

Modèle d'un projet django

Documentation de la partie serveur

Arezki ABERKANE
a.aberkane@audensiel.fr

10 mars 2021

Table des matières

I	Documentation	3
1	Manuel utilisateur	4
1.1	Plan du site et navigation	4
1.2	Tableau de bord	5
1.3	Visualisation des données	7
1.4	Kube	8
1.5	Analyses	9
1.6	Jeux de données	10
1.7	Autres fonctionnalités	11
1.7.1	Accès au manuel	11
1.7.2	Support	11
1.7.3	Messagerie	13
1.7.4	Suivi des analyses	13
1.7.5	Notifications	13
1.7.6	Profil	14
1.7.7	Recherche	14
1.8	Administration	15
2	Documentation du code	16
2.1	Lancement du serveur	16
2.1.1	Initialisation	16
2.1.2	Démarrage rapide	16
2.2	Langages et environnement	17
2.3	Back-end : Django REST	17
2.3.1	Structure des dossiers	17
2.3.2	Navigation sur le site	18
2.3.3	Génération des pages	18
2.3.4	Contenu des pages	19
2.3.5	Modèles de données	20
2.3.6	APIs	22
2.4	Front-end : JavaScript	25
2.4.1	base.js	25
2.4.2	page_analysis.js	25
2.4.3	page_graphics.js	26
2.4.4	page_dashboard.js	28
2.4.5	page_kubex.js	28
2.5	Outils d'analyse	29
2.5.1	Scripts R	29
2.5.2	Scripts Python	29
II	Spécifications	30
3	Spécifications fonctionnelles	31
3.1	Préambule	31
3.1.1	Flux des données	31
3.1.2	Définitions	32
3.2	Visualisation de graphiques	32
3.3	Gestion des Box of Service	32

3.4	Lancement d'analyses	32
3.5	Gestion des jeux de données	33
4	Spécifications techniques	35
4.1	Spécifications de format annexes	35
4.1.1	Formats JSON dans les modèles	35
4.1.2	Formats dans les API	36
4.1.3	Formats de fichiers	37

Première partie

Documentation

Chapitre 1

Manuel utilisateur

1.1 Plan du site et navigation

La barre de navigation est située en haut et à gauche de chaque page du site et permet d'accéder rapidement à ses fonctions principales : Tableau de bord 1.2, Visualisation des données 1.3, gestion des *Kube* 1.4, lancement d'analyses 1.5, et gestion des données 1.6.

Ces fonctions principales sont complétées par des icônes situées à droite de la barre supérieure :

Manuel Donne accès au manuel utilisateur 1.7.1

Tickets derniers tickets ouverts 1.7.2

Messages derniers messages reçus 1.7.3

Tâches dernières analyses lancées 1.7.4

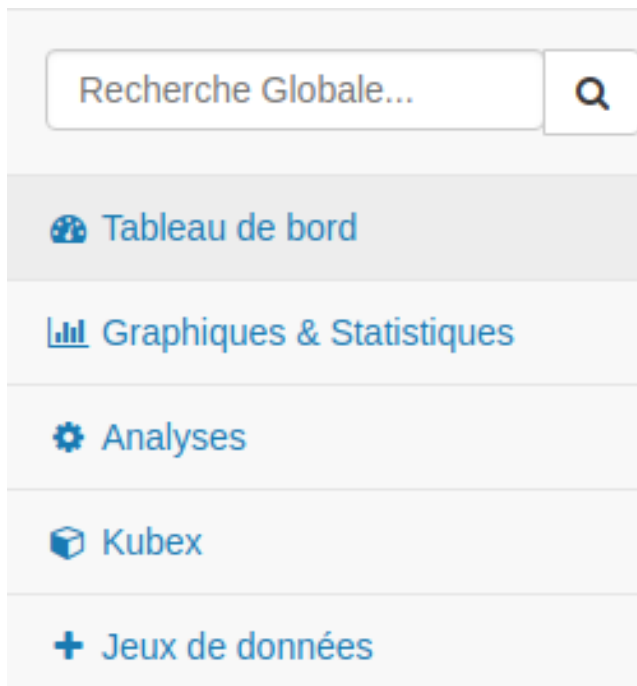
Notifications dernières notifications reçues 1.7.5

Menu utilisateur Fonctions utilisateur

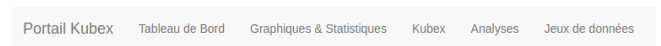
Profil Modifier son profil utilisateur 1.7.6

Paramètres Paramètres de l'interface graphique

Déconnexion déconnecte la session en cours



(a) Barre de navigation



(b) Onglets de navigation



(c) Menu général

FIGURE 1.1 – Accès rapides présents sur toutes les pages

1.2 Tableau de bord

Le tableau de bord est la page principale d'informations de l'interface Kubex, et également la première à s'ouvrir après s'être authentifié. Il contient de nombreux liens rapides et informations utiles qui permettent d'effectuer un tour d'horizon rapide de l'état des systèmes Kubex connectés et de leurs données.

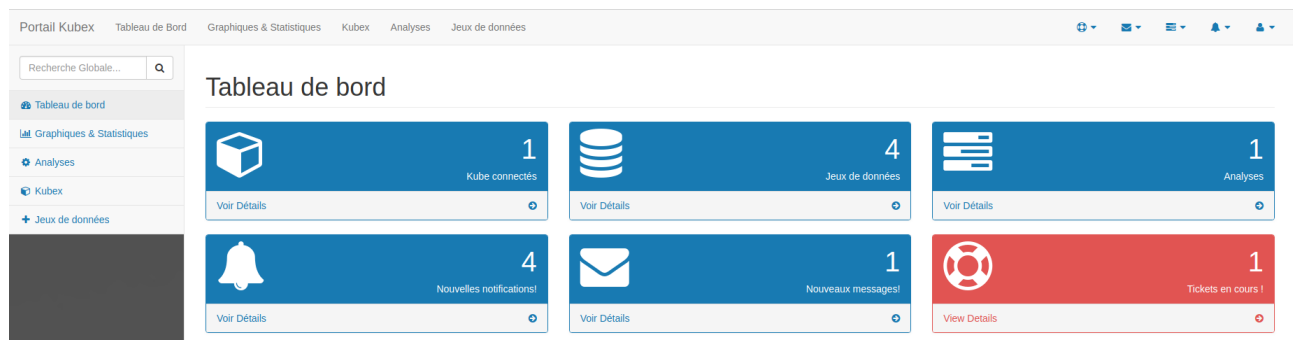


FIGURE 1.2 – Tableau de bord

Le tableau de bord présente 6 différentes boites d'information rapide :

Kube Nombre de *Kube* rattachés à l'utilisateur

Jeux de données Jeux de données auxquels a accès l'utilisateur

Analyses Analyses lancées par l'utilisateur

Notifications Nouvelles notifications

Messages Messages reçus non lus

Tickets Tickets en cours de traitement

En cliquant sur la partie *Plus de détails*, l'utilisateur accède directement à l'interface de gestion de la ressource concernée. De cette façon, il peut observer dès la page d'accueil l'état actuel du système et se rendre d'un simple clic sur les domaines où il souhaite agir.

Chercher dans les jeux de données...

