Documentation ID-si

Version 8

ANAP

15 June 2016

Table des matières

CHAPITRE 1

Licence

Copyright ANAP, (2016)

Le logiciel documenté ici est un programme informatique servant à réaliser un inventaire des systèmes d'information hospitaliers..

Ce logiciel et sa documentation sont régis par la licence CeCILL soumise au droit français et respectant les principes de diffusion des logiciels libres. Vous pouvez utiliser, modifier et/ou redistribuer ce programme et sa documentation sous les conditions de la licence CeCILL telle que diffusée par le CEA, le CNRS et l'INRIA sur le site "http://www.cecill.info".

En contrepartie de l'accessibilité au code source et des droits de copie, de modification et de redistribution accordés par cette licence, il n'est offert aux utilisateurs qu'une garantie limitée. Pour les mêmes raisons, seule une responsabilité restreinte pèse sur l'auteur du programme, le titulaire des droits patrimoniaux et les concédants successifs.

A cet égard l'attention de l'utilisateur est attirée sur les risques associés au chargement, à l'utilisation, à la modification et/ou au développement et à la reproduction du logiciel par l'utilisateur étant donné sa spécificité de logiciel libre, qui peut le rendre complexe à manipuler et qui le réserve donc à des développeurs et des professionnels avertis possédant des connaissances informatiques approfondies. Les utilisateurs sont donc invités à charger et tester l'adéquation du logiciel à leurs besoins dans des conditions permettant d'assurer la sécurité de leurs systèmes et ou de leurs données et, plus généralement, à l'utiliser et l'exploiter dans les mêmes conditions de sécurité.

Le fait que vous puissiez accéder à cet en-tête signifie que vous avez pris connaissance de la licence CeCILL, et que vous en avez accepté les termes.

CHAPITRE 2

Introduction

Cette documentation technique fournit les informations nécessaires pour à la fois comprendre l'architecture technique retenue pour l'application ID-si et les détails d'implémentation faits au cours du développement. Elle s'attache par ailleurs à décrire aussi bien le back-end (fondé sur Django) et le front-end.

Elle comporte les chapitres suivants :

Mettre en place l'environnement de développement Ce chapitre comprend les instructions à suivre pour créer l'environnement de développement qui permet de tester et faire évoluer l'application.

L'architecture de l'application Ce chapitre présente l'architecture technique retenue pour l'application ANAP ID-si et comporte une liste de toutes les dépendances techniques utilisées lors du développement.

La configuration Cette section expose les paramètres de configuration spécifiques qui ont été utilisés pour paramètrer les différents composants et packager l'application.

Les modèles de données Ce chapitre comporte la documentation des modèles utilisées pour persister les données saisies par les utilisateurs, ainsi que le schéma de base de données qui en résulte.

Les classes de formulaires Ce court chapitre documente les éléments utilisés pour faciliter l'affichage et la validation des formulaires affichés par ID-si.

Les routes Toutes les routes mises en place pour l'application sont ici détaillées.

Les vues Ce chapitre détaille le comportement des différentes vues de l'application et des templates qui leur sont directement associés.

Les templates utilitaires Ce chapitre présente les templates utilitaires utilisés de façon générique dans diverses sections de l'application.

Les graphes de restitution Ce chapitre présente les composants logiciels mis en place pour dessiner les graphes, ainsi que les éléments de configuration utilisés pour afficher les différents graphes concrets.

Le front-end Ce chapitre détaille les développements javascript supplémentaires réalisés dans le cadre de la conception du front-end.

L'API REST Ce chapitre documente l'API REST mis en place pour déployer certaines des fonctionnalités dynamiques dans les formulaires.

CHAPITRE 3

Mettre en place l'environnement de développement

3.1 Installer les dépendances de l'outil

Le fichier `conda-environment.yml` contient une description des dépendances logicielles de l'application ID-si, et liste notamment :

- la version de l'interpréteur Python à utiliser
- les versions de chacune des dépendances, afin d'éviter que l'évolution d'une des dépendances ne puisse introduire des problèmes dans ID-si.

Cette démarche suppose que l'utilitaire **conda** (installé par défault avec la distribution Anaconda) est bien installé.

En tout premier lieu, il faut ouvrir la ligne de commande et se placer dans le dossier contenant les sources comme répertoire de travail (en utilisant la commande **cd**).

Les commandes suivantes permettent ensuite de créer un environnement Python nommé *mesis* :

```
set CONDA_FORCE_32BIT=1
conda env create -f conda-environment.yml
```

La première commande sert à indiquer que l'environnement créé doit utiliser une version 32 bits de Python, ce qui permettra d'exporter une version à même de fonctionner sur toutes les versions de Windows, 32 bits comme 64 bits.

3.2 Activer l'environnement créé

Les commandes suivantes permettent d'activer l'environnement :

```
set CONDA_FORCE_32BIT=1
set DJANGO_SETTINGS_MODULE=anap.settings.local
activate mesis
```

Ici, la seconde ligne sert à indiquer que c'est la configuration de développement et de test (`anap.settings.local`) qui devra être utilisé par Django (se réfèrer à la page La configuration pour plus de détail sur les différentes configurations disponibles).

A partir de maintenant, l'application s'executera dans cet environnement spécifique, comme en atteste la présence de la mention `[mesis]` en tête du prompt.

Ces commandes ne sont valables que pour la session en cours. Il faut les exécuter **à chaque nouvelle session**, c'est-à-dire à chaque nouvelle ouverture d'une fenêtre **cmd**.

3.3 Le fichier `manage.py`

Le fichier `manage.py`, automatiquement créé par Django lors de la création d'une application, est un utilitaire en ligne de commande utilisé pour réaliser toutes les tâches administrative liées à Django, comme la mise à jour des bases de données, le lancement d'un serveur de test, ou la création d'une nouvelle sous-application.

Pour toutes autres utilisations que celles détaillées ci-dessous, se référer à la documentation relative à manage.py pour la version 1.8 de Django.

3.4 Créer la base de données de développement

En développement, une base de données SQLite3 spécifique est utilisée pour ne pas toucher à la version de production qui sera intégrée dans le packaging.

La commande suivante permet de la créer :

```
python manage.py migrate
```

Le fichier `db.sqlite3` devrait avoir été créé dans le répertoire.

3.5 Lancer le serveur web de test

Pour lancer un serveur de développement capable de recharger automatiquement les différents modules, executez la commande suivante :

```
python manage.py runserver
```

Par défaut, le serveur est lancé à l'adresse http://127.0.0.1:8000

3.6 Mettre à jour les bases de données

En cas de changement des modèles de données qui induiraient des modifications sur la structure des bases de données, il peut être nécessaire d'en mettre à jour la structure.

Pour le vérifier, executer la commande suivante :

```
python manage.py makemigrations
```

Si la commande détecte des changements, elle produit une description des changements à appliquer à la base de données dans des fichiers appelés *migrations*.

La commande suivante permet de les appliquer à la base de développement :

```
python manage.py migrate
```

En vue du packaging, ils doivent aussi être appliqués à la base de production :

```
cmd /C "set DJANGO_SETTINGS_MODULE=anap.settings.production&& python manage.py migrate"
```

3.7 Extraire cette documentation

La commande suivante permet de générer la documentation au format HTML

```
cmd /C "cd doc && make html"
```

Dans certains cas, la table des matières peut ne pas s'être mise à jour sur toutes les pages. Dans ce cas, executez la commande suivante avant de retenter celle ci-dessus

```
cmd /C "cd doc && make clean"
```

3.8 Réaliser l'empaquetage de l'application

Les deux commandes suivantes permet de réaliser l'empaquetage de l'application

```
cmd /C "set DJANGO_SETTINGS_MODULES=anap.settings.production&& python manage.py collectstatic"
pyinstaller packaging\package.spec
```

La première permet de repérer d'éventuelles modifications des fichiers statiques. La seconde réalise le paquet lui-même.

L'application est exportée comme un dossier appelé `mesis2` dans le répertoire `dist`.

CHAPITRE 4

L'architecture de l'application

4.1 ID-si : un projet Django

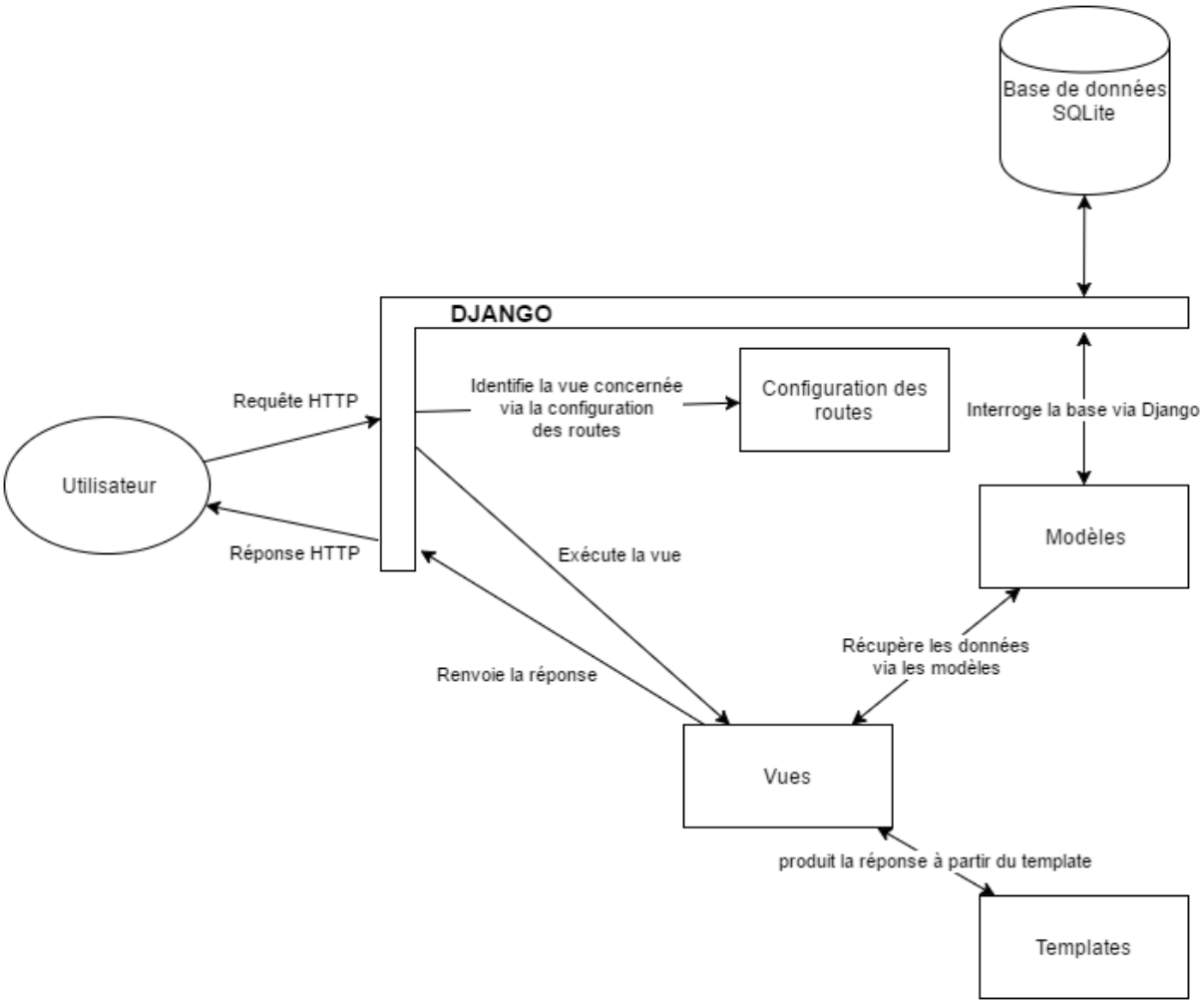
Django est un framework Python facilitant le développement d'applications web. Il propose une approche modulaire, fondée sur un modèle de conception MTV (Modèle-Template-View).

Conceptuellement, Django agit comme une interface entre le code de l'application et le monde extérieur. En langage technique, on parle d'une *inversion de contrôle* : ce n'est plus le code de l'application qui est responsable d'appeler Django, c'est Django qui prend en main le déroulé de l'execution et qui appelle le code de l'application aux moments pertinents.

