Map<String, Object> ldapValues = extLdapValue.initValues(user, signRequest);
   for(Field field : fields) {
      if(field.getExtValueServiceName() != null && !field.getExtValueServiceName().isEmpty()) {
         if(field.getExtValueServiceName().equals("ldap")) {
            String extValueName = field.getExtValueReturn();
            if(ldapValues.containsKey(extValueName)) {
               if(extValueName.equals("schacDateOfBirth")) {
                  field.setDefaultValue(extLdapValue.getValueByName("schacDateOfBirth", user,
signRequest));
               } else {
                  field.setDefaultValue((String) ldapValues.get(extValueName));
               }
            }
         } else if(field.getExtValueServiceName().equals("default")) {
            String extValueName = field.getExtValueReturn();
            if(defaultValues.containsKey(extValueName)) {
               field.setDefaultValue((String) defaultValues.get(extValueName));
            }
            if(field.getExtValueReturn().equals("covid(duree)")) {
               field.setDefaultValue("1");
            }
         } else if(field.getExtValueServiceName().equals("toto"))
{                                          #gestion des champs pré-remplis par toto
            String extValueName = field.getExtValueReturn();
            if(totoValues.containsKey(extValueName)) {
               field.setDefaultValue((String) totoValues.get(extValueName));
            }
         }
      }
      filledFields.add(field);
   }
   return filledFields;
}
```

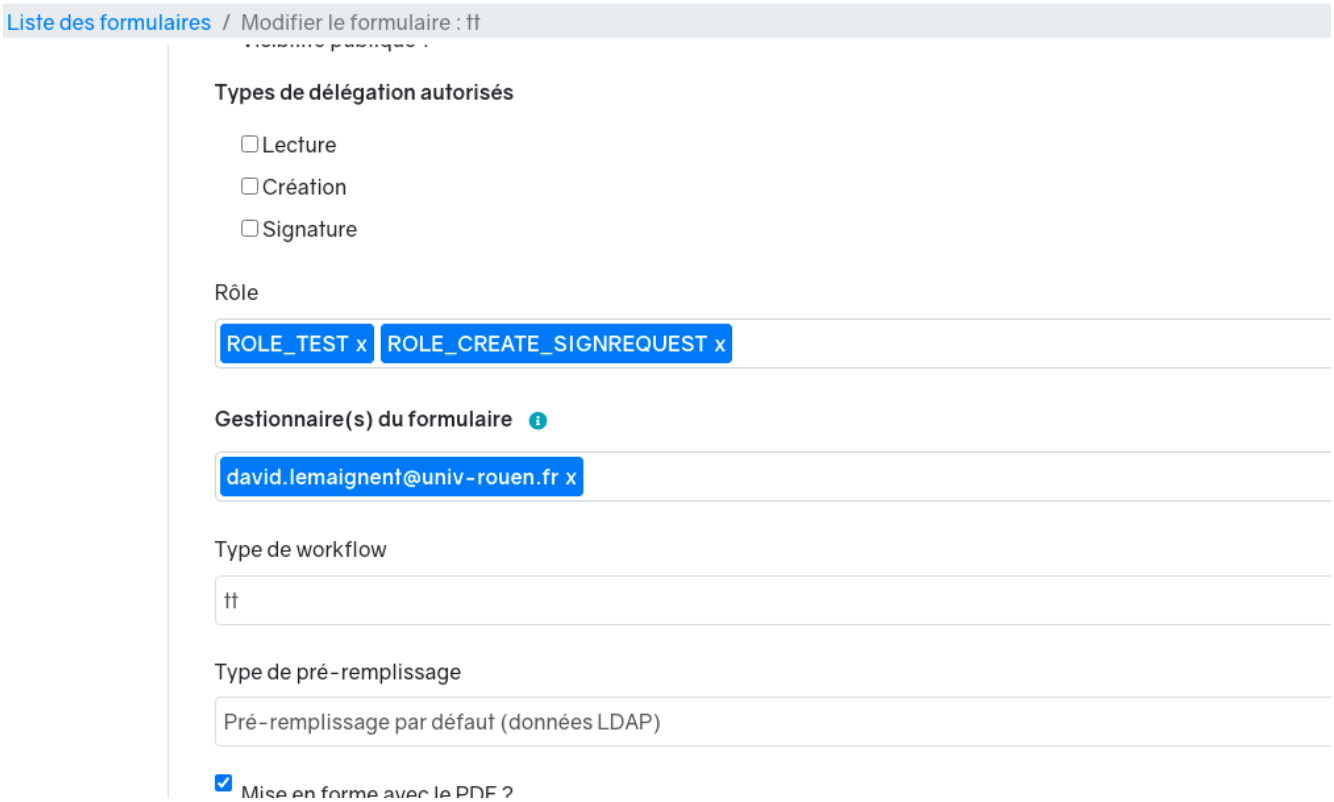
On y ajoute le nouveau service ExtValue "toto" en respectant la syntaxe. A ce niveau il est en possible de faire des calculs ou de mettre des données en dur avant l'affectation de la valeur aux champs du formulaire. A noter que l'on pourrait imaginer qu'un ExtValue puisse retourner différents types de données comme c'est le cas avec LdapExtValue (person, organizationalUnit). Dans ce cas on dispatche arbitrairement les données au moment du pré-remplissage.

4. Après recompilation / redémarrage et redéploiement, le service de pré-remplissage "toto" doit être disponible. On choisi donc le service puis le type de données (dans notre cas, "toto" puis "attributs") et on saisie manuellement le nom de l'attribut "nom" ou prenom..

Il est tout à fait possible de se passer des classes ExtValue en nommant explicitement des noms d'attributs dans la classe Prefill. De plus rien empêche de faire appel à des web service plutôt qu'à une base de données. Dans ce cas on utilisera restTemplate fourni pas spring pour récupérer les informations : https://www.baeldung.com/rest-template

Gestionnaire du formulaire

Il est possible de définir un ou plusieurs gestionnaires au niveau des paramètres du formulaire.



Les personnes désignées auront accès en lecture à tous les documents correspondant a ce formulaire. De plus il leur sera possible de télécharger toutes les données saisies au format CSV.

L'accès à ces données se fait par le menu "Outils" "Espace gestionnaire"

Actions Javascript

Les actions qui peuvent être incorporées dans le PDF Form ne sont pas prises en compte lors de l'import et du rendu du formulaire. Cependant, pour rendre le formulaire plus dynamique, il est possible de coder des comportements à l'aide du champ "Action". Dans la mesure du possible il est préférable de ne pas avoir à utiliser cette fonction. En général cela peut dépanner pour verrouiller/déverrouiller des champs en fonction d'autres champs.

Le fonctionnement est simple, vous devrez identifier les ID des éléments prenants part au dynamisme, à l'aide de l'inspecteur de code de votre navigateur. Ensuite pourrez écrire du code, en Javascript et/ou en jQuery, en vous référant aux identifiants des éléments du formulaire. Aucune fonction spécifique à esup-signature n'est présente, il s'agit de javascript totalement standard.

Le code sera exécuté à chaque rendu d'une page du document, c'est à dire à chaque zoom ou changement de page. Il est donc conseiller de systématiquement supprimer les listenner avant de les créer/recréer en utilisant : removeEventListener() en js ou unbind() en jQuery