Dans les champs

Tous les champs (3) ▾

Lancer la recherche 🔍

Date ▾ Titre ▾ Kube ▾ Taille ▾

FIGURE 1.3 – Barre de recherche de jeu de données

Plus bas sur cette même page sont présentés les différents jeux de données accessibles accompagnés d'une barre de recherche et de tri, ce qui permet un accès rapide à leur édition ou au lancement d'analyses. Une frise chronologique des derniers événements permet également de résumer rapidement les évolutions survenues depuis la dernière connexion.

1.3 Visualisation des données

La première étape pour comprendre les jeux de données stockés dans le cloud est de les visualiser. Pour cela l'onglet de visualisation des données propose des outils diversifiés, adaptés à chaque type de données et de besoin. Ces visualisations restent ajustables dynamiquement même après affichage pour permettre une visualisation interactive des données.



FIGURE 1.4 – Visualisation de données

La première section permet de choisir le format de données du jeu de données à afficher, parmi des formats génériques très différents :

Table Tableau formaté de données numériques ou littéraire

Image Collection d'images (incluant calques, filtres etc)

Audio Sons et filtres associés

Video Enregistrements vidéo (incluant calques, filtres, annotations etc)

La sélection de ce premier format permet de définir quel type de conteneur sera utilisé pour l'affichage, et de sélectionner les jeux de données parmi ceux dont le format correspond. L'utilisateur sélectionne alors les jeux de données pour lesquels ils souhaite une visualisation parmi la liste générée dynamiquement, puis peut ensuite affiner l'affichage au moyen du panneau de contrôle.

Ce panneau contient en premier lieu le type de visualisation à afficher (par exemple pour un tableau de données le type de courbe ou graphique), puis les données à afficher au sein des jeux sélectionnés, la façon dont elles seront affichées, et divers paramètres d'affichage.

En appuyant sur le bouton *Visualiser*, la visualisation souhaitée apparaît alors. Celle-ci reste modifiable soit au moyen du panneau de contrôle, soit directement en utilisant les outils pré-intégrés à celle-ci (zoom etc).

1.4 Kube

Cette section permet de lister les *Kube* auxquels un utilisateur a accès, renseignant les propriétés suivantes :

- Nom du *Kube*
- Numéro de série
- Utilisateurs
- Adresse IP
- Jeux de données associés
- Configuration (Administrateurs uniquement)

Pour les administrateurs des *Box of Service* et Audensiel, il est également possible de gérer les *Kube* à distance au moyen de cette interface, ce qui inclut leurs paramètres de fonctionnement mais aussi les capteurs et actionneurs auxquels ils sont reliés, les accès utilisateurs et la communication entre *Box of Service* et le cloud.



FIGURE 1.5 – Panneaux d’information et d’édition des *Kube*

L’administrateur de la *Box of Service* et les administrateurs Audensiel peuvent également voir et modifier directement la configuration des *Kube* et capteurs au moyen d’un bouton *Paramétrer* supplémentaire. Ce bouton ouvre la fenêtre modale 1.5b.

Cette fenêtre permet de reconfigurer directement les capteurs sur le serveur ; la nouvelle configuration sera propagée sous peu du serveur à la *Box of Service*. Cette fonctionnalité peut être très efficace pour adapter la configuration de la BoS toute entière, cependant son utilisation doit faire le cas d’une extrême précaution, car toute fausse manipulation pourrait mettre complètement hors d’état la *Box of Service*.

1.5 Analyses

Cet onglet permet à l'utilisateur de lancer des analyses sur tous les jeux de données auxquels il a accès, au moyen d'une liste d'algorithmes proposés.

L'utilisateur est ensuite invité à sélectionner les données d'entrée pour l'analyse depuis des listes générées à partir des jeux de données sélectionnés, de la même façon que pour la visualisation de données.

Veillez choisir l'algorithme à appliquer

Classification par apprentissage

Classification de données basée sur un apprentissage par un réseau de neurones d'après un des données similaires connues

Veillez choisir un Kubex

Kube Jasmin

Titre de l'analyse

Classification des températures

Description de l'analyse

Classification des températures normales et de surchauffe d'un Kube.

Options de analyse

Input Field

Données de prédiction

Température Kube

Classe des données

temp_class

Données d'apprentissage

Température Kube

Optional Parameter Field

Epoch

50

Taille des Batch

20

-- Veuillez choisir votre period --

Submit

FIGURE 1.6 – Formulaire de lancement d'analyses

Le formulaire ci-dessus s'affiche alors, invitant l'utilisateur à rattacher l'analyse à une *Kube*, lui donner un titre et une description. Les champs de titre et de description serviront non seulement pour l'analyse, mais aussi pour les jeux de données qu'elle générera.

Les champs nécessaires aux données d'entrée et paramètres de l'algorithme sont alors générés automatiquement, l'utilisateur n'a alors plus qu'à sélectionner les entrées souhaitées. De plus, les paramètres sont remplis avec une valeur par défaut pour permettre aussi bien aux utilisateurs novices de lancer des analyses déjà paramétrées qu'aux utilisateurs expérimentés de personnaliser plus finement leurs analyses.

1.6 Jeux de données

Cet onglet permet de gérer les jeux de données générés par les *Kube* auxquels l'utilisateur a accès. Chaque jeu de données est présenté sous la forme d'un panneau où figurent son titre, sa description, son format (tableau/image/son/vidéo), le système Kubex auquel il est rattaché, mais aussi sa taille et les mots-clés qui lui sont associés. Il est possible de renommer les jeux de données, modifier leur description, ajouter des mots-clés, mais aussi ajouter de nouveaux jeux de données externes et les associer à un *Kube* existant pour procéder à des analyses comparatives.

+ Ajouter un jeu de données

Titre du jeu de données

Veuillez choisir le Kubex d'origine

Kube Jasmin

Veuillez choisir le format des données

Table

Veuillez choisir les mots-clés associés

Description

Décrivez brièvement les données

Choisir un fichier

Aucun fichier choisi

Ajouter

Chercher dans les jeux de c

Dans les champs

Tous les champs (3)

Lancer la recherche

Date

Titre

Kube

Taille

Thermomètre interne

Température interne du Kube

Format : Table

Kube : Kube Jasmin

Senseur : 1

Date : 28 septembre 2018 17:29

Taille : 2183 octets

Modifier

Supprimer

Analyse

Export

Mots-clés:

FIGURE 1.7 – Ajout externe et modification des jeux de données

1.7 Autres fonctionnalités

Cette section détaille les autres fonctionnalités accessoires de l'interface Kubex. Celles-ci sont accessibles depuis le menus de la barre supérieure et possèdent chacune une page qui leur est dédiée. Ces fonctionnalités supplémentaires ne son pas nécessaires à l'utilisation de l'interface Kubex, mais elles présentent cependant de nombreux services utiles pour les utilisateurs.

1.7.1 Accès au manuel

Il est possible d'accéder à ce manuel utilisateur depuis la page de connexion ou depuis le menu situé en haut de chaque page, comme présenté ci-dessous :

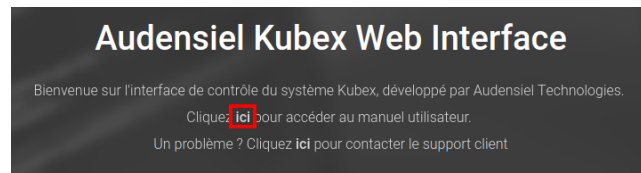


FIGURE 1.8 – Accès au manuel utilisateur depuis la page de connexion



FIGURE 1.9 – Accès au manuel utilisateur depuis le menu

En cliquant sur ces liens, ce manuel sera directement envoyé à l'utilisateur au format PDF, pour être ensuite ouvert par le navigateur directement ou téléchargé pour être consulté localement.