Les avantages sont notamment les suivants :

- C'est Django qui se charge de l'initialisation de tous les composants nécessaires pour le bon fonctionnement de l'application
- Django prend en charge toute la complexité liée aux interactions avec des acteurs extérieures : par exemple, il prend en charge toute l'interaction directe avec le navigateur internet du client, en particulier tous les aspects liés au protocole HTTP, pour ne fournir à l'application qu'une vision très abstraite de la transaction et tout à fait adaptée aux besoins d'une application web. De la même façon, l'utilisation des modèles pour l'accès aux données à sauvegarder permet de s'abstraire de l'utilisation d'un modèle de base de données particulier

Le schéma ci-dessous propose une vue générale de l'architecture d'une application Django :



4.1.1 Une approche modulaire

Les projets Django sont généralement divisées en plusieurs applications. ID-si reprend cette architecture générale et comprend les composants logiciels suivants :

- Un premier *package*, `anap` sert de racine au projet et comporte le code nécessaire pour le faire tourner, notamment le fichier principal de routage (`anap.urls`), le fichier de configuration du serveur et de la base de données SQLite (`anap.settings`) et le module pour faire tourner le serveur (`anap.wsgi`). Ces fichiers sont décrits brièvement dans la section La configuration.
- `mesis` comprend l'essentiel du code de l'application. Il est décrit plus en détail ci-dessous et dans le reste de cette documentation.
- `api` comprend du code complémentaire pour l'interfaçage entre le code Python utilisé en back-end et le code javascript utilisé en front-end

En supplément, deux dossiers supplémentaires comprennent des fichiers annexes :

- Le dossier `doc` comporte les fichiers utilisés pour construire cette documentation technique.
- Le dossier `static` comporte les dépendances javascript et css nécessaires pour la partie front-end de l'application.

4.1.2 Un modèle de conception MTV

Au sein de chaque composant logiciel, Python encourage l'utilisation d'un modèle de conception Modèle/Template/Vue pour organiser le code source :

- les *modèles* représentent les données de l'application. Ils permettent d'organiser facilement la persistence des données.

— La logique de sélection et d'interaction de l'utilisateur avec les données est décrite par un ensemble de *vues*
— La façon dont ces données sont ensuite présentées à l'utilisateur est prise en charge par les *templates*.

Les deux applications `mesis` et `api` sont organisées selon ces principes.

4.2 L'application `mesis`

Le package `mesis` concentre l'essentiel du code et est structurée comme une application Django. Il utilise donc les fichiers standards suivants :

— `mesis.urls` configure les routes de l'application. Il s'agit des instructions qui permettent à Django d'associer les comportements aux *URLs* accédées par le navigateur de l'utilisateur. Le contenu de ce module est détaillé dans le chapitre Les routes
— *`mesis.models`* comporte la définition des modèles de l'application. Ces classes décrivent les données qui doivent être sauvegardées par l'application et les comportements qui leurs sont associés. Voir le chapitre Les modèles de données.
— *`mesis.views`* définit les vues de l'application. C'est ici que va être défini l'essentiel de la logique métier de l'application. Le chapitre Les vues en décrit le fonctionnement.
— *`mesis.forms`* comporte du code spécifique pour définir les objets de formulaires utilisés pour faciliter la gestion des formulaires à afficher et à valider dans les vues, tels que détaillés dans le chapitre Les classes de formulaires
— *`mesis.rendus`* comporte les moteurs de rendu utilisés pour générer les éléments logiciles pour afficher les graphes dans le navigateur
— *`mesis.graphs`* comporte le code définissant les graphes eux-mêmes dans leur dimension métier : ils délèguent ensuite aux objets du module *`mesis.rendus`* le soin de générer les composants logiciels pour l'affichage.
— *`mesis.barometres`* comporte le code pour générer les baromètres généraux. Ce module utilise aussi les composants de *`mesis.rendus`* pour afficher les graphes.
— Le dossier `templates` contient les templates utilisés par l'application ID-si pour l'affichage des vues, ainsi que quelques templates utilitaires utilisées pour encapsuler certaines fonctionnalités récurrentes. Les templates associés aux vues sont décrites dans le chapitre Les vues, et les templates spécifiques dans le chapitre Les templates utilitaires
— Le dossier `templatetags` comporte quelques tags et filtres spécifiques utilisés pour faciliter le développement des templates. Ils sont aussi documentés dans le chapitre Les templates utilitaires
— Le dossier `static` comporte des éléments statiques spécifiques à l'application ID-si.

Le package comprend aussi un autre composant non standard, spécifique à ID-si :

— *`mesis.nomenclatures`* contient, sous une forme directement exploitable en Python, les nomenclatures utilisées dans le cadre de l'application, notamment :
  — Le référentiel des compétences en systèmes d'information d'octobre 2013 (*`mesis.nomenclatures.competences`*)
  — Le référentiel des infrastructures pour le formulaire infrastructure (`mesis.nomenclatures.infrastructure`)
  — Le référentiel des applicatifs pour le formulaire applicatifs (*`mesis.nomenclatures.applicatifs`*
  — Le référentiel des référentiels applicatifs (*`mesis.nomenclatures.referentiels`*)

4.3 L'application `api`

Cette application fournit une API REST pour afficher et modifier les modèles de données de l'application `mesis`. Elle est notamment utilisée pour les formulaires dynamiques pour la sélection des liens entre contrats et composants/applicatifs.

Son fonctionnement est détaillé dans le chapitre L'API REST.

4.4 L'empaquetage

Cette application est distribuée sous la forme d'un paquet logiciel executable sans avoir à installer quoi que ce soit.

Pour permettre cet empaquetage, le dossier `packaging` contient plusieurs éléments, documentés dans le chapitre Les éléments pour l'empaquetage.

4.5 Dépendances logicielles

ID-si utilise les dépendances logicielles suivantes.

4.5.1 Pour le back-end

- *Python* 3.4.3, sous licence PSF, sans copyleft
- *Django* 1.8.4, licence BSD à 3 clauses
- *Django Rest Framework* 3.3.1, licence BSD à 2 clauses
- *django-widget-tweaks* 1.4.1, licence MIT
- *waitress** 0.8.10, Zope Public License (ZPL) Version 2.1

4.5.2 Pour le front-end

- *Boostrap* 3.3.5 (licence MIT) et ses dépendances :
  - *html5shiv* 3.7.3 (licence MIT)
  - *respond.js* 1.4.2 (licence MIT)
- *jQuery* 1.11.3 (licence MIT)
- *Backbone* 1.2.3 (licence MIT)
- *underscore* 1.8.3 (licence MIT)
- *json2* (dernière version, domaine public)
- *d3.js* (licence BSD à 3 clauses) et *c3.js* (licence MIT) pour l'affichage des graphes.

4.5.3 Pendant le développement

- *Sphinx* 1.3.1 (licence BSD à 2 clauses) pour la génération de cette documentation

CHAPITRE 5

La configuration

Ce fichier décrit les configurations respectives utilisées pour produire l'environnement de développement, pour Django et pour le serveur web qui gère les requêtes en entrée de Django.

5.1 L'environnement conda de développement

L'ensemble des paquets utilisées pour l'environnement de développement sont décrits dans le fichier `conda-environment.yml`. Cela inclut aussi toutes les bibliothèques logicielles inclues dans l'application elle-même.

Lorsque la distribution Anaconda (ou Miniconda) a été installée, la commande suivante permet de recréer l'environnement de développement.

5.2 La configuration de Django

Le paquet `anap.settings` comprend les différents profils de configurations de l'application ID-si :

- `anap.settings.base` comprend les paramètres de configuration de base, communs à tous les profils de configurations. Il fixe notamment les éléments suivants :
  - Les bibliothèques Django à importer : *rest_framework* et *widget_tweaks*
  - Les sous-composants Django à utiliser
  - La langue et le fuseau horaire pour la France
  - Le chemin auquel seront disponibles les fichiers statiques (/static/)
- `anap.settings.local` est le profil de configuration utilisé pour le développement et les tests :
  - Il utilise la base de données SQLite locale `db.sqlite3` pour sauvegarder les informations.
  - Le mode DEBUG est activé.
- `anap.settings.heroku` est utilisé pour héberger l'application sur Heroku pour la version bac-à-sable en ligne :
  - La base de données PostgreSQL d'Heroku est utilisée
  - Le mode DEBUG est activé
  - Les fichiers statiques seront collectés et servis depuis un dossier "staticfiles"
- `anap.settings.production` est utilisé dans la version empaquetée pour l'utilisation :
  - Il utilise la base de données SQLite : file :*production.sqlite3* qui doit donc être laissée vide (mais il ne faut pas oublier de migrer les tables si on modifie les modèles)
  - Pour la phase de test avec les établissements, DEBUG est activé, mais sera desactivé pour la version mise en production.
  - Les fichiers statiques seront collectés et servis depuis un dossier "staticfiles"

5.3 La configuration du serveur HTTP

Hors tests en local, l'application Django tourne derrière les éléments suivants :

— Le serveur HTTP `waitress` sur le port 8000. Sur Heroku, il est lancé via le fichier `Procfile`. Dans l'environnement de production, c'est le fichier `packaging.start_server` qui crée et lance le serveur.
— Un *middleware* chargé de servir les fichiers statiques, `dj_static`. Il est configuré dans le fichier `anap.wsgi`.

CHAPITRE 6

Les nomenclatures

Ce fichier décrit le module *mesis.nomenclatures* qui regroupe l'ensemble des nomenclatures référencées sous une forme directement exploitable par l'application.

mesis.nomenclatures.**MOIS = ['jan', 'fev', 'mar', 'avr', 'mai', 'juin', 'juil', 'aout', 'sept', 'oct', 'nov', 'dec']**
La liste des abréviations de mois à utiliser pour les légendes de graphe.

mesis.nomenclatures.**applicatifs = [('1.', 1, 'Gestion administrative et facturation', None), ('1.1', 2, 'Gestion adm**
Nomenclature des domaines et sous-domaines applicatifs

mesis.nomenclatures.**competences = [{'processus' : [{'activités' : ["Définition des orientations stratégiques et de la**
Référentiel de compétences en Systèmes d'Information

Il s'agit d'une reproduction directe en objets Python du référentiel de compétences réalisé dans le cadre du programme national Hôpital Numérique par l'ANAP

mesis.nomenclatures.**infra_base = [{'limite' : 5, 'titre' : 'Acquisition gros ordinateurs, système stockage', 'aide_ag**
Nomenclature des catégories et sous-catégories de composants d'infrastructure

mesis.nomenclatures.**referentiels = [{'référentiels' : [{'aide' : ["les données d'identité NE sont PAS <b>systémat**
Nomenclature des différents catégories et sous-catégories de référentiels pertinents pour une DSI hospitalière

mesis.nomenclatures.**regroup**(*nom*)
Retraite une nomenclature pour en donner une vision hiérarchique

Ce générateur permet de transformer la nomenclature infrastructure ou la nomenclature applicatifs en une structure hiérarchique plus facilement exploitable pour regrouper les sous-domaines ou sous-catégories en domaines ou catégories.

Paramètres **nom** – la nomenclature à restructurer

Retourne un objet générateur qui renvoie successivement pour chaque domaine ou catégorie un triplet (iden, libellé, liste des identifiants des sous-domaines ou sous-catégories)