1.7.2 Support

Le centre de support utilisateur est accessible depuis la liste des tickets en cours, soit en cliquant sur l'icône de bouée dans le menu, soit depuis le récapitulatif présenté dans le tableau de bord (voir ci-dessous).

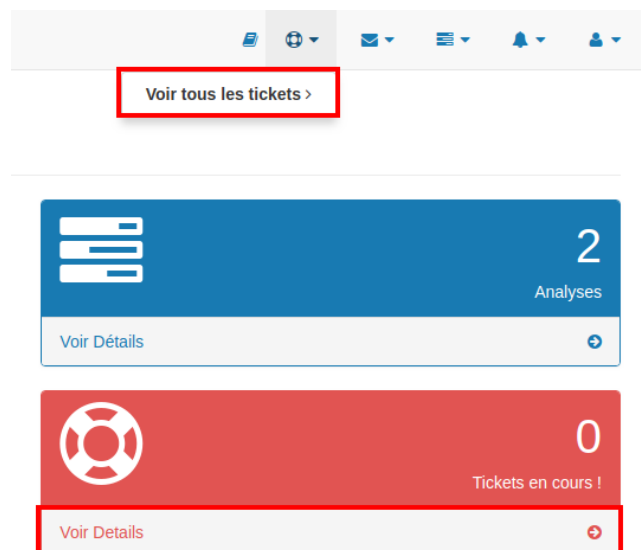


FIGURE 1.10 – Accès au support utilisateur

Le support utilisateur se base principalement sur le système de tickets : quand un utilisateur souhaite faire une requête au support Kubex pour son compte ou un tiers, rencontre des difficultés, a remarqué un dysfonctionnement ou demande de nouvelles fonctionnalités, il peut créer un ticket dans cette section, qui sera assigné à un membre du support Kubex afin d'être suivi et résolu dans les plus brefs délais.

+ Créer un ticket

Nom de l'incident

Veuillez choisir les tags associés

Description

Décrivez brièvement le problème rencontré

Les notifications concernant ce ticket seront envoyées à Testeur KUBEX

Créer le ticket

🐛 bug de redirection

La redirection des pages d'ajout de données (datasets et tickets) n'est pas encore effective et renvoie sur la page de l'API. Cette page ne devant se limiter qu'à une simple confirmation et n'étant pas mise en forme graphiquement, penser à mettre en place une redirection vers la page d'origine.

Ouvert par: Testeur KUBEX

Statut: Nouveau

👍 Résolu

✉ Message

Tags:

API

bug

développement

redirection

FIGURE 1.11 – Centre de support utilisateur

Pour créer un ticket, il suffit de lui donner un titre clair, puis de décrire aussi précisément que possible la requête dans le champ "description" et éventuellement ajouter des tags représentant le type de requête et le domaine concerné pour qu'il soit assigné aux personnes adéquates le plus rapidement possible. En cliquant sur "Valider", le ticket sera alors automatiquement envoyé au staff Kubex dont une personne pourra alors le prendre en charge. Pendant toute la période où le ticket sera ouvert, il sera possible de suivre son évolution sur cette même page, au moyen de plusieurs informations :

- Titre
- Description
- Auteur
- Statut

Nouveau Ticket nouvellement créé encore non assigné

Assigné Ticket pris en charge par un membre du staff

En attente Ticket en attente d'une action extérieure

Résolu Ticket résolu et validé par son auteur

- Responsable dans le staff
- Tags du ticket
- Actions sur le ticket
 - Marquer comme résolu
 - Envoyer un message au responsable

1.7.3 Messagerie

L'interface web Kubex propose également la possibilité d'échanger des messages entre les différents utilisateurs et administrateurs. Ceci permet de faciliter grandement la communication entre les acteurs au sein d'une *Box of Service* et avec les administrateurs Kubex. Il suffit alors d'envoyer un simple message au moyen de cette interface à la personne de son choix pour communiquer de façon rapide et efficace au sujet de Kubex, que ce soit pour échanger à propos des résultats obtenus ou d'un ticket en cours. Cette fonction est complémentaire à une boîte mail classique est amenée à être reliée à l'adresse mail utilisateur dans le futur.

Messages

Messages reçus

Non lu
De: Support AUDENSIEL
A : Testeur KUBEX
Le : 16 mai 2018 16:15
Objet : Bienvenue sur l'interface Kubex !

Bienvenue sur l'interface utilisateur de Kubex !
Vous trouverez ici tout le nécessaire pour visualiser les données issues de votre système Kubex, procéder aux analyses de votre choix, et gérer votre système Kubex à distance.
N'hésitez pas à consulter le manuel utilisateur, et en cas de problème à contacter le support Kubex en créant un ticket.
Cordialement
L'équipe Kubex Audensiel

Messages envoyés

+ Nouveau message

Destinataire :
Support AUDENSIEL

Objet du message :

Message :
Ecrivez votre message ici

Ajouter

FIGURE 1.12 – Messagerie Kubex

1.7.4 Suivi des analyses

Cette page permet d'afficher un le résumé des analyses en cours et passées. Celles-ci sont représentés au moyen de petits panneaux dont la couleur indique leur statut :

- bleu** En cours
- vert** Terminée (avec succès)
- rouge** En erreur
- jaune** En pause (non implémenté)

1.7.5 Notifications

Cette page recense l'ensemble des notifications reçues par l'utilisateur. La couleur de celles-ci recense le type de notification et permet d'avoir un aperçu rapide du nombre d'erreurs et validations.

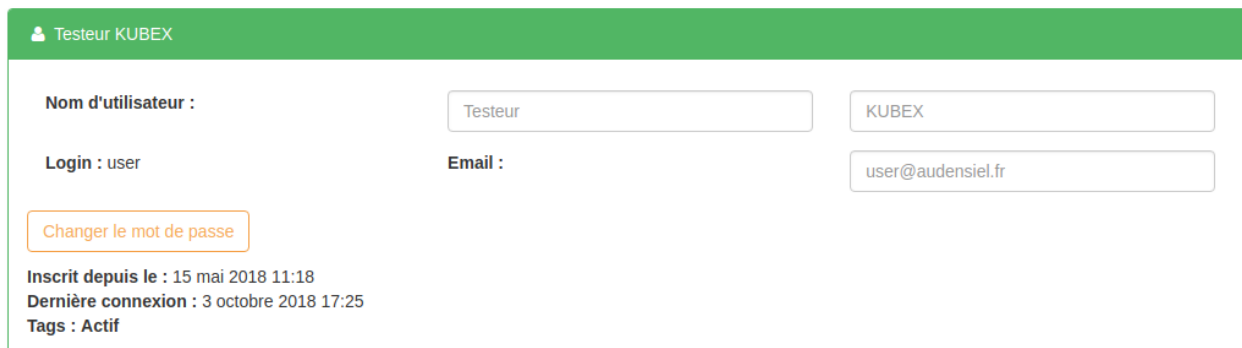
- bleu** Information générale, historique
- vert** Confirmation, tâche effectuée avec succès
- rouge** Erreur
- jaune** Avertissement, message à lire

Une fois la notification lue, il est possible de la nettoyer d'un simple clic sur la croix pour qu'elle n'apparaisse plus, ou au contraire de la conserver pour plus tard.

1.7.6 Profil

Cette page permet à l'utilisateur de modifier ses informations personnelles, ainsi que son mot de passe. Il présente également les groupes dont il est membre, ainsi que ses informations de connexion.

Profil



The screenshot shows a user profile interface. At the top, a green header bar contains a user icon and the name 'Testeur KUBEX'. Below this, the profile details are organized into two columns. The left column contains the 'Nom d'utilisateur' (Testeur), 'Login' (user), a 'Changer le mot de passe' button, and registration/login timestamps. The right column contains input fields for 'Email' (user@audensiel.fr) and 'KUBEX'. The interface is clean with a white background and green accents.

Nom d'utilisateur : Testeur

Login : user

Changer le mot de passe

Inscrit depuis le : 15 mai 2018 11:18

Dernière connexion : 3 octobre 2018 17:25

Tags : Actif

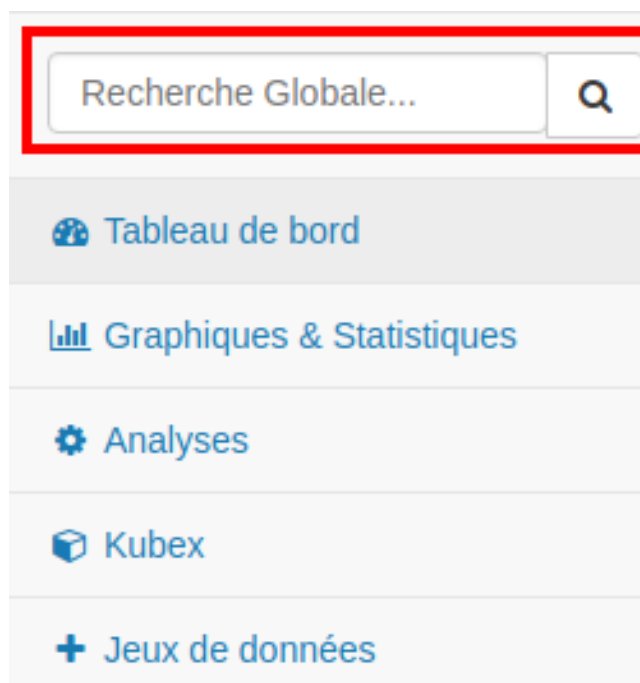
Email : user@audensiel.fr

KUBEX

FIGURE 1.13 – Profil utilisateur

1.7.7 Recherche

Au contraire des autres fonctionnalités, l'accès à cette page ne se fait pas au moyen d'un bouton de menu, mais avec la barre de recherche globale située en haut du menu latéral.



The screenshot shows a vertical sidebar menu. At the top, a search bar with the placeholder text 'Recherche Globale...' and a magnifying glass icon is highlighted with a red rectangular border. Below the search bar, the sidebar contains five menu items, each with an icon and text: 'Tableau de bord' (globe icon), 'Graphiques & Statistiques' (bar chart icon), 'Analyses' (gear icon), 'Kubex' (cube icon), and 'Jeux de données' (plus icon).

Recherche Globale...

Tableau de bord

Graphiques & Statistiques

Analyses

Kubex

+ Jeux de données

FIGURE 1.14 – Recherche dans l'ensemble des données

Cette fonction va rechercher dans les jeux de données, analyses, et messages toutes les entrées contenant les mots indiqués, que ce soit en tant que mot-clés, dans le titre, la description, ou l'utilisateur. Cette fonction est destinée à regrouper avec une seule recherche toutes les ressources disponibles sur un sujet, afin d'obtenir une vue d'ensemble d'une simple clic.

1.8 Administration

L'interface d'administration permet aux administrateurs d'Audensiel de gérer les utilisateurs, les différents *Kube*, et de répondre rapidement aux tickets ouverts. L'accès à la page en elle-même est réservé aux utilisateurs disposant de droits d'administration sur le site.

Panneau d'administration



Utilisateurs

Support AUDENSIEL
Jasmin CÉVOST
Testeur KUBEX

FIGURE 1.15 – Interface d'administration

Cette interface propose l'accès aux trois catégories détaillées ci-dessus :

Utilisateurs Donne un accès administrateur à tous les profils d'utilisateur pour modifier leurs droits et éventuellement mot de passe (en cas de perte)

Kubex Donne un accès administrateur à tous les Kubex pour accéder et gérer leur configuration à distance

Tickets Liste les tickets en cours pour permettre de se les assigner et résoudre les demandes utilisateurs.

Depuis cette interface, les administrateurs Audensiel pourront donc directement effectuer des tâches d'administration sur les éléments de ces trois catégories, leur permettant un accès rapide, intuitif et réduisant le temps nécessaire à leurs interventions.

Chapitre 2

Documentation du code

Ce projet contient tous les fichiers nécessaires au déploiement d'une interface utilisateur pour le cloud Kubex, y compris une base de données basique et les prérequis de l'environnement virtuel.

2.1 Lancement du serveur

2.1.1 Initialisation

Installer et configurer PostgreSQL si nécessaire.

```
sudo apt-get install postgresql postgresql-contrib
sudo -su postgres
psql
postgres=> CREATE USER audensiel PASSWORD 'Audensiel2018';
postgres=> CREATE DATABASE kubex;
```

Charger la base de données kubex :

```
psql -U audensiel kubex < backup/database.psql
```

Pour sauvegarder les modifications de la base de données dans le fichier *backup.psql*, la commande est :

```
pg_dump -a kubex -f backup/database.psql
```

Ensuite, installer les modules nécessaires dans un environnement Python V3.5 :

```
pip install -r cloud_GUI/requirements.txt
```

Une fois l'environnement initialisé, vérifier que la base de données est à jour avec les modèles django et réaliser les migrations si besoin. Ceci permet également de tester le bon fonctionnement ainsi que la configuration de l'environnement et de la base de données.

```
python manage.py makemigrations
python manage.py migrate
```

2.1.2 Démarrage rapide

Script de lancement global

Pour plus de simplicité, les commandes lançant le serveur Django, le serveur de calcul Celery, et le serveur de réception des données Bottle sont rassemblées dans le fichier *launcher.sh* qu'il suffit d'appeler en ligne de commandes :

```
./launcher.sh
```

Les outputs de chacun de ces scripts seront stockés dans le répertoire *logs*, avec pour chaque processus la sortie standard dans un fichier *.out* et la sortie d'erreur dans un *.err*, de la façon suivante :

— serveur : GUI.*

Détail des commandes

Les différentes commandes qui le composent peuvent être lancées séparément, à commencer par l'activation de l'environnement virtuel :

```
source my_env/bin/activate
```

L'environnement virtuel permet de conserver des versions fonctionnelles pour tous les modules Python, et également de pouvoir être déployé rapidement sur une nouvelle infrastructure.

Pour sortir de cet environnement virtuel, il suffira de taper :

```
deactivate
```

.