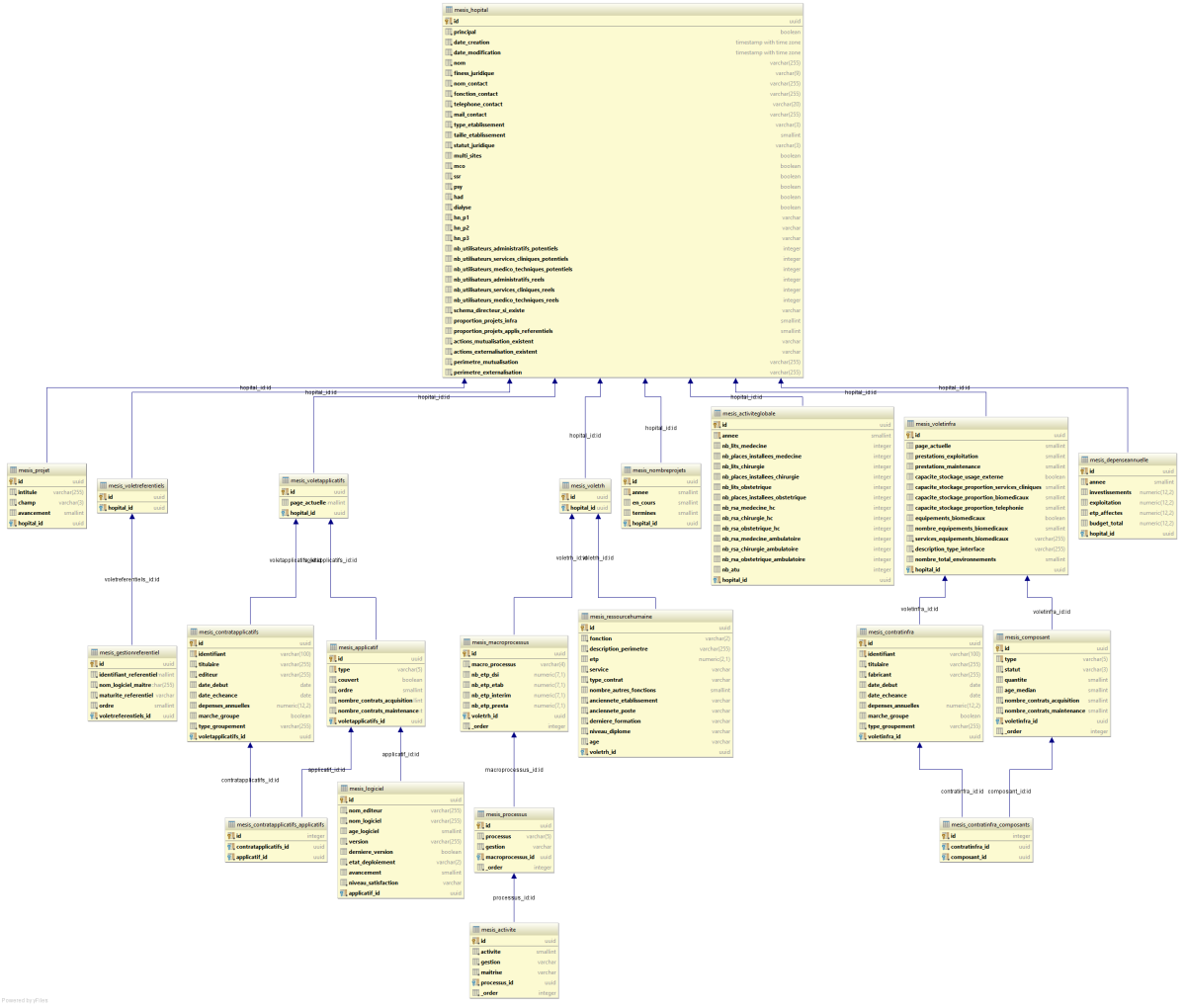
Les modèles de données

Les modèles décrivent la structure des données manipulées par l'application ID-si. Django se charge automatiquement d'en assurer la persistence en les sauvegardant en bases de données et en fournissant un certain nombre de méthodes de sélection et de filtrage.

Lors de la mise en place, Django réalise ainsi la mise en place du schéma de base de données.

7.1 Le schéma de bases de données généré

Lors de la mise en place de la base de données, Django génère le schéma suivant à partir des définitions des modèles de données :



7.2 Les modèles

7.2.1 Organisation générale

Ce module comporte l'ensemble des modèles de données qui permettront de faire persister les données saisies par l'utilisateur ainsi que celles qu'il pourra éventuellement importer à partir de données exportées par une autre instance de cette application.

Ces modèles s'organisent de la façon suivante :

- Le modèle principal de cette application est l'`Hopital` : il s'agit de la représentation d'un établissement hospitalier. Il héberge directement un certain nombre d'informations générales.
  Les autres modèles de cette catégorie y font référenec via un champ de type `django.db.models.ForeignKey`
  - `ActiviteGlobale` enregistre les données d'activité d'un établissement pour une année
  - *`DepenseAnnuelle`* enregistre les dépenses d'un établissement pour une année
  - *`NombreProjets`* comptabilise le nombre de projets d'un établissement pour une année
  - *`Projet`* permet de saisir le détail des projets d'un établissement.
- Les informations du formulaire RH
  - *`VoletRH`* se lie directement avec une instance de la class *`Hopital`* et fournit quelques méthodes utilitaires pour manipuler les informations du volet RH. Les autres modèles de cette catégorie y sont liés via un champ de type `django.db.models.ForeignKey`
  - *`MacroProcessus`* enregistre les informations liés aux macro-processus, notamment le nombre d'ETP associés
  - *`Processus`* indiquera si le processus est géré ou non
  - *`Activite`* indiquera si l'activité est géré, et si oui, avec quel niveau de maîtrise
  - *`RessourceHumaine`* détaille les informations propres à une ressource humaine.
- Les informations du formulaire infrastructure
  - *`VoletInfra`* se lie directement avec une instance de la classe *`Hopital`* et sauvegarde les champs de base du formulaire infrastructure. Tous les modèles liés au formulaire infrastructure s'y réfèrent directement.
  - *`Composant`* décrit un type composant d'infrastructure ou de matériel spécifique. Il enregistre si un composant de ce type est en service, la quantité de composants de ce type et leur âge médian. Il dispose d'un manager Django spécifique pour n'afficher que la liste des composants actifs (*`ComposantManager`*)
  - *`ContratInfra`* enregistre les informations d'un contrat d'acquisition ou de maintenance d'un élément d'infrastructure ou de matériel informatique. Un contrat peut couvrir plusieurs composants, et un composant peut être couverts par plusieurs contrats.
- Les informations du formulaire applicatifs
  - `VoletApplicatifs'se lie directement avec une instance de la class :class:'Hopital`. Tous les modèles liés au formulaire applicatifs s'y réfèrent directement.
  - *`Applicatif`* identifie un élément de couverture applicative, et indique si il est couvert par un ou plusieurs logiciels au sein de l'établissement.
  - *`Logiciel`* décrit un logiciel spécifique. Il est relié à un *`Applicatif`* dont il couvre le domaine.
  - `ContratApplis` décrit un contrat applicatif spécifique. Un contrat peut couvrir plusieurs fonctions applicatives, et une fonction applicative peut être couverte par plusieurs contrats.
- Les informations du formulaire référentiels
  - *`VoletReferentiels`* se lie directement avec une instance de la class *`Hopital`*. Les autres modèles liés au formulaire référentiel s'y réfèrent directement.
  - *`GestionReferentiel`* représente l'état de gestion d'un référentiel spécifique.

De façon générale, ces modèles permettent de créer directement les champs de formulaire affichés pour l'utilisateur. En conséquence, ils peuvent être *null*, pour permettre à l'utilisateur de faire une saisie partielle des informations et tout de même sauvegarder ses informations.

7.2.2 Détail des modèles

class mesis.models.**Activite**(**args*, ***kwargs*)
Sauvegarde l'état de gestion et le niveau de maîtrise rattachés à une activité.

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**processus_id** (*ForeignKey*) – Processus
—**activite** (*PositiveSmallIntegerField*) – le code technique de l'activité (identifiant numérique généré automatiquement à partir de la nomenclature)
—**gestion** (*CharField*) – l'état de gestion de cette activité : est-elle couverte dans le cadre des activités de la DSI ?
—**maitrise** (*CharField*) –
Notions
Capacité à mettre en oeuvre le savoir-faire dans des situations courantes et simples, en étant tutoré / encadré
Intermédiaire
Capacité à mettre en oeuvre le savoir-faire dans des situations courantes et simples, en toute autonomie
Avancé
Capacité à mettre en oeuvre le savoir-faire dans des situations courantes et complexes, en toute autonomie
Expert
Capacité à mettre en oeuvre le savoir-faire dans des situations complexes et inhabituelles, capacité à former ou à être tuteur sur le domaine concerné
—**_order** (*OrderWrt*) – order

actif()
Indique si l'activité est considérée comme active
Une activité est active si elle est indiquée comme gérée.

class mesis.models.**Applicatif**(**args*, ***kwargs*)
Identifie une fonction applicative spécifique et son état de service

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**voletapplicatifs_id** (*ForeignKey*) – Voletapplicatifs
—**type** (*CharField*) – Type
—**couvert** (*BooleanField*) – Couvert
—**ordre** (*PositiveSmallIntegerField*) – Ordre
—**nombre_contrats_acquisition_maintenance** (*PositiveSmallIntegerField*) – le nombre de contrats couvrant à la fois acqusition et maintenance.
—**nombre_contrats_acquisition** (*PositiveSmallIntegerField*) – le nombre de contrats d'acquisition portant sur ce sous-domaine applicatif
—**nombre_contrats_maintenance** (*PositiveSmallIntegerField*) – le nombre de contrats d'acquisition portant sur ce sous-domaine applicatif

nb_logiciels()
Renvoie le nombre de logiciels utilisés pour couvrir cette fonction applicative

Retourne

class mesis.models.**ChampActivite**(**args*, *categorie=None*, *automatique=True*, ***kwargs*)
Cette classe est conservée pour conserver le bon fonctionnement des migrations, mais elle n'est plus utilisée.

class mesis.models.**Composant**(**args*, ***kwargs*)
Identifie un composant d'infrastructure et son état de service

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**voletinfra_id** (*ForeignKey*) – Voletinfra
—**type** (*CharField*) – le code du type de composant détenu par l'établissement.
—**statut** (*CharField*) – le statut du composant associé.
—**quantite** (*PositiveSmallIntegerField*) – le nombre de composants de cette catégorie.

—**age_median** (*PositiveSmallIntegerField*) – l'âge médian des équipements appartenant à cette catégorie de composant.
—**nombre_contrats_acquisition_maintenance** (*PositiveSmallIntegerField*) – le nombre de contrats couvrant à la fois acqusition et maintenance.
—**nombre_contrats_acquisition** (*PositiveSmallIntegerField*) – le nombre de contrats portant uniquement sur l'acquisition
—**nombre_contrats_maintenance** (*PositiveSmallIntegerField*) – le nombre de contrats portant uniquement sur la maintenance

class mesis.models.**ComposantManager**

Un manager spécifique pour les composants d'infrastructure

Ce manager fournit une méthode spécifique *actifs()* pour ne récupérer que les éléments actifs.

Il est destiné à être utilisé aussi pour les champs reliés, notamment sur le modèle *VoletInfra*, ce qui explique que la variable de classe use_for_related_fields soit à True

actifs()

Filtre pour ne garder que les éléments actifs

Pour déterminer si un élément est actif, on regarde la variable statut qui doit être une des valeurs données dans la variable de classe du modèle EN_SERVICE.

Retourneun queryset Django qui ne comprend que les éléments actifs

class mesis.models.**ContratApplicatifs**(**args*, ***kwargs*)

Représente un contrat d'acquisition ou de maintenance d'un élément applicatif

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**voletapplicatifs_id** (*ForeignKey*) – Voletapplicatifs
—**identifiant** (*CharField*) – un identifiant unique pour le contrat ou le marché
—**titulaire** (*CharField*) – Titulaire du contrat
—**editeur** (*CharField*) – Éditeur du logiciel
—**date_debut** (*DateField*) – Date de début du contrat
—**date_echeance** (*DateField*) – Date d'échéance du contrat
—**depenses_annuelles** (*DecimalField*) – Dépenses annuelles
—**marche_groupe** (*NullBooleanField*) – Marché groupé
—**type_groupement** (*CharField*) – Type de groupement
—**applicatifs** (*ManyToManyField*) – Applicatifs

class mesis.models.**ContratInfra**(**args*, ***kwargs*)

Représente un contrat d'acquisition ou de maintenance d'un élément d'infrastructure

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**voletinfra_id** (*ForeignKey*) – Voletinfra
—**identifiant** (*CharField*) – un identifiant unique pour le contrat ou le marché
—**titulaire** (*CharField*) – Titulaire du contrat
—**fabricant** (*CharField*) – Fabricant du composant
—**date_debut** (*DateField*) – Date de début du contrat
—**date_echeance** (*DateField*) – Date d'échéance du contrat
—**depenses_annuelles** (*DecimalField*) – Dépenses annuelles
—**marche_groupe** (*NullBooleanField*) – Marché groupé
—**type_groupement** (*CharField*) – Type de groupement
—**composants** (*ManyToManyField*) – Composants

class mesis.models.**CouvertureApplicatifs**(*id*, *objet*, *contrat*, *date_ajout*)

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**objet_id** (*ForeignKey*) – Objet
—**contrat_id** (*ForeignKey*) – Contrat
—**date_ajout** (*DateTimeField*) – Date ajout

class mesis.models.**CouvertureInfra**(*id*, *objet*, *contrat*, *date_ajout*)

Paramètres

—**id** (*UUIDField*) – un identifiant universellement unique
—**objet_id** (*ForeignKey*) – Objet
—**contrat_id** (*ForeignKey*) – Contrat
—**date_ajout** (*DateTimeField*) – Date ajout

class mesis.models.**DepenseAnnuelle**(**args*, ***kwargs*)
Les dépenses globales d'un établissement sur un an

Trois instances de ce modèle seront associées à chaque hôpital pour enregistrer ses dépenses sur les trois dernières années.