Enfin, lancer localement le serveur de développement

```
python manage.py runserver 0:8080
```

Pour y accéder ensuite, entrer l'adresse `http://localhost:8080` (ou remplacer *localhost* par l'adresse IP du serveur depuis un PC distant).

Les comptes réservés permettant d'accéder à cette interface dont :

audensiel (mdp :Audensiel2018) Compte administrateur

user (mdp :user123) Compte de test standard

2.2 Langages et environnement

Le code de la partie serveur est constitué des langages HTML, CSS et Javascript pour la partie front-end, tandis que la partie back-end est assurée par le framework django REST dans un environnement virtuel en python v3.5. Les modules requis pour cet environnement peuvent être importés depuis le fichier *requirements.txt* du dossier *backup*.

La base de donnée utilisée est postgresQL ; une copie de sauvegarde de la base de donnée utilisée pour le développement se trouve dans le fichier *database.psql*. Le déploiement en production est prévu via un serveur apache2 qui sera configuré ultérieurement.

2.3 Back-end : Django REST

Le back-end de l'interface graphique est assuré par le framework Django en utilisant un environnement virtuel de Python v3.5, dont les bibliothèques sont installables par *pip* en utilisant le fichier versionné *requirements.txt*. La base de données postgresQL dont le backup se situe dans le fichier *backup.psql* doit également être opérationnelle.

2.3.1 Structure des dossiers

L'organisation des dossiers de l'interface du cloud s'effectue sur la base du Django REST Framework : dans le dossier principal se trouvent le dossier racine (*Kubex_GUI*), ainsi que le dossier de l'application (*data*). Viennent s'y ajouter les dossier servant à stocker des données, logs, d'autres modules ou scripts.

L'architecture générale est présentée ci-dessous, les fichiers permettant de lancer et configurer le serveur ont été soulignés.

backup/ dossier contenant les données de sauvegarde comme *requirements.txt*, *database.psql* et une copie des configurations des

data/ ** Application web principale contenant l'essentiel du code et templates

doc/ Documentation du GUI, incluant le manuel utilisateur

kubex_data/ * Stockage local des jeux de données et configurations

datasets/ Jeux de données classés par numéro de Kubex

- config/** Fichiers de configuration des Kubes et senseurs
- Kubex_GUI/** Dossier principal contenant les fichiers de configuration de django
- settings.py** Configuration générale du serveur
- static/** Fichiers statiques du serveur (JavaScript, CSS, images...)
- launcher.sh** Script de lancement global de tous les services
- logs/** * Contient les logs de tous les modules
- manage.py** Script principal lançant le serveur web
- scripts/** * Lien (symbolique) vers les scripts d'analyse??
- tmp/** * Dossier de stockage temporaire des données en provenance des BoS

Le contenu de ces dossiers n'est pas pris en compte par *git*.

* L'essentiel des fonctions du serveur web se trouvant dans le dossier *data*, la liste détaillée est présentée ci-dessous pour plus de clarté :

- admin.py** * Fonctionnalités réservées aux administrateurs
- apis.py** Fonctions d'interaction utilisateur/serveur (API)2.3.6
- apps.py** * Configuration de l'application (data)
- celery.py** * Configuration de Celery
- migrations/** Migrations effectuées et en cours (par ex : modèles dans DB)
- models.py** Modèles de données2.3.5
- serializers.py** * Sérialiseurs associés aux modèles
- tasks.py** Tâches exécutables par Celery??
- templates/** Templates html utilisées pour l'affichage des pages2.3.4
 - panel/** Sous-templates d'éléments intégrés aux pages
- templates-factory/** Modèles de templates pour le développement futur
- tests.py** * Tests et validation
- urls.py** Mapping des urls vers les pages2.3.2 (*views.py*) ou APIs (*apis.py*)
- views.py** Génération dynamique des pages2.3.3

* Pour ces fichiers, seule la configuration de base est utilisée

2.3.2 Navigation sur le site

La navigation entre les différentes pages de l'interface se fait d'après le mapping réalisé dans le fichier *urls.py*. Ces URLs correspondent en général à deux types de fonctionnalités : des *API* permettant d'interagir avec le serveur au moyen de fonctions du module *apis.py* (URLs commençant par *api/<model>/*)2.3.6 , ou bien des *vues* basées sur des fonctions du module *views.py* générant dynamiquement des pages HTML2.3.3.

2.3.3 Génération des pages

Chacune des pages parcourues par l'utilisateur est générée dynamiquement par une fonction contenue dans le module *views.py*. Ces fonctions génèrent les pages en se basant sur trois éléments distincts : la requête envoyée par l'utilisateur, la template de la page à renvoyer, et enfin le contexte qui permettra de modifier dynamiquement la template.

Après avoir vérifié l'authentification de l'utilisateur, la fonction appelée via *urls.py* charge la template appropriée, puis le contexte général (dictionnaire) utilisé pour les barres de menus (via la fonction *baseView*), et enfin ajoute les éléments de contexte spécifiques à chaque page dans le dictionnaire. Elle utilise ensuite la fonction *HttpResponse* afin de renvoyer le *render* de la template en utilisant les informations présentes dans la requête et le contexte.

URL	Fonction	Description
/	views.root	Redirige vers 'dashboard'
index	views.index	Redirige vers 'dashboard'
login	auth_views.LoginView.as_view	Page de connexion
admin	views.admin	Page d'administration
dashboard	views.dashboard	Tableau de bord
graphics	views.graphics	Page de visualisation graphique
analysis	views.analysis	Page d'analyse
api/analysis/launch	apis.launchAnalysis.as_view	Lancement d'une analyse
kubex	views.kubex	Page des Kubex
api/kubex/status	apis.getKubexConfig.as_view	Informations sur le Kube
api/kubex/config	apis.setKubexConfig.as_view	Modification du Kube
datasets	views.datasets	Jeux de données
inbox	views.inbox	Boîte de messagerie
api/algorithm/<id>	apis.getAlgorithmParameters.as_view	Chargement d'un algorithme
api/algorithm/list/	apis.getAlgorithms.as_view	Liste les algorithmes
tasks	views.tasks	Page listant les analyses
alerts	views.alerts	Notifications
api/alerts/delete/<id>	apis.deleteAlert.as_view	Supprimer une notification
profile	views.profile	Profil utilisateur
settings	views.settings	Paramètres (TODO)
logout	auth_views.LogoutView.as_view	Déconnexion
search	apis.searchQuery.as_view	Lancement d'une recherche
support	views.support	Centre de support/tickets
support/add	apis.addTicket.as_view	Création d'un ticket
manual	apis.getManual.as_view	Affiche le manuel utilisateur
api/dataset/list/	apis.getDatasetList.as_view	Liste les jeux de données (par format)
api/dataset/add	apis.addDataset.as_view	Ajout d'un jeu de données
api/dataset/edit	apis.editDataset.as_view	Edition des métadonnées
api/dataset/<id>	apis.getDatasetMetadata.as_view	Chargement des métadonnées
api/dataset/delete/<id>	apis.deleteDataset.as_view	Suppression d'un jeu de données
api/dataset/column	apis.getDatasetColumns.as_view	Chargement des données
api/dataset/search	apis.searchDataset.as_view	Recherche d'un Dataset
api/message/add	apis.addMessage.as_view	Ecriture d'un message

TABLE 2.1 – Correspondances des URLs

2.3.4 Contenu des pages

Le contenu de chaque page est défini dans le dossier *templates* ; celles ci correspondent généralement à une page qui sera générée par une fonction de *views.py*, avec deux exceptions notables :

- La template *base.html* sert de base pour toutes les autres. Elle en définit la forme général, le menu, ainsi que les scripts JavaScript généraux. Les templates qui en héritent inséreront simplement leur(s) code(s) spécifique(s) aux endroits appropriés.
- Les sous-templates du sous-dossier *panels* correspondent à des éléments répétés dans plusieurs pages, correspondant à la représentation d'instances des modèles

Dans une page classique (ici *datasets.html*), on peut retrouver les éléments dynamiques suivants :

```
{% extends 'base.html' %}
```

Cette balise signifie que la template hérite de celle comprenant la forme générale et les menus. Lors du rendu, la page web finale inclura le code de la template ciblée à l'intérieur de *base.html*, et le positionnera selon les balises suivantes :

```
{% block css %}
<liens CSS>
{% endblock %}
{% block content %}
<code de la template>
{% endblock %}
{% block scripts %}
```

```
<liens JavaScript>
{% endblock %}
```

Ces "blocs" délimités par les balises `{%block <nom>%}` et `{% endblock %}` définissent les segments qui seront inclus dans la template `base.html`. Tout le code présent à l'intérieur d'un bloc sera inséré entre les balises correspondantes de la template `base.html`.

Le processus inverse est également possible : il consiste à inclure (au moyen de l'instruction *include*) des segments de code HTML + Django préformatés à l'intérieur d'une page. Cette méthode possède deux avantages principaux : elle permet de réutiliser un template simple dans l'ensemble des pages tout en conservant une unique copie modifiable, et ils permettent également de redéfinir le contexte particulier pour ce sous-template.

L'exemple ci-dessous illustre la façon dont les panneaux correspondant à l'ensemble des datasets sont générés de façon simple et efficace :

```
{% for dataset in datasets %}
{% include "panels/dataset.html" with dataset=dataset only %}
{% endfor %}
```

- La boucle *for* permet de générer le code pour chaque instance de *datasets*
- L'instruction *include* indique de charger l'ensemble de la sous-template `panels/dataset.html` à cet emplacement, et restreint le contexte uniquement au *dataset* actuellement examiné par la boucle
- Le *render* générera à chaque itération le panneau d'information correspondant au jeu de données au moyen du contexte fourni

Une autre fonctionnalité utile offerte par Django pour les templates est de générer automatiquement les liens vers les fichiers statiques. Pour ce faire, le ou les emplacements des fichiers statiques doivent être définis dans `settings.py` (correspondant ici au dossier `Kubex_GUI/static`). L'instruction `{% load static %}` cherchera alors les fichiers demandés parmi les fichiers statiques pour générer les liens dynamiquement, même en cas de déplacement des fichiers.

```
{% load static %}
<link href="{% static 'assets/css/page_dataset.css' %}" rel="stylesheet">
<link href="{% static 'assets/bootstrap-multiselect/bootstrap-multiselect.css' %}" rel="st
<script src="{% static 'assets/bootstrap-multiselect/bootstrap-multiselect.js' %}"></scrip
<script src="{% static 'js/page_dataset.js' %}"></script>
```

Pour finir, il est également possible de fournir une sécurité contre les attaques de type *Cross Site Request Forgery* au moyen d'un jeton généré lors de l'authentification et reporté dans chaque formulaire, via l'instruction suivante :

```
{% csrf_token %}
```

Cette instruction génère un jeton d'authentification propre à la session en cours, qui devra être identique dans toutes les requêtes effectuées par l'utilisateur, et évitera ainsi d'effectuer des actions non désirées en cliquant sur un lien non généré par le site.

2.3.5 Modèles de données

Les données gérées par l'interface graphiques sont stockées dans une base de données PostgreSQL dont Django gère les modèles. Les sections ci-dessous présentent les modèles exprimés dans le module `models.py` qui correspondent ensuite à des tables dans la base de données PostgreSQL dont le nom sera `data_<modèle>`. Django est en charge de la gestion des contraintes, liens entre les tables et clés primaires. Lorsque ce n'est pas explicitement précisé autrement, la clé primaire de chaque modèle est un champ entier *id* auto-incrémental.

Les relations de type *ManyToMany* impliquent la création d'une table intermédiaire (`data_<modèle1>_<modèle2>`) qui servira de lien entre deux modèles. Les relations de type *GenericForeignKey* référencent une autre entrée pouvant appartenir à différents modèles. Dans ce but, les champs `object_id` et `content_type_id` seront créés pour référencer le type de modèle au sein de la table `django_content_type` (`content_type_id`), et l'identifiant au sein de la table associée à ce modèle (`object_id`). Une *GenericRelation* est également créée dans les modèles ciblés afin de permettre les requêtes dans le sens inverse.

Kubex

Nom	Type	Description
IP	GenericIPAddressField	Adresse IP du <i>Kube</i>
SN	PositiveIntegerField	Numéro de série du <i>Kube</i> *
name	CharField	nom du <i>Kube</i>
owner	ForeignKey('KubexUser')	Administrateur du <i>Kube</i>
users	ManyToManyField('KubexUser')	Utilisateurs de ce <i>Kube</i>

Ce champ constitue la clé primaire du modèle.

Les données sont triées dans l'ordre du champ *name*.

Sensor

Nom	Type	Description
Kubex	ForeignKey	Kubex d'appartenance
SID	PositiveInteger Field	ID du capteur
config	FileField	Fichier de configuration
title	CharField	Nom du capteur
data	GenericRelation	Lien avec les Datasets

Message

Nom	Type	Description	Défaut
sender	ForeignKey('auth.User')	Expéditeur	request.user
recipient	ForeignKey('auth.User')	Destinataire	
title	CharField	Objet du message	
time	DateTimeField	Date d'envoi	auto_add_now
description	CharField	Contenu du message	
status	CharField	Non Lu/Lu/Archivé	Non Lu

Les données sont triées dans l'ordre inverse du champ *time*.

Ticket

Nom	Type	Description	Défaut
owner	ForeignKey('auth.User')	Auteur du ticket	request.user
in_charge	ForeignKey('auth.User')	Admin en charge	audensiel
creation_time	DateTimeField	Création du ticket	auto_add_now
modification_time	DateTimeField	Dernière mise à jour du ticket	auto_now
title	CharField	Titre du ticket	
status	CharField	Nouveau/Assigné/Résolu/En attente	Nouveau
description	CharField	Description du problème/demande	
keywords	ManyToManyField('Keyword')	Tags associés	

Les données sont triées dans l'ordre inverse du champ *modification_time*.

Dataset

Nom	Type	Description	Défaut
kubex	ForeignKey('Kubex')	Kubex d'origine	
description	CharField	Description du jeu de données	
title	CharField	Titre du jeu de données	
data_format	CharField	Table/Image/Audio/Vidéo	Table
creation_time	DateTimeField	Création	auto_now_add
modification_time	DateTimeField	Dernière modification	auto_now
keywords	ManyToManyField('Keyword')	Mots-clés associés	
data	FileField	Fichier de données	
origin	GenericForeignKey	Origin du Dataset	

Les données sont triées dans l'ordre inverse du champ *modification_time*.