Les dimensions sauvegardées sont :
—dépenses d'investissements
—dépenses d'exploitation
—etp_affectes
—budget_total (pas seulement SI)

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**hopital_id** (*ForeignKey*) – Hopital
—**annee** (*PositiveSmallIntegerField*) – Année
—**investissements** (*DecimalField*) – les investissements en systèmes d'information (y compris téléphonie) réalisées par la DSI, à exprimer en milliers d'euros
—**exploitation** (*DecimalField*) – les dépenses d'exploitation des systèmes d'information (y compris téléphonie) réalisées par la DSI, à exprimer en milliers d'euros
—**etp_affectes** (*DecimalField*) – les ETP rémunérés au sein de la DSI, à exprimer en valeur, en milliers d'euros
—**budget_total** (*DecimalField*) – le budget total (investissements et exploitation) de l'établissement, à exprimer en milliers d'euros

annee_saisie()
Indique si les données ont bien été saisies pour cette année

depenses_si()
Renvoie le total des dépenses en Systèmes d'Information de l'établissement

Retournele total des dépenses en SI (investissements + exploitation + ETP)

class mesis.models.**GestionReferentiel**(**args*, ***kwargs*)
Indique l'état de gestion d'un référentiel spécifique

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**voletreferentiels_id** (*ForeignKey*) – Voletreferentiels
—**reference** (*CharField*) – Reference
—**nom_logiciel_maitre** (*CharField*) – Nom du logiciel maître portant le référentiel
—**maturite_referentiel** (*CharField*) – le niveau de maturité de l'établissement pour la gestion de ce référentiel
—**ordre** (*PositiveSmallIntegerField*) – l'ordre dans lequel les référentiels doivent être affichés ; initialisé à partir du référentiel

class mesis.models.**Hopital**(**args*, ***kwargs*)
Modèle Django représentant un établissement hospitalier

Les établissements représentés dans cette application sont de deux natures :
—l'établissement principal, dont le champ principal est mis à True, et qui représente l'établissement qui sera renseigné dans cette instance de l'application
—les autres établissements (principal à False), qui ont nécessairement été importés à partir d'un fichier exporté par une autre instance de l'application.

Les champs de ce modèle correspondent en grande partie aux champs du formulaire "Informations générales" auxquels s'ajoutent quelques champs techniques, placés en tête.
—date_creation : date de création de l'établissement, pour indiquer à l'utilisateur quand il a débuté sa saisie
—date_modification : date de dernière modification de l'établissement (et donc date d'importation pour les établissements non principaux)

Paramètres

—**id** (*UUIDField*) – un identifiant unique (y compris entre les différentes instances de cette application) généré aléatoirement par la fonction `pkgen()` et dont la longueur est déterminée par *LONGUEUR_ID*

—**principal** (*BooleanField*) – une valeur booléenne qui indique s'il s'agit de l'établissement principal ou non.

—**date_creation** (*DateTimeField*) – la date de création de l'établissement, pour indiquer à l'utilisateur quand il a débuté sa saisie.

—**date_modification** (*DateTimeField*) – la date de dernière modification de l'établissement

—**nom** (*CharField*) – le nom de l'établissement de santé, utilisé notamment pour bien distinguer les établissements lors des comparaisons inter-établissements.

—**finess_juridique** (*CharField*) – le FINESS d'entité juridique permet d'identifier de façon unique l'établissement de soin. Il est composé de 9 caractères dont les 2 premiers correspondent au numéro du département pour la métropole. A bien distinguer du numéro FINESS d'établissement qui correspond à une implantation géographique particulière.

—**finess_geographique** (*CharField*) – Le FINESS géographique permet d'identifier de façon unique l'établissement de soin. Il est composé de 9 caractères dont les 2 premiers correspondent au numéro du département pour la métropole. A bien distinguer du numéro FINESS juridique qui correspond à une entité juridique .

—**nom_contact** (*CharField*) – le nom de la personne de référence pour le renseignement de cet inventaire

—**fonction_contact** (*CharField*) – la fonction de la personne de référence pour le renseignement de cet inventaire.

—**telephone_contact** (*CharField*) – un numéro de téléphone auquel peut être joint la personne de référence pour le renseignement de cet inventaire.

—**mail_contact** (*EmailField*) – une adresse email à laquelle peut être joint la personne de référence pour le renseignement de cet inventaire.

—**type_etablissement** (*CharField*) – le type d'établissement (CHU, CH, clinique, ESPIC, CHS, hôpital local ou CLCC)

—**taille_etablissement** (*PositiveSmallIntegerField*) – la taille de l'établissement, exprimée en nombre de lits MCO

—**statut_juridique** (*CharField*) – le statut juridique de l'établissement (public, ESPIC ou privé)

—**multi_sites** (*NullBooleanField*) – indique si l'établissement est divisé en plusieurs sites (au sens administratif du terme)

—**service_rattachement** (*CharField*) – A quelle direction au sein de l'établissement la DSI est-elle rattachée administrativement ?

—**mco** (*BooleanField*) – Mco

—**ssr** (*BooleanField*) – Ssr

—**psy** (*BooleanField*) – Psy

—**had** (*BooleanField*) – Had

—**dialyse** (*BooleanField*) – Dialyse

—**hn_p1** (*CharField*) – l'indicateur de pré-requis hôpital numériques P1/ identités, mouvements a-t-il été atteint par l'établissement

—**hn_p2** (*CharField*) – l'indicateur de pré-requis hôpital numériques P2/ fiabilité, disponibilité a-t-il été atteint par l'établissement

—**hn_p3** (*CharField*) – l'indicateur de pré-requis hôpital numériques P3/ confidentialité a-t-il été atteint par l'établissement

—**nb_utilisateurs_administratifs_potentiels** (*PositiveIntegerField*) – Ce chiffre correspond à l'effectif administratif au sein de l'établissement : par exemple, la DRH, la DAF, la Direction des achats, les services biomédicaux... Pour les établissements sociaux saisissez ici l'ensemble des personnels qui ne participent pas directement à la prise en charge.

—**nb_utilisateurs_services_cliniques_potentiels** (*PositiveIntegerField*) – Ce chiffre correspond à l'effectif des services cliniques au sein de l'établissement (toutes les unités de soin). Pour les établissements sociaux, saisissez ici l'ensemble des personnels qui participent directement à la prise en charge des patients.

—**nb_utilisateurs_medico_techniques_potentiels** (*PositiveIntegerField*) – Ce chiffre correspond à l'effectif médico-technique au sein de l'établissement (tous les plateaux techniques comme les laboratoires, l'imagerie). Cette catégorie n'est pas pertinente dans le cadre d'un établissement social.
—**externe_medecins_liberaux** (*BooleanField*) – Médecins libéraux
—**externe_infirmiers_liberaux** (*BooleanField*) – Infirmiers libéraux
—**externe_plateau_technique** (*BooleanField*) – Plateau technique de ville
—**externe_autres** (*BooleanField*) – Autres utilisateurs externes
—**schema_directeur_si_existe** (*CharField*) – Existe-t-il un schéma directeur des systèmes d'information valide ?
—**proportion_projets_infra** (*PositiveSmallIntegerField*) – la proportion des projets menés par l'établissement qui portent sur l'infrastructure, à indiquer en pourcentage
—**proportion_projets_applis_referentiels** (*PositiveSmallIntegerField*) – la proportion des projets menés par l'établissement qui portent sur les applicatifs ou les référentiels, à indiquer en pourcentage
—**proportion_projets_transversaux** (*PositiveSmallIntegerField*) – la proportion des projets menés par l'établissements qui portent transversalement sur les infrastructures, les applicatifs et les référentiels, à indiquer en pourcentage
—**actions_mutualisation_existent** (*CharField*) – Définition : La mise en commun d'objectifs, qui se décline, dans le cadre d'une gouvernance commune et dédiée, en la mise en commun de toute phase d'un projet ou service du système d'information, dans laquelle chacune des parties prenantes y trouve un avantage, et qui permet, en dépassant un seuil critique, d'atteindre des ambitions plus grandes (faisabilité du projet, niveau de service accru, etc.), à coût marginal décroissant et qualité accrue.»
—**actions_externalisation_existent** (*CharField*) – Définition : Service hébergé par un industriel externe au donneur d'ordre.
—**perimetre_mutualisation** (*CharField*) – Périmètre de la mutualisation
—**perimetre_externalisation** (*CharField*) – Périmètre de l'externalisation

derniere_annee()
Renvoie la dernière année de saisie

Cette méthode calcule la dernière année pour laquelle doivent être saisies les informations en faisant appel à la fonction *derniere_annee()*

class mesis.models.**HopitalManager**
Un manager spécifique pour les hôpitaux qui intègre la notion d'hôpital principal.

HopitalManager fournit la méthode supplémentaire *principal()* qui renvoie l'hôpital saisi dans cette application, s'il existe, plutôt que l'un des hôpitaux importés.

principal()
Renvoie l'hôpital principal

L'hôpital principal désigne l'hôpital renseigné dans les formulaires de cette instance de l'application, plutôt qu'un des hôpitaux importés à partir des données exportées par une autre instance de l'application.

Retournel'*Hopital* principal

Lève Hopital.DoesNotExistsi l'hôpital principal n'existe pas encore

mesis.models.**LONGUEUR_ID = 12**
Constante déterminant la taille des identifiants à utiliser pour les hôpitaux

Le choix d'une taille de 12 permet de limiter les risques d'avoir des collisions d'identifiants entre différentes instances de cette application.

class mesis.models.**Logiciel**(**args*, ***kwargs*)
Détaille un logiciel spécifique utilisé par un établissement

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**applicatif_id** (*ForeignKey*) – Applicatif
—**nom_editeur** (*CharField*) – Le nom de l'éditeur de ce logiciel. Ce champ pourra être récupéré automatiquement depuis oSIS

—**nom_logiciel** (*CharField*) – Le nom commercial de ce logiciel. Ce champ pourra être récupéré automatiquement depuis oSIS
—**age_logiciel** (*PositiveSmallIntegerField*) – Le temps écoulé, en années, depuis la sortie de la version majeure actuellement en service.
—**version** (*CharField*) – Le numéro de la version actuellement en service chez vous.
—**derniere_version** (*NullBooleanField*) – La version du logiciel déployée dans l'établissement est-elle la dernière version publiée, ou au moins la dernière version publiée remplissant vos exigences de stabilité, dans le cas où la toute dernière version rencontrerait encore des problèmes ?
—**etat_deploiement** (*CharField*) – L'état de déploiement actuel de ce logiciel. Ce champ pourra être récupéré automatiquement depuis oSIS
—**avancement** (*PositiveSmallIntegerField*) – L'état d'avancement du déploiement de ce logiciel. Il est de 100 % pour un projet dont le déploiement est achevé. Ce champ pourra être récupéré automatiquement depuis oSIS
—**taux_utilisation** (*CharField*) – Le taux d'utilisation approximatif de cette application par rapport au total des utilisateurs qui pourraient potentiellement en faire usage.
—**niveau_satisfaction** (*CharField*) – Le niveau de satisfaction des utilisateurs de ce logiciel, évalué à partir des retours utilisateurs que vous pourriez avoir obtenu, et, si possible, des études de satisfaction utilisateur réalisées.
—**alerte** (*BooleanField*) – Existe-t-il un risque externe sur la perennité de ce logiciel ?
Pour évaluer l'existence d'un risque, il faut prendre en compte deux éléments.
Il existe une contrainte externe faisant porter un risque sur la perennité de l'application : arrêt de la solution par l'éditeur rachat ou disparition de l'éditeur changement d'architecture ou d'orientation de la solution vous obligeant à changer de solution incapacité de la solution à évoluer dans la perspective d'un GHT (par exemple, pas de traitement multi-entités juridiques
Le délai associé à ce risque est inférieur à 3 ans.
—**alerte_raison** (*TextField*) –
—**_order** (*OrderWrt*) – order

mesis.models.**MOIS_CHANGEMENT = 7**

Le mois de changement déterminant les dates de saisie d'information

Les établissements doivent saisir certaines catégories d'informations pour les trois dernières années. Cette constante détermine le mois au cours duquel l'année demandée "changera".

class mesis.models.**MacroProcessus**(**args*, ***kwargs*)

L'ensemble des ETP rattachés à un macro-processus particulier

Les ETP sont segmentés selon leur statut (interne, intérim, prestataire) et leur direction de rattachement (DSI ou autre).

Des fonctions utilitaires permettent de vérifier si le macro-processus est actif et d'obtenir le nombre total d'ETP rattachés à ce macro-processus.

Paramètres

—**id** (*UUIDField*) – un identifiant universellement unique
—**voletrh_id** (*ForeignKey*) – Voletrh
—**macro_processus** (*CharField*) – le code du macro processus, tel que défini dans la nomenclature
—**nb_etp_dsi** (*DecimalField*) – le nombre d'ETP travaillant sur des fonctions SI et affectés directement dans les effectifs de la DSI
—**nb_etp_etab** (*DecimalField*) – le nombre d'ETP travaillant sur des fonctions SI et affectés dans un autre service de l'établissement
—**nb_etp_interim** (*DecimalField*) – le nombre d'ETP en intérim affectés sur des fonctions SI, que ce soit sur le budget de la DSI ou non
—**nb_etp_presta** (*DecimalField*) – le nombre d'ETP de prestataires externes affectés sur des fonctions SI, que ce soit sur le budget de la DSI ou non
—**_order** (*OrderWrt*) – order

actif()

Indique si le macro-processus est actif

Un macro-processus est actif à partir du moment où des ETP lui ont été rattachés.

Cet indicateur permet de savoir si les processus composant ce macro-processus doivent être affichés dans le formulaire de saisie du volet RH.

Retournevaleur booléenne indiquant si le macro-processus est actif ou non

nb_etp_total
Retourne le nombre total d'ETP rattachés à ce macro-processus

Retournele nombre total d'ETP rattachés à ce macro-processus

class mesis.models.**NombreProjets**(**args*, ***kwargs*)
Le nombre de projets conduits sur un an pour un établissement.

Trois instances de ce modèle seront associées à chaque hôpital pour enregistrer le nombre de projets en cours et terminés sur les trois dernières années.

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**hopital_id** (*ForeignKey*) – Hopital
—**annee** (*PositiveSmallIntegerField*) – Année
—**en_cours** (*PositiveSmallIntegerField*) – le nombre de projets qui étaient en cours à la fin d'une année donnée (ce qui n'inclut donc pas les projets terminés cette année)
—**termines** (*PositiveSmallIntegerField*) – le nombre de projets qui se sont terminés au cours d'une année donnée

class mesis.models.**PageMixin**(**args*, ***kwargs*)
Un *mixin* ajoutant un champ enregistrant la page_actuelle à un modèle Django.

Le *mixin* fournit aussi la méthode *sauver_page()* qui permet de changer la page actuelle pour ce composant.

Paramètres**page_actuelle** (*PositiveSmallIntegerField*) – un champ servant à sauvegarder la page actuelle (utilisé pour les formulaires multi-pages).

sauver_page(*page_num*)
Sauvegarde la nouvelle page actuelle

Vérifie si la nouvelle valeur est différente de l'ancienne, et si c'est le cas, sauvegarde le modèle.

Paramètres**page_num** – le nouveau numéro de page

RetourneNone

class mesis.models.**Processus**(**args*, ***kwargs*)
Sauvegarde l'état de gestion d'un processus

Un processus est rattaché à un macro-processus spécifique. Il sert aussi d'objet de référence pour les objets *Activite* qui le composent.

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**macroprocessus_id** (*ForeignKey*) – Macroprocessus
—**processus** (*CharField*) – Processus
—**_order** (*OrderWrt*) – order

actif()
Indique si le processus est actif

Un processus est actif s'il est indiqué comme **géré** ou **partiellement géré**. Cette valeur détermine si les objets *Activite* qui le composent seront affichés dans le formulaire de saisie du volet RH.

Retourneune valeur booléenne indiquant si le processus est actif ou non

saisi()
Indique si toutes les données du processus ont été saisies

class mesis.models.**Projet**(**args*, ***kwargs*)
La description d'un projet spécifique

Les utilisateurs peuvent, de façon facultative, saisir la liste complète des projets SI conduits dans leur établissement.

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**hopital_id** (*ForeignKey*) – Hopital

—**intitule** (*CharField*) – cet intitulé permettra d'identifier et de distinguer ce projet
—**champ** (*CharField*) – le champ sur lequel porte le projet : l'infrastructure et les matériels, les applicatifs, les référentiels, ou plusieurs de ces champs en même temps (transversal).
—**avancement** (*PositiveSmallIntegerField*) – l'état d'avancement du projet, à saisir en pourcentage
—**_order** (*OrderWrt*) – order

class mesis.models.**RessourceHumaine**(**args*, ***kwargs*)
Correspond au profil d'un employé ou d'une ressource humaine
Une ressource humaine est rattachée au volet RH d'un hôpital.

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**voletrh_id** (*ForeignKey*) – Voletrh
—**fonction** (*CharField*) – DSIDirecteur des Systèmes d'information (DSI ou DSIO) RSI-Responsable des Systèmes d'Information (RSI - RSIO) CPIChef de projet informatique ETExpert technique (réseau, système, bases de données, etc.) RAIResponsable d'application informatique A/UArchitecte / Urbaniste du SI RSSIResponsable de la Sécurité des Systèmes d'Information (RSSI) EIExploitant informatique (réseau, systèmes, bases de données, etc.) TI/ATTechnicien d'intervention / Assistant technique REFRéférent
—**description_perimetre** (*CharField*) – Description du périmètre de la fonction au sein de l'établissement
—**etp** (*DecimalField*) – le volume consacré par ce profil aux activités de la DSI, en équivalent temps plein
—**service** (*CharField*) – le service auquel est rattaché le profil, s'il est en interne, ou le statut de rattachement, s'il est en externe
—**type_contrat** (*CharField*) – le statut juridique de ce profil
—**nombre_autres_fonctions** (*PositiveSmallIntegerField*) – le nombre d'autres fonctions exercées par ce profil, en plus de la fonction d'affectation principale
—**anciennete_etablissement** (*CharField*) – l'ancienneté de ce profil dans l'établissement
—**anciennete_poste** (*CharField*) – l'ancienneté de ce profil dans le poste actuel
—**derniere_formation** (*CharField*) – Date de la dernière formation suivie par ce profil. N'indiquez que les formations diplômantes, ou directement en lien avec la fonction.
—**niveau_diplome** (*CharField*) – le plus haut niveau de diplôme obtenu
—**age** (*CharField*) – l'âge du profil
—**_order** (*OrderWrt*) – order

taux_renseignement()
Le taux de renseignement de ce profil

class mesis.models.**VoletApplicatifs**(**args*, ***kwargs*)
L'objet de référence pour tous les modèles du volet applicatifs
Ce modèle, en relation un-à-un avec le modèle *Hopital*, est l'objet auquel se réfère tous les modèles qui sauvegardent des données saisies dans le formulaire sur les aspects applicatifs. Il permet de séparer toutes les fonctions utilitaires portant sur ces dimensions.

Paramètres
—**page_actuelle** (*PositiveSmallIntegerField*) – un champ servant à sauvegarder la page actuelle (utilisé pour les formulaires multi-pages).
—**id** (*UUIDField*) – un identifiant universellement unique
—**hopital_id** (*OneToOneField*) – Hopital

taux_renseignement()
Calcule le taux de renseignement du formulaire applicatifs
Le taux de renseignement est calculé comme ceci :
—Pour la page 1 :
—Si aucun sous-domaine applicatif n'a été rempli, le score est de 0
—Sinon, on pondère par le nombre de sous-domaines applicatifs remplis :
—pour chacun, si aucun logiciel n'a été rempli, le score est de 0
—Sinon, on renvoie le taux moyen de renseignement des logiciels
—Pour la page 2, le score est calculé pour chaque composant actif :

—le score est de zéro si les nombres de contrats (acquisition et maintenance) n'ont pas été saisi
—le score est de 100% si nombre de contrats indiqués est de 0
—sinon, le score est égal à la fraction du nombre de contrats effectivement associés sur le nombre de contrats indiqués.
—Pour la page 3, le score est le taux de renseignement moyen des contrats saisis.

Retournele taux de renseignement du formulaire infrastructure

class mesis.models.**VoletInfra**(**args*, ***kwargs*)

L'objet de référence pour tous les modèles du volet infrastructure

Ce modèle, en relation un-à-un avec le modèle *Hopital*, est l'objet auquel se réfère tous les modèles qui sauvegardent des données saisies dans le formulaire sur les aspects infrastructure. Il permet de séparer toutes les fonctions utilitaires portant sur ces dimensions.

Paramètres

—**page_actuelle** (*PositiveSmallIntegerField*) – un champ servant à sauvegarder la page actuelle (utilisé pour les formulaires multi-pages).
—**id** (*UUIDField*) – un identifiant universellement unique
—**hopital_id** (*OneToOneField*) – Hopital
—**prestations_exploitation** (*PositiveSmallIntegerField*) – les prestations réalisées dans le cadre de l'exploitation des infrastructures et matériels, exprimées en euros par an.
—**prestations_maintenance** (*PositiveSmallIntegerField*) – les prestations réalisées dans le cadre de la maintenance des infrastructures et matériels, exprimées en euros par an.
—**capacite_stockage_usage_externe** (*NullBooleanField*) – Capacité de stockage utilisée pour des applicatifs gérés hors dsi
—**capacite_stockage_proportion_services_cliniques** (*PositiveSmallIntegerField*) – la proportion de la capacité de stockage qui est utilisée par des applicatifs déployés par les services cliniques.
—**capacite_stockage_proportion_biomedicaux** (*PositiveSmallIntegerField*) – la proportion de la capacité de stockage qui est utilisée par des applicatifs liés à des équipements biomédicaux.
—**capacite_stockage_proportion_telephonie** (*PositiveSmallIntegerField*) – la proportion de la capacité de stockage qui est utilisée par des applicatifs liés à la téléphonie.
—**equipements_biomedicaux** (*NullBooleanField*) – des appareils biomédicaux sont-ils connectés au SIH ?
—**nombre_equipements_biomedicaux** (*PositiveSmallIntegerField*) – le nombre d'équipements biomédicaux qui sont connectés au SIH de l'établissement.
—**services_equipements_biomedicaux** (*CharField*) – la liste des services utilisant des équipements biomédicaux.
—**description_type_interface** (*CharField*) – description des types d'interface : sont-elle normées, quelles sont les normes utilisées.
—**nombre_total_environnements** (*PositiveSmallIntegerField*) – le nombre total d'environnements distincts disponibles (paramétrage, recette, formation, production, etc.)

taux_renseignement()

Taux de renseignement pour le formulaire infrastructure

Chacune des trois pages du formulaire a un poids de 1/3.
—Pour la page 1, la première section (renseignement des composants) à un poids de 3/4, et les deux autres sections ont un poids de 1/8
—pour la première section, chaque composant à le même poids : le score du composant est de 1 s'il est saisi comme non géré, et il est égal au taux de renseignement du composant s'il est indiqué comme géré.
—pour les deux autres sections, il s'agit d'une combinatoire plus simple sur les champs remplis/non remplis
—Pour la page 2, le score est calculé pour chaque composant actif :
—le score est de zéro si les nombres de contrats (acquisition et maintenance) n'ont pas été saisi
—le score est de 100% si nombre de contrats indiqués est de 0
—sinon, le score est égal à la fraction du nombre de contrats effectivement associés sur le nombre de contrats indiqués.

—Pour la page 3, le score est le taux de renseignement moyen des contrats saisis.

Retournele taux de renseignement du formulaire infrastructure

class mesis.models.**VoletManager**

Un manager spécifique pour les volets spécifiques sur lesquels sont interrogés les utilisateurs.

Cette classe définit une méthode pour accéder au volet correspondant de l'objet principal.

principal()

Renvoie l'hôpital principal

L'hôpital "principal" est l'hôpital saisi par l'utilisateur de cette instance de l'application, par opposition aux éventuels établissements importés par la suite.

class mesis.models.**VoletRH**(**args*, ***kwargs*)

L'objet de référence pour tous les modèles du volet RH

Ce modèle, en relation un-à-un avec le modèle *Hopital*, est l'objet auquel se réfère tous les modèles qui sauvegardent des données saisies dans le formulaire sur les aspects RH. Il permet de séparer toutes les fonctions utilitaires portant sur ces dimensions.

Paramètres

—**id** (*UUIDField*) – un identifiant universellement unique

—**hopital_id** (*OneToOneField*) – Hopital

nb_etp_dsi

Retourne le nombre d'ETP affectés à la DSI

Le nombre d'ETP est renvoyé tous macro-processus confondus.

Retournele nombre d'ETP affectés à la DSI

nb_etp_etab

Retourne le nombre d'ETP affectés hors de la DSI mais intervenant sur des métiers SI

Le nombre d'ETP est renvoyé tous macro-processus confondus.

Retournele nombre d'ETP internes à l'établissement mais hors DSI

nb_etp_interim

Retourne le nombre d'ETP en intérim intervenant sur des métiers SI

Le nombre d'ETP est renvoyé tous macro-processus confondus.

Retournele nombre d'ETP en intérim

nb_etp_presta

Retourne le nombre d'ETP prestataires intervenant sur des métiers SI

Le nombre d'ETP est renvoyé tous macro-processus confondus.

Retournele nombre d'ETP prestataires

nb_etp_total

Retourne le nombre d'ETP total intervenant sur des métiers SI

Le nombre d'ETP est renvoyé tous macro-processus confondus.