Alert

Nom	Type	Description	Défaut
owners	ManyToManyField('auth.User')	Liste de diffusion	
status	CharField	Indication/Erreur/Validation/Avertissement	Indication
time	DateTimeField	Date de la notification	auto_now_add
title	CharField	Titre de la notification	
source	GenericForeignKey	Source de la notification	

Les données sont triées dans l'ordre inverse du champ *time*.

Analyse

Nom	Type	Description	Défaut
status	CharField	en cours/arrêtée/en erreur/terminée	en erreur
owner	ForeignKey('auth.User')	Propriétaire de l'analyse	request.user
kubex	ForeignKey('Kubex')	Kubex associé	
datasets	ManyToManyField('Dataset')	Datasets d'entée	auto_now_add
creation_time	DateTimeField	Date de lancement	auto_now
modification_time	DateTimeField	Dernier accès	
title	CharField	Titre de l'analyse	
description	CharField	Description de l'analyse	
algorithm	ForeignKey('Algorithm')	Algorithme utilisé	
progress	DecimalField	avancement de l'analyse (%)	0%
inputs	JSONField	Données d'entrée	
parameters	JSONField	Paramètres utilisés	
command	CharField	Commande de lancement	
results	GenericRelation('Dataset')	permet le lien avec le dataset généré	

Les données sont triées dans l'ordre inverse du champ *modification_time*.

Algorithm

Nom	Type	Description	Défaut
algo_type	CharField	Régression/Classification/ Statistiques/Apprentissage/Analyse factorielle/Traitement d'image	Table
data_format	CharField	Table/Image/Audio/Vidéo	
title	CharField	Titre de l'algorithme	
inputs	JSONField	Données d'entrée	
parameters	JSONField	Paramètres	
description	CharField	Description de l'algorithme	
command	CharField	Commande de lancement	
progression_regex	CharField	Format de l'indicateur de progression	
output	JSONField	Données de sortie	

Les données sont triées dans l'ordre du champ *title*.

Keyword

Nom	Type	Description	Défaut
word	CharField	mot-clé	NA

Les données sont triées dans l'ordre du champ *word*

2.3.6 APIs

Une majorité des fonctions offertes par Django réside dans les *APIs* grâce auxquelles l'utilisateur peut interagir avec les données. Ces APIs sont rassemblées dans le fichier *apis.py* et sont généralement appelées via une fonction JavaScript2.4 renvoyant à une adresse de la forme *api/<modèle>/<fonction>*.

Elles sont constituées d'une classe dérivée de *APIView*, dont au moins une fonction est nommée *get* ou *post* et correspond à la méthode HTML associée. Lors de l'envoi d'une requête à une de ces fonctions, l'objet *request*

contiendra toutes les données associées, principalement l'utilisateur dans *request.user*, et les données dans *request.query_params* pour un GET ou *request.data* pour un POST.

Les APIs vont alors vérifier que l'utilisateur est authentifié, charger les modèles requis par la requête, éventuellement procéder à des traitements supplémentaires puis renvoyer une réponse HTML. Cette réponse sera généralement sous un format JSON accompagnée du code HTML indiquant le succès ou l'échec de la requête.

Manual

On accède à cette API en cliquant sur les liens *Manuel* proposés sur la page d'authentification et le menu supérieur 1.7.1, qui redirigent vers l'URL */manual*.

Cette URL redirige vers la classe *getManual* dont la méthode *get* récupère simplement le fichier *doc/user-manual.pdf* et l'affiche dans la fenêtre actuelle. Ce document contient la documentation utilisateur de la GUI qui peut être trouvée au chapitre 1, à l'exception de la section réservée aux administrateurs 1.8.

Search

On accède à cette API par le champ *recherche globale* situé dans le menu latéral gauche 1.7.7, qui renvoie à l'url */search?content=<kw1,kw2...>*. La fonction *get* de la classe *searchQuery* effectue alors une recherche dans les jeux de données, analyses, et messages (à ce jour par mots exacts, insensibles à la casse) et renvoie une liste de résultats dans la page *search* 1.7.7.

Cette API génère une page de la même façon que les fonctions du module *views.py* 2.3.3, à la différence qu'elles prennent en entrée une requête complète, puis effectuent 3 recherches poussées dans la base de données au moyen de fonctions *Q* servant à combiner les critères, et de *iregex* permettant de chercher une expression régulière de façon insensible à la casse. Chacune des requêtes charge un élément du contexte (*query_<modèle>* avec *<modèle>* parmi *dataset*, *analyse*, ou *message*)

Algorithmes

On accède à ces API dans la page *analyse* 1.5, lors de la sélection de l'algorithme utilisé pour l'analyse. Elles permettent à l'utilisateur de sélectionner et paramétrer un algorithme d'analyse pour répondre de la façon la plus adaptée possible à ses besoins.

Liste des algorithmes En sélectionnant le format des données, un appel à l'URL */api/algorithm/list?dataFormat=<T|A|I|V>* redirigeant vers la méthode *get* de la classe *getAlgorithms* va permettre de récupérer la liste des algorithmes applicables. La liste (titre et identifiant) des algorithmes correspondant au format de données choisi (*dataFormat* valant *T/A/I/V* pour *Table/Audio/Image/Vidéo*) sera renvoyée au format JSON pour être affichée dans le champ adéquat par un JavaScript, accompagné du code 200. En cas d'erreur, un code 500 et un message d'erreur seront retournés.

Chargement d'un algorithme Lorsque l'on sélectionne un algorithme parmi la liste proposée dans le menu déroulant, un appel à l'URL */api/algorithm/(?P<algo_id>+)* redirigeant vers la méthode *get* de la classe *getAlgorithmParameters* permettra de charger les différents champs requis pour les données d'entrée et les paramètres qui seront utilisés lors de l'analyse. La valeur de *<algo_id>* est utilisée pour effectuer la requête de l'algorithme dans la base de données, l'API renvoie toutes ses propriétés sous forme de JSON ainsi qu'un code HTML 200 en cas de succès, un message d'erreur et un code HTML 500 en cas d'erreur.

Ces propriétés serviront à générer automatiquement tous les champs requis au paramétrage de l'algorithme??.

APIs des analyses

Une fois tous les champs requis requis, l'utilisateur peut lancer une analyse, ce qui se traduira par un appel à l'url *api/analysis/launch*, qui transmettra toutes les informations fournies à la fonction *post* de la classe *launchAnalysis*. Celle-ci est en charge de créer l'objet *Analyse* associé et de le transmettre à la tâche Celery *launchAnalysis??*.

Elle crée également l'objet *PeriodicTask* associé à cette Analyse si celui-ci est requis, ce qui permet de lancer cette même analyse à des intervalles réguliers sur les nouvelles données générées par les mêmes capteurs.

En cas de succès de la transmission de l'analyse à Celery, une confirmation portant le code HTTP 201 est renvoyée. En cas d'échec, un message d'erreur accompagné du code HTTP 500 est renvoyé. Lors de l'ajout d'une tâche périodique, son succès ou son erreur sont précisés dans le message de confirmation de l'analyse mais en changeant pas le code de confirmation de l'analyse en elle-même. De plus, cette première confirmation signifie simplement que la requête a pu être interprétée et transmise à Celery, le succès ou l'échec de la tâche Celery sera quant à lui transmis par le biais de notifications.