Retournele nombre total d'ETP

taux_renseignement()

Renvoie le taux de renseignement du formulaire RH

Ce taux est calculé selon la formule suivante :

—les trois sections reçoivent chacune un poids de 1/3

—pour la section de saisie des ETP par macro processus :

—Si aucun ETP n'a été rempli là-haut, le score est par défaut de 0 %

—A partir du moment où un ETP a été saisi, le score est de 100 %

—pour la section de saisie des processus, on divisera le score par le nombre de processus affichés :

—un processus indiqué comme non géré compte pour 1

—un processus indiqué comme géré compte comme le taux de renseignement moyen de ses activités

—pour la section de saisie des profils

—le score est de zéro si aucun profil n'est saisi

—sinon, il est égal à la moyenne des taux de renseignement des profils saisis

total_profils()

Renvoie la somme des ETPs déclarés par profil

Retourneun nombre flottant d'ETPs

totaux()

Renvoie les totaux d'etp par statut d'affectation

Dans l'ordre, sont renvoyés :
—les ETP au sein de la DSI
—les ETP au sein de l'établissement hors DSI
—les ETP en intérim
—les ETP avec un statut de prestataire
Cette méthode est notamment utilisée pour l'affichage du formulaire RH

Retourneune liste de 4 décimaux

class mesis.models.**VoletReferentiels**(**args*, ***kwargs*)

L'objet de référence pour tous les modèles du volet référentiels

Ce modèle, en relation un-à-un avec le modèle *Hopital*, est l'objet auquel se réfère tous les modèles qui sauvegardent des données saisies dans le formulaire sur les aspects référentiels. Il permet de séparer toutes les fonctions utilitaires portant sur ces dimensions.

Paramètres
—**id** (*UUIDField*) – un identifiant universellement unique
—**hopital_id** (*OneToOneField*) – Hopital

taux_renseignement()

Renvoie le taux de renseignement du formulaire référentiels

Ce taux est égal à la moyenne des taux de renseignement de chacun des référentiels.

Retourne

mesis.models.**derniere_annee**(*date_reference*)

Renvoie l'année la plus récent pour laquelle il faudra saisir les informations

Cela se fait en comparant la date de référence (qui est soit la date du jour, soit la date de création d'un objet par exemple) avec la date de changement. Si on se trouve avant la date de changement dans l'année, l'année renvoyée est l'année actuelle moins 2. Sinon, c'est l'année actuelle moins 1.

Par exemple, si la date de changement fixée est le 1er juillet :
—si la date de référence est le 4 mars 2016 : le 4 mars se situe avant le 1er juillet, donc l'année sera l'année actuelle moins 2, soit 2014
—si la date de référence est le 18 septembre 2016 : le 18 septembre se situe après le 1er juillet, donc l'année sera l'année actuelle moins 1, soit 2015

Paramètresdate_reference – la date qui servira de comparaison

Retourne

mesis.models.**douze_prochains_mois**()

La liste des douze prochains mois à venir

Cette fonction est notamment utilisé pour la génération de certains des graphes.

Retourneune liste de tuples (mois, annee)

mesis.models.**nombre_champs_remplis**(*model*, *include=None*, *exclude=None*)

Renvoie le nombre de champs remplis

Paramètres
—**model** – l'objet modèle dont il faut compter les champs
—**include** – la liste des champs à vérifier
—**exclude** – la liste des champs à exclure de la vérification

Retournele nombre de champs remplis

mesis.models.**proportion_champs_remplis**(*model*, *include=None*, *exclude=None*)

Renvoie la proportion de champs remplis pour le modèle

Paramètres
—**model** – l'objet modèle dont il faut compter les champs
—**include** – la liste des champs à vérifier
—**exclude** – la liste des champs à exclure de la vérification

Retournela proportion de champs remplis

mesis.models.**totaux_champs**(*model*, *include=None*, *exclude=None*)

Décompte le nombre total de champs et le nombre de champs remplis pour le model

Cette fonction utilitaire est utilisée pour calculer des taux de renseignement de composants.

L'un des deux paramètres `include`, soit `exclude` doit être fourni ; les deux ne peuvent être fourni lors d'une même invocation.

Paramètres

—**model** – l'objet modèle dont il faut compter les champs

—**include** – la liste des champs à vérifier

—**exclude** – la liste des champs à exclure de la vérification

Retourneun tuple (nombre de champs non vides, nombre de champs totaux)

CHAPITRE 8

Les classes de formulaires

Ce module comporte l'ensemble des composants logiciels pour l'affichage et le traitement des formulaires saisis par l'utilisateur.

Ces composants dérivent très largement des composants par défaut fournis par Django.

`mesis.forms.`**`ActiviteForm`**
: Formset Django pour saisir le niveau de gestion des activités

  L'utilisation d'un simple ModelFormset (plutôt qu'un InlineFormset est plus aisé, car il faut générer un formulaire pour chaque Activite, indépendamment des macroprocessus auxquels ils sont rattachés. Cela permet ainsi dans la vue de traiter tous les processus en une seule fois.

  alias de `ActiviteFormFormSet`

`mesis.forms.`**`ApplicatifFormset`**
: Un Formset pour la saisie de l'état de couverture de chaque sous-domaine applicatif

  alias de `ApplicatifFormFormSet`

`mesis.forms.`**`ComposantFormset`**
: Un Formset pour la saisie des informations liées à chaque composant d'infrastructure

  alias de `ComposantFormFormSet`

`mesis.forms.`**`ContratApplicatifsFormset`**
: Un Formset pour la saisie des informations lies à chaque contrat applicatif

  alias de `ContratApplicatifsFormFormSet`

class `mesis.forms.`**`ContratInfraForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*)
: Un ModelForm spécifique pour les contrats d'infrastructure

`mesis.forms.`**`ContratInfraFormset`**
: Un Formset pour la saisie des informations liées à chaque contrat infrastructure

  alias de `ContratInfraFormFormSet`

class `mesis.forms.`**`CustomDateInput`**(**args*, ***kwargs*)
: Un champ custom pour s'assurer que les champs date s'affichent correctement

  Ce champ se contente de rajouter le texte par défaut 'JJ/MM/AAAA' pour indiquer à l'utilisateur le format de saisie des dates.

class `mesis.forms.`**`CustomInlineFormset`**(*data=None*, *files=None*, *instance=None*, *save_as_new=False*, *prefix=None*, *queryset=None*, ***kwargs*)
: Un inline formset custom qui permet de grouper des champs par catégorie.

  Sa méthode **:method :'groupes'** regroupe les champs par catégories et est destinée à être utilisée dans les templates pour l'affichage du formulaire.

class mesis.forms.**DateCreationForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*)

Le formulaire utilisé lors de la création de l'Hôpital pour mémoriser la date de création

Ce formulaire permettra de conserver la date à laquelle le formulaire a été affiché pour la première fois comme date de création de l'hôpital.

Ce formulaire est nécessaire pour gérer un ensemble de cas limites liés à la génération des objets annexes à l'hôpital pour les 3 dernières années. Comme le choix d'année est fait au moment de la génération du formulaire, il vaut mieux garder cette date comme la date de création de l'hôpital.

mesis.forms.**DepenseAnnuelleForm**

Un Formset Django pour la saisie des informations de dépense annuelle

Ce Formset permettra de faciliter la saisie des informations qui seront ensuite sauvegardées dans des instances du modèle *mesis.models.DepenseAnnuelle*

L'année sera incluse sous la forme d'un champ caché ce qui facilitera l'affichage du formulaire.

Ce Formset est utilisé pour l'affichage du formulaire *Informations générales*

alias de DepenseAnnuelleFormFormSet

class mesis.forms.**FormImport**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*)

Le formulaire d'import pour le fichier JSON d'un autre établissement.

class mesis.forms.**HopitalForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*)

Le formulaire pour les informations générales de l'hôpital

Ce modèle de formulaire permet la saisie des champs "simples" des hôpitaux, c'est-à-dire ceux qui ne font pas appel à un modèle complémentaire à *mesis.models.Hopital*

Il est utilisé sur le formulaire *Informations générales*

mesis.forms.**LogicielFormset**

Un Formset pour la saisie des logiciels rattachés à un sous-domaine applicatif

alias de LogicielFormFormSet

class mesis.forms.**MacroProcessusForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*)

Un model form custom permettant la localisation des champs pour les macro processus

Cela permet l'utilisation de la virgule comme séparateur décimal dans tous les navigateurs.

mesis.forms.**MacroProcessusFormset**

Un Formset Django pour la saisie des ETP associés aux différents macro-processus

Ce Formset est utilisé pour l'affichage du formulaire RH

L'utilisation d'un simple ModelFormset (plutôt qu'un InlineFormset) est plus aisée, car les objets Macro-processus ont déjà été créés.

alias de MacroProcessusFormFormSet

mesis.forms.**NombreProjetsForm**

Un Formset Django pour la saisie des informations de nombre de projets

Ce Formset permettra de faciliter la saisie des informations qui seront ensuite sauvegardées dans des instances du modèle *mesis.models.NombreProjets*

L'année sera incluse sous la forme d'un champ caché ce qui facilitera l'affichage du formulaire

Ce Formset est utilisé pour l'affichage du formulaire *Informations générales*

alias de NombreProjetsFormFormSet

mesis.forms.**ProjetForm**

Un Formset Django pour la saisie des descriptions des projets d'un établissement

Ce Formset permettra de faciliter la saisie des informations qui seront ensuite sauvegardées dans des instances du modèle *mesis.models.Projet*

Ce Formset est utilisé pour l'affichage du formulaire *Informations générales*

alias de `ProjetFormFormSet`

`mesis.forms.`**`ReferentielsFormset`**

Un Formset pour la saisie des informations liées à chaque référentiel

alias de `GestionReferentielFormFormSet`

class `mesis.forms.`**`RessourceHumaineForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*)

Model form custom pour la localisation des champs Ressource Humaine

`mesis.forms.`**`RessourceHumaineFormset`**

Formset Django pour les profils RH

Un InlineFormset est favorisé car il permet plus facilement la création de nouvelles instances du modèle.

alias de `RessourceHumaineFormFormSet`

class `mesis.forms.`**`VoletInfraForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*)

Un formulaire pour les champs spécifiques du VoletInfra

CHAPITRE 9

Les routes

9.1 Les routes du projet

A la racine du projet les routes suivantes sont configurées :

- `r'^api/'` : redirige vers l'api utilisée pour implémenter les composants de gestion de la couverture des contrats, `api`
- le reste redirige vers l'application `mesis`

9.2 Les routes de l'application `mesis`

- A la racine, l'accueil
  - `r'^$` : renvoie vers l'accueil pris en charge par `views.accueil()`
- les formulaires :
  - pour les informations générales de l'hôpital
    - `r'^formulaire/informations_generales$'` : servi par la vue `views.informations_generales()`
  - pour les ressources humaines
    - `r'^formulaire/ressources_humaines$'` : servi par la vue `views.rh()`
  - pour l'infrastructure
    - `r'^formulaire/infrastructure/?$'` : page d'accueil puor l'infra, servi par `views.infra()`, initialise le formulaire et redirige ensuite vers la page adéquate
    - `r'^formulaire/infrastructure/1$'` : la page 1, `views.infra_1()`
    - `r'^formulaire/infrastructure/2$'` : la page 2, `views.infra_2()`
    - `r'^formulaire/infrastructure/3$'` : la page 3, `views.infra_3()`
  - pour les applicatifs
    - `r'^formulaire/applicatifs/?$'` : `views.applis()`
    - `r'^formulaire/applicatifs/1$'` : la page 1, `views.applis_1()`
    - `r'^formulaire/applicatifs/2$'` : la page 2, `views.applis_2()`
    - `r'^formulaire/applicatifs/3$'` : la page 3, `views.applis_3()`
  - les référentiels
    - `r'^formulaire/referentiels$` : `views.volet_referentiels()`
- les restitutions
  - `r'^restitution/decideurs/$` : pour le baromètre décideurs, servi par `views.decideurs()`
  - `r'^restitution/decideurs_multi/$` : pour le baromètre décideurs multi-établissements, servi par `views.decideurs_multi()`
  - Pour les restitutions mono-établissement :
    - `r'^restitution/simple/$` : l'accueil, chargé de rediriger `views.restitution_simple()`
    - `r'^restitution/simple/(?P<categ>[a-z_]+)?$'`, les pages, par catégorie `views.restitution_simple_categ()`
  - Pour les restitutions multi-établissements :

— `r'^restitution/multi/$` : l'accueil, chargé de rediriger `views.restitution_multi()`
— `r'^restitution/multi/(?P<categ>[a-z_]+)?$'`, les pages, par catégorie `views.restitution_multi_categ()`

9.3 Les routes de l'API REST

Elles sont prises en charge par la classe `rest_framework.DefaultRouter`.

CHAPITRE 10

Les vues

Les vues définissent l'essentiel de la logique métier de l'application ID-si.

mesis.views.**APPLIS_PAGE_LIST = (('mesis :volet_applis_1', 'Recensement des applicatifs et logiciels'), ('mesis :volet**
La liste des (nom de la vue, nom de la page) pour les pages du formulaire applicatifs

Cette constante est utilisée pour afficher la pagination pour les pages du formulaire applicatifs.

mesis.views.**INFRA_PAGE_LIST = (('mesis :volet_infra_1', 'Recensement infrastructures et matériels'), ('mesis :volet**
La liste des (nom de la vue, nom de la page) pour les pages du formulaire infrastructures

Cette constante est utilisée pour afficher la pagination pour les pages du formulaire infrastructures.

mesis.views.**accueil**(*request*)
Cette vue affiche la page d'accueil.

Une boîte placée de façon bien visible renseigne l'utilisateur sur l'état de sa saisie.

Paramètres`request` – l'objet requête Django

Retournel'objet réponse Django

mesis.views.**applis**(*request*)
Cette vue coordonne les pages pour le formulaire applicatifs

Si l'hôpital n'existe pas encore, elle affiche un message renvoyant l'utilisateur vers le formulaire de saisie des informations générales de l'établissement.

Si l'hôpital existe, mais que le volet applicatifs n'a pas encore été abordé, elle crée l'objet *mesis.models.VoletApplicatifs* et tous les objets *mesis.models.Applicatif* selon la nomenclature contenue dans `mesis.nomenclatures.infrastructure`.

Par la suite, elle redirige vers la page actuelle du formulaire applicatifs. S'il n'y a pas de page actuelle, elle renvoie vers la première page.

Paramètres`request` – L'objet requête Django

RetourneL'objet réponse Django

mesis.views.**applis_1**(*request*)
Affiche la page 1 du formulaire applicatifs

Si l'objet *mesis.models.Hopital* et/ou l'objet *mesis.models.VoletApplicatifs* n'existent pas, elle renvoie vers la vue `mesis:volet_applis` (prise en charge par *applis()*).

Sinon, elle affiche les formulaires pour saisir les états de saisie des applicatifs.

Paramètres`request` – L'objet requête Django

RetourneL'objet réponse Django

mesis.views.**applis_2**(*request*)
Affiche la page 2 du formulaire applicatifs

Si l'objet *mesis.models.Hopital* et/ou l'objet *mesis.models.VoletApplicatifs* n'existent pas, elle renvoie vers la vue `mesis:volet_applis` (prise en charge par *applis()*).

Sinon, elle affiche les formulaires pour saisir les états de saisie des applicatifs.

Paramètres`request` – L'objet requête Django

RetourneL'objet réponse Django

mesis.views.**applis_3**(*request*)
Affiche la page 2 du formulaire applicatifs
Si l'objet *mesis.models.Hopital* et/ou l'objet *mesis.models.VoletApplicatifs* n'existent pas, elle renvoie vers la vue `mesis:volet_applis` (prise en charge par *applis()*).
Sinon, elle affiche les formulaires pour saisir les états de saisie des applicatifs.

Paramètres **request** – L'objet requête Django

Retourne L'objet réponse Django

mesis.views.**decideurs**(*request*)
Affiche le baromètre décideurs mono-établissement

Paramètres **request** – la requête Django

Retourne la réponse Django

mesis.views.**export**(*request*)
Vue permettant de télécharger l'établissement principal
L'accès à cette vue déclenche le téléchargement du fichier JSON d'export

Paramètres **request** – la requête Django

Retourne la réponse Django

mesis.views.**export_contrats_applicatifs**(*request*)
Permet l'export des contrats d'applicatifs.
Cette vue génère un fichier CSV avec un entête HTTP indiquant qu'il s'agit d'un téléchargement.
Chaque contrat recensé est représenté par une ligne dans le fichier CSV généré.

Paramètres **request** –

Retourne

mesis.views.**export_contrats_infra**(*request*)
Permet l'export des contrats d'infrastructures et de matériels.
Cette vue génère un fichier CSV avec un entête HTTP indiquant qu'il s'agit d'un téléchargement.
Chaque contrat recensé est représenté par une ligne dans le fichier CSV généré.

Paramètres **request** –

Retourne

mesis.views.**gestion**(*request*)
Affiche la page de gestion de l'import/export des établissements

Paramètres **request** – la requête Django

Retourne la réponse Django

mesis.views.**hopital_principal**(*request*, *hopital_id=None*)
Change l'hôpital principal.
Cette vue est utilisée par le sélecteur d'hôpital pour sélectionner le nouvel hôpital à visualiser. Si l'argument hopital_id est à `None`, c'est le cas d'un nouvel hôpital.

Paramètres **request** – la requête Django

Retourne la réponse Django

mesis.views.**import_contrats_applicatifs**(*request*)
Cette vue permet l'import des contrats applicatifs
Seul l'accès par HTTP POST est possible. Un fichier doit avoir été envoyé vie le formulaire `forms.FormImport`
Le fichier est interprété comme un fichier CSV avec le jeu de caractères Windows 1252. Les noms des champs doivent correspondre exactement à ce qui est prévu, et dans le même ordre.
Les valeurs des champs sont contrôlés pour vérifier qu'elles sont admises.
Chaque ligne du tableau conduit à :
—une mise à jour du contrat correspondant si un contrat porte le même numéro de marché
—une insertion d'un nouveau contrat dans le cas contraire.

Paramètres **request** –

Retourne

mesis.views.**import_contrats_infra**(*request*)

Cette vue permet l'import des contrats d'infrastructure

Seul l'accès par HTTP POST est possible. Un fichier doit avoir été envoyé vie le formulaire `forms.FormImport`

Le fichier est interprété comme un fichier CSV avec le jeu de caractères Windows 1252. Les noms des champs doivent correspondre exactement à ce qui est prévu, et dans le même ordre.

Les valeurs des champs sont contrôlés pour vérifier qu'elles sont admises.

Chaque ligne du tableau conduit à :
—une mise à jour du contrat correspondant si un contrat porte le même numéro de marché
—une insertion d'un nouveau contrat dans le cas contraire.

Paramètres **request** –

Retourne

mesis.views.**import_osis**(*request*)

Cette vue réalise l'import oSIS.

L'accès à cette vue n'est possible qu'en HTTP POST : tout autre accès déclenche une erreur. Si l'accès est bien en POST, on vérifie alors que le champ décrit par le formulaire `forms.FormImport` sont bien présents, et on récupère ainsi le fichier qui a été téléchargé par l'utilisateur pour l'import oSIS.

Ce fichier est ensuite interprété comme un fichier CSV, avec le jeu de caractères Windows-1252, et un séparateur point-virgule. On vérifie ensuite qu'il possède bien 3 lignes (les lignes suivantes seront ignorées).

Les sous-domaines applicatifs sont censés se situer sur la première ligne : pour éviter les problèmes liés à des espaces excédentaires ou la présence de ponctuation non prévue, on normalise les champs en retirant la ponctuation, en supprimant les espaces au début et à la fin, et en supprimant les espaces multiples.

Cela permettra ensuite de reconnaître les sous-domaines à partir de ceux recensés dans notre nomenclature, en leur faisant subir le même sort.

Pour chacun des sous-domaines ainsi reconnus, on vérifie ensuite que les quatre champs situés sur la même colonne et les trois suivants sont bien ceux attendus. Si c'est le cas, on peut alors importer ce sous-domaine.

Ces règles de gestion permettent d'importer selon les spécifications prévues dans le document envoyé à l'ATIH.

mesis.views.**importer**(*request*)

Importe un établissement externe.

Paramètres **request** – la requête Django

Retourne la réponse Django

mesis.views.**informations_generales**(*request*)

Cette vue affiche le formulaire permettant de saisir les informations générales de l'établissement

Elle commence par chercher si l'objet Hopital principal (= celui associé à cette application, plutôt qu'un établissement importé) existe et le récupère si c'est le cas. Ensuite, elle vérifie la méthode d'accès à cette vue :
—Si c'est POST, on initialise les formulaires avec ce qui a été renvoyé. Si pas d'erreur, on crée/met à jour l'hôpital principal. Sinon, on renvoie le formulaire avec les messages d'erreur.
—Sinon, on crée le formulaire à partir de l'hôpital s'il existe déjà et on l'affiche.
C'est le template `mesis/informations_generales.html`

Paramètres **request** – L'objet requête Django

Retourne L'objet réponse Django

mesis.views.**infra**(*request*)

Cette vue coordonne les pages pour le formulaire infrastructure

Si l'hôpital n'existe pas encore, elle affiche un message renvoyant l'utilisateur vers le formulaire de saisie des informations générales de l'établissement.

Si l'hôpital existe, mais que le volet infrastructure n'a pas encore été abordé, elle crée l'objet *mesis.models.VoletInfra* et tous les objets *mesis.models.Composant* selon la nomenclature contenue dans `mesis.nomenclatures.infrastructure`.

Par la suite, elle redirige vers la page actuelle du formulaire infrastructure. S'il n'y a pas de page actuelle, elle renvoie vers la première page.

Paramètres **request** – L'objet requête Django

Retourne L'objet réponse Django

mesis.views.**infra_1**(*request*)

Affiche la page 1 du formulaire infrastructure

Si l'objet *mesis.models.Hopital* et/ou l'objet *mesis.models.VoletInfra* n'existent pas, elle renvoie vers la vue `mesis:volet_infra` (prise en charge par *infra()*).

Sinon, elle affiche les formulaires pour saisir les états de saisie des composants.

Paramètres **request** – L'objet requête Django

Retourne L'objet réponse Django

mesis.views.**infra_2**(*request*)

Affiche la page 2 du formulaire infrastructure

Si l'objet *mesis.models.Hopital* et/ou l'objet *mesis.models.VoletInfra* n'existent pas, elle renvoie vers la vue `mesis:volet_infra` (prise en charge par *infra()*).

...

Paramètres **request** – L'objet requête Django

Retourne L'objet réponse Django

mesis.views.**infra_3**(*request*)

Affiche la page 3 du formulaire infrastructure

Si l'objet *mesis.models.Hopital* et/ou l'objet *mesis.models.VoletInfra* n'existent pas, elle renvoie vers la vue `mesis:volet_infra` (prise en charge par *infra()*).

...

Paramètres **request** – L'objet requête Django

Retourne L'objet réponse Django

mesis.views.**restitution_simple**(*request*)

Vue d'accès à la restitution mono-établissement : redirige vers la restitution informations générales

Paramètres **request** – la requête Django

Retourne la réponse Django

mesis.views.**restitution_simple_categ**(*request*, *categ=None*)

Affiche la restitution mono-établissement

Paramètres **request** – la requête Django

Retourne la réponse Django

mesis.views.**supprimer_hopital**(*request*)

Supprime un établissement importé.

Paramètres **request** – la requête Django

Retourne la réponse Django

mesis.views.**volet_referentiels**(*request*)

Affiche le formulaire référentiels

Paramètres **request** – la requête Django

Retourne la réponse Django

Les graphes de restitution

11.1 Les composants de dessin des graphes

class mesis.rendus.**BarAddLine**(**args*, *second_axis=True*, *second_axis_percent=False*, ***kwargs*)
Un Mixin permettant de transformer un diagramme à barre en un diagramme à barre avec une ligne additionnelle.

class mesis.rendus.**BarGrouped**(*categories*, *percent=False*, *rotated=False*, **args*, ***kwargs*)
Génère un graphe à barres groupées

Le constructeur prend en argument la liste des catégories : un groupe de barres sera affichée pour chaque catégorie, et chaque série devra définir un point par catégorie.

Les données reçues par render doivent correspondre aux différentes séries.

class mesis.rendus.**BarGroupedWithLine**(**args*, *second_axis=True*, *second_axis_percent=False*, ***kwargs*)
Un diagramme à barres groupés avec une ligne additionnelle

class mesis.rendus.**BarHorizontal**(*nom_valeur*, *units=''*, *percent=True*, **args*, ***kwargs*)
Affiche une barre d'avancement horizontale.

class mesis.rendus.**BarStacked**(*noms*, *percent=False*, *rotated=False*, **args*, ***kwargs*)
Affiche des barres empilées

class mesis.rendus.**BarStackedVariable**(*noms*, *percent=False*, **args*, ***kwargs*)
Un diagramme stacké qui fonctionne à l'envers des autres diagrammes : c'est le nombre de séries qui est connu à l'avance, et le nombre de barres variable

class mesis.rendus.**BarStackedWithLine**(**args*, *second_axis=True*, *second_axis_percent=False*, ***kwargs*)
Un diagramme à barres empilées et avec une ligne additionnelle

class mesis.rendus.**Pie**(*valeurs*, *percent=True*, **args*, ***kwargs*)
Affiche un diagramme à secteurs.

class mesis.rendus.**Radar**(*categories*, **args*, ***kwargs*)
Affiche un radar

class mesis.rendus.**Rendu**(**args*, ***kwargs*)
Classe de base pour définir un moteur de rendu de graphe

Un moteur de rendu de graphe doit redéfinir les éléments suivants :
—les champs template et (si le template l'utilise) js_class
—le constructeur `__init__()` pour initialiser les éventuels paramètre nécessaires (selon les besoins du composants : libellés de catégories, titres, etc.)
—la méthode `format_data()` à laquelle les libellés et séries fournies à `render()` sont transmis pour les metttre en forme.
—la méthode `create_context()` qui crée le contexte à partir du retour de `format_data()` et y ajoute des options supplémentaires éventuelles.

Il s'utilise ensuite en invoquant la méthode `render()` avec les deux listes de libellés et de séries en arguments. En pratique, c'est la classe `mesis.Graphe` ou la classe `mesis.Barometre` qui se chargera de cette invocation.

11.2 Les graphes pour la restitution mono et multi-établissements

class `mesis.graphs.`**`Graphe`**(*mode*, **args*, ***kwargs*)

Cette classe de base fournit le comportement de base pour le calcul et l'affichage d'un graphe

Pour générer un nouveau graphe, il suffit de :

—Hériter de cette classe

—Définir les champs titre, titre_aide, texte_aide, texte_impossible (si pertinent), rendu

—Redéfinir soit `get_queryset()` soit *`get_queryset_mono()`* et *`get_queryset_multi()`* pour générer le queryset contenant les données nécessaires pour la génération du graphe

—Redéfinir la méthode *`get_data()`* qui prend en argument le queryset et renvoie deux listes : une liste de libellés et une liste de séries de données

Elle intègre le mixin *`HtmlRenderer`*

`get_data`(*queryset*)

Méthode à redéfinir pour générer les libellés et séries de données à renvoyer au rendu

`get_other_args`()

Méthode utilitaire qui pourra éventuellement être redéfini pour rajouter des options pour le rendu

Les éléments rajoutés ici ne seront pas automatiquement convertis en JSON.

Retourneun dictionnaire de contexte additionnel

`get_queryset_mono`()

Méthode à redéfinir pour récupérer le queryset pertinent pour générer le graphe mono-établissement

La version par défaut renvoie le queryset ne comprenant que l'hôpital par défaut

`get_queryset_multi`()

Méthode à redéfinir pour récupérer le queryset pertinent pour générer le graphe mono-établissement

La version par défaut renvoie le queryset comprenant tous les hôpitaux

`information_suffisante`(*queryset*)

Méthode à redéfinir pour indiquer si les données saisies sont suffisantes pour générer le graphe

Par défaut renvoie False, entraînant l'absence d'affichage du graphe.

`render_script`()

Renvoie le tag script à insérer dans la page HTML pour générer le graphe

A appeler dans le template pertinent.

`rendu`

Propriété à redéfinir si le même rendu est utilisé pour le mono et le multi.

Sinon, il vaut mieux redéfinir séparemment `rendu_mono` et `rendu_multi`

class `mesis.graphs.`**`HtmlRenderer`**

Mixin permettant de générer le bloc html pour afficher un graphe en pleine page

Ce mixin introduit la méthode `render_html()`, et une méthode utilitaire `get_html_context()`

11.3 Les baromètres

Ce module définit les trois baromètres affichés en haut des trois pages de restitution RH, Infra et Applis et dans la restitution pour les décideurs.

`mesis.barometres.`**`BAR_VALUE`** **= {None : 0.0, 1 : 0.25, 2 : 0.5, 3 : 0.75, 4 : 1.0}**

Dictionnaire indiquant comment convertir les quatre valeurs possibles d'un indicateur en taille de barre

`mesis.barometres.`**`BM_ELEVE`** **= 3**

Une constante identifiant le niveau *Elevé* d'un baromètre

`mesis.barometres.`**`BM_FAIBLE`** **= 1**

Une constante identifiant le niveau *Faible* d'un baromètre

mesis.barometres.**BM_INTERMEDIAIRE = 2**
Une constante identifiant le niveau *Intermédiaire* d'un baromètre

mesis.barometres.**BM_NON_CALCULE = None**
Une constante qui identifie l'incapacité à calculer

mesis.barometres.**BM_TRES_ELEVE = 4**
Une constante identifiant le niveau *Très élevé d'un baromètre

mesis.barometres.**COULEURS_BAROMETRE = {1 : 'success', 2 : 'warning', 3 : 'danger', 4 : 'danger', 5 : 'danger'}**
Dictionnaire indiquant la couleur à afficher en fonction du niveau de risque.

mesis.barometres.**NIVEAUX_BAROMETRE = ['Faible', 'Moyen', 'Elevé', 'Très élevé']**
Les niveaux de baromètre tels qu'ils s'affichent

mesis.barometres.**NIVEAU_RISQUE = {('D', 3) : 2, ('D', 4) : 1, ('C', 3) : 3, ('D', 2) : 3, ('C', 2) : 2, ('D', None) : 4, ('C',**
Dictionnaire indiquant le niveau de risque associé à une valeur du baromètre, compte tenu du sens de croissance

mesis.barometres.**RISQUE_CROISSANT = 'C'**
Constante indiquant que le risque associé à l'indicateur est croissant avec sa valeur.

mesis.barometres.**RISQUE_DECROISSANT = 'D'**
Constante indiquant que le risque associé à l'indicateur est décroissant avec sa valeur.

mesis.barometres.**montrer_labels**(*labels*)
Fonction utilitaire utilisée pour montrer les labels de base de données et le pourcentage associé

Cette fonction est utilisée par le seul indicateur "Fragmentation du parc de bases de données"

mesis.barometres.**moyenne_arithmetique**(**args*)
Réalise une moyenne arithmétique

Cette fonction peut être utilisée selon deux interfaces avec un itérable comme unique argument, ou avec plusieurs arguments.

Paramètresargs –

Retourne

mesis.barometres.**moyenne_risques**(*risques*)
Fonction réalisant la moyenne de plusieurs risques

Implémente ici une moyenne quadratique (qui favorisera les risques plus élevés)

CHAPITRE 12

L'API REST

Cette API est utilisée par le module javascript chargé de gérer les associations entre composants d'infrastructure, applicatifs et contrats. Elle lui permet d'accéder directement à ces objets sous une forme programmatique.

L'application utilise la bibliothèque *django-rest-framework* qui fournit un ensemble de composants permettant de faciliter la création d'API REST.

Deux types d'éléments sont nécessaires lorsqu'on utilise *django-rest-framework* :

— Des *serializers*, chargé d'extraire des modèles de données les champs pertinents, et dans l'autre sens, de persister les modifications transmises.

— Les ressources, qui explicitent les points d'entrées de l'API (et qui viennent utiliser les serializers).

Finalement, c'est dans le module `api.urls` que ces ressources sont associées aux différentes routes, par l'intermédiaire d'un composant fourni par django-rest-framework, `rest_framework.DefaultRouter`. En plus de l'accès programmatique, il génère aussi automatiquement un environnement de test pour pouvoir tester l'API.

12.1 Les serializers

class `api.serializers.`**`ApplicatifSerializer`**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Serialize un objet *mesis.models.Applicatif*

static `get_libelle`(*applicatif*)

Renvoie le libellé correspondant à l'instance d'applicatif

Le libellé est extrait à partir de la nomenclature *mesis.nomenclatures.applicatifs*

Paramètres **`applicatif`** – l'objet composant que l'on cherche à serializer

Retourne la chaîne de caractères correspondant au libellé

static `get_logiciels`(*applicatif*)

Renvoie la liste des logiciels rattachés à l'instance d'applicatif sous la forme Editeur, Nom

class `api.serializers.`**`ComposantSerializer`**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Serialize un objet *mesis.models.Composant*

static `get_libelle`(*composant*)

Renvoie le libellé correspondant à l'instance de composant

Le libellé est extrait à partir de la nomenclature `mesis.nomenclatures.infrastructure`

Paramètres **`composant`** – l'objet composant que l'on cherche à serializer

Retourne la chaîne de caractères correspondant au libellé

class `api.serializers.`**`ContratApplicatifsSerializer`**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Serialize un objet *mesis.models.ContratApplicatifs*

class api.serializers.**ContratInfraSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Sérialize un objet *mesis.models.ContratInfra*

class api.serializers.**CouvertureApplicatifsSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Serialize un objet *mesis.models.CouvertureInfra*

class api.serializers.**CouvertureInfraSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Serialize un objet *mesis.models.CouvertureInfra*

12.2 Les ressources

class api.views.**ApplicatifView**(***kwargs*)

ViewSet en charge de l'affichage des applicatifs

Cette vue n'affiche que les applicatifs *actifs* pour l'hôpital *principal*

serializer_class

alias de ApplicatifSerializer

class api.views.**ComposantView**(***kwargs*)

ViewSet en charge de l'affichage des composants d'infrastructure

Cette vue n'affiche que les composants *actifs* pour l'hôpital *principal*

serializer_class

alias de ComposantSerializer

class api.views.**ContratApplicatifsView**(***kwargs*)

ViewSet en charge de l'affichage des contrats d'applicatifs

Cette vue n'affiche que les contrats pour l'hôpital *principal*

serializer_class

alias de ContratApplicatifsSerializer

class api.views.**ContratInfraView**(***kwargs*)

ViewSet en charge de l'affichage des contrats d'infrastructure

Cette vue n'affiche les contrats que pour l'hôpital *principal*

serializer_class

alias de ContratInfraSerializer

class api.views.**CouvertureApplicatifsView**(***kwargs*)

ViewSet en charge de l'affichage des couvertures infra

serializer_class

alias de CouvertureApplicatifsSerializer

class api.views.**CouvertureInfraView**(***kwargs*)

ViewSet en charge de l'affichage des couvertures infra

serializer_class

alias de CouvertureInfraSerializer

CHAPITRE 13

Les templates utilitaires

Un certain nombre de templates utilitaires sont présents dans le dossier *mesis/templates/mesis/utils*.

- `avec_annee.html` : génère le code HTML nécessaire pour afficher les sous-formulaires du formulaire *informations générales* dont les données doivent être saisies sur les 3 dernières années.
- `backbone.html` : génère les champs script nécessaires pour l'importation de la librairie *backbone* et de ses dépendances dans les pages qui la requièrent.
- `champ_simple.html` : génère le code HTML pour inclure un champ de formulaire et les éventuels messages d'erreur associés.
- `messages.html` : composant affichant les messages de succès et d'erreur dans les formulaires.
- `metriques.hml` : composant d'affichage des metriques conditionnelles dans le baromètre décideurs.
- `pagination.html` : composant d'affichage de la pagination pour les formulaires infrastructure et applicatifs.
- `show_errors.html` : pour l'affichage des erreurs de formsets.

CHAPITRE 14

Le front-end

14.1 Les feuilles de style

Les feuilles de style suivantes sont utilisées :

- `static/styles.css` : les styles utilisées pour les formulaires
- `static/main.css` : les styles supplémentaires utilisées pour les graphes et les restitutions
- `static/mesis.css` : des styles supplémentaires pour la bonne impression des baromètres décideurs

14.2 Le code javascript supplémentaire

- `mesis.js` comprend quelques composants génériques utilisés dans les formulaires :
  - la fonctionnalité d'ajout de ligne aux formulaires multiples
  - la fonctionnalité permettant de fixer les entêtes de tableaux
- `main.js` comprend les scripts pour l'affichage de la plupart des graphes
- `RadarChart.js` comprend le code spécifique pour RadarChart.js
- `contract.js` comprend le code spécifique pour gérer les association composant/applicatifs et contrats.

CHAPITRE 15

Les éléments pour l'empaquetage

Les éléments suivants sont utilisés pour l'empaquetage et regroupés dans le dossier `packaging` :

- Le script `start_server.py`
- La spécification du passage `package.spec`
- Le fichier de *hook* `hook-anap.py` qui comprend le code nécessaire permettant d'identifier l'ensemble des dépendances de ID-si qui devront être incluses dans le paquet.

15.1 Le point d'entrée `start_server.py`

Ce simple script est le point d'entrée du paquet qui sera généré :

- Initialiser le serveur HTTP de production (en utilisant le paquet `waitress` qui contient un serveur performant et multi-plateformes)
- Afficher le message d'information sur le rôle de la fenêtre de commande.
- Demander au navigateur par défaut l'ouverture automatique de la page.

15.2 Le fichier de spécification `package.spec`

Il contient :

- La description du paquet à produire, notamment le point d'entrée, et le nom du dossier de sortie
- Certains modules inutiles qu'il n'est pas utile d'inclure
- Les fichiers hors code Python à inclure dans le paquet

15.3 Le fichier de *hook* `hook-anap.py`

Ce script indique comment venir déterminer les dépendances de l'application ID-si pour pouvoir les inclure dans le paquet.

a

m

A

B

C

D

E

S

T

V