APIS des jeux de données

getDatasetMetadata Cette fonction permet de récupérer les informations relatives à un dataset, comme son titre, description, et champs. cette fonction est appelée depuis une requête GET à l'adresse `/api/dataset/(?P<data_id>+)` et renvoie un format JSON contenant les informations demandées sur le dataset d'identifiant `<data_id>`.

addDataset Cette fonction permet d'ajouter un nouveau dataset aux données d'un senseur existant. En envoyant une requête POST du formulaire présent dans l'onglet *Jeux de données* à l'URL `/api/dataset/add`, l'API va télécharger le fichier et créer l'entrée correspondant au Dataset dans la base de données. La confirmation se fera par l'envoi d'un statut HTTP 201.

getDatasetColumns Cette fonction permet de renvoyer les données contenues dans les colonnes d'un dataset. Elle est utilisée exclusivement depuis l'interface *Visualisation graphique* au moyen d'un appel l'URL `/api/dataset/column` où une requête POST détaille les colonnes de données requises par l'utilisateur. Cette API va alors renvoyer un JSON de format similaire où les valeurs de chaque colonne auront été ajoutées. Ceci permettra aux fonctions d'affichage Javascript de générer les courbes, images ou autres visualisations demandées par l'utilisateur.

editDataset Cette API permet d'éditer directement un Dataset depuis l'onglet associé. Une fois le formulaire correspondant au Dataset validé, un appel POST à l'URL `api/dataset/edit` va mettre à jour les informations du Dataset de la base de données (les fichiers eux-mêmes ne sont pas directement modifiables), et renverra un message de confirmation accompagné du code HTTP 200 en cas de succès.

deleteDataset La suppression d'un dataset est possible au moyen de cette API, passant par un appel à une requête DELETE adressée à l'URL `api/dataset/delete/<id>`. Une fois l'authenticité de l'utilisateur et ses droits sur le dataset validés, le dataset sera supprimé de la base de données et son fichier supprimé. Une amélioration possible permettant d'éviter les erreurs même après confirmation pourrait être de déplacer temporairement ce dataset vers un dossier de *backup* où il pourra être récupéré.

getDatasetList Cette fonction permet d'obtenir la liste des Datasets d'un format choisi, afin de permettre au front-end de générer une liste des datasets pouvant être sélectionnés par l'utilisateur, ce soit pour une visualisation ou une analyse. Pour appeler cette fonction, une requête GET est transmise à l'URL `/api/dataset/-list?dataFormat=<T|V|I|A>`, où la lettre T,V,I ou A spécifie le format de Table, Audio, Vidéo ou Image. La liste des datasets est alors renvoyée au format JSON sous forme d'un tableau de doublets identifiant/titre pour permettre leur sélection.

searchDataset Cette API permet d'effectuer une recherche au sein des Datasets à disposition de l'utilisateur. Elle est accessible depuis la barre de recherche des datasets, au moyen d'une requête POST envoyée à l'adresse `/api/dataset/search?content=<kw1,kw2...>`. L'API va chercher les mots fournis dans `<kw1,kw2>` dans le titre, la description et les mots-clés des datasets et retourner la liste des identifiants des datasets correspondants. Pour les mots clés, une correspondance exacte sera utilisée, cependant la recherche dans le titre et la description seront insensibles à la casse.

APIs des tickets

Depuis la page *tickets*, en validant le formulaire proposé postant les données à l'URL `/api/ticket/add`, la class `addTicket` ajoute un nouveau ticket. Un code HTML 201 est renvoyé pour signifier le succès de cette tâche ; les administrateurs recevront dès lors une notification permettant d'assigner et résoudre ce ticket.

2.4 Front-end : JavaScript

Les fichiers javascript permettent de lancer les fonctionnalités dans le front-end et faire la communication entre le front-end et le serveur, assurer le bon fonctionnement du GUI. Pour l'instant, il y a 6 fichiers javascript personnalisés : *base.js*, *page_analysis.js*, *page_dashboard.js*, *page_dataset.js*, *page_graphics.js* et *page_kubex.js*, qui fournissent les fonctions pour les templates *base.html*, *analysis.html*, *dashboard.html*, *datasets.html*, *graphics.html* et *kubex.html*.

2.4.1 base.js

Dépendances : jquery.min.js, bootstrap.min.js, bootstrap-tagsinput.min.js, typeahead.bundle.min.js.

Variables globales

Ce fichier ne présente aucune variable globale.

Fonctions

function substringMatcher : Reçoit un input de type "string", renvoie une fonction de détection utilisée par librairie *typeaheadjs* pour détecter l'input string, après afficher les résultats dans l'élément html *input#search-input*.

function handleSearch : Obtient les contenus de barre de recherche à gauche de la page, puis redirige à la page *search* avec ces contenus comme paramètre de recherche. S'il n'y a pas contenu dans barre de recherche cette fonction va passer "null" comme paramètre.

function responseMsg : Une fonction globale utilisé pour renvoyer une notification dans front-end quand javascript reçoit la réponse du serveur.

function confirmation : Une fonction globale utilisé pour afficher un bloc de confirmation dans le front-end quand l'utilisateur essaie de supprimer un dataset depuis la page "Tableau de Bord" ou "Jeux de données".

function deleteDataset : Une fonction globale utilisé pour supprimer un dataset dans la page "Tableau de Bord" ou "Jeux de données", qui va envoyer une requête DELETE à l'api *deleteDataset* du serveur (url : *api/dataset/delete/(?P<data_id>d+)*), et une fois supprimé recharge de la page.

function deleteDataset : Une fonction globale utilisé pour supprimer une notification dans page "Notifications", qui va envoyer une requête DELETE à l'api *deleteAlert* du serveur (url : *api/alerts/delete/(?P<alert_id>d+)*).

2.4.2 page_analysis.js

Dépendances : jquery.min.js, bootstrap.min.js, bootstrap-multiselect.js, js.cookie.js.

Variables globales

datasets enregistrement des datasets renvoyés par le serveur

input_infos enregistrement de l'input d'algorithme renvoyé par le serveur

params_infos enregistrement du paramètre d'algorithme renvoyer par le serveur

PERIOD_UNIT enregistrement des valeurs nécessaires pour créer les champs de *periodic_task*

Fonctions

function setDatasetFormat : Une fonction qui contrôle l'affichage et cache du bloc *algoType_container* et bloc *selectBox_container* selon la sélection du type de donnée par l'utilisateur.

function getDatasetList : Une fonction qui fait un appel GET à l'api *getDatasetList* du serveur (url : *api/dataset/list*), passe le format de dataset comme paramètre et obtient une liste de datasets correspondant, incluant le titre et l'id du dataset.

function getData : Une fonction qui fait un appel GET à l'api *getDatasetMetadata* du seueur (url : *api/dataset/(?P<data_id>d+)*), passe l'id de un dataset comme paramètre et obtient l'info du dataset correspondant, incluant le titre, l'id, data et metadata détail du dataset, sans data valeur.

function getSelectedColumnData : Une fonction qui fait un POST appelle à l'api *getDatasetColumns* du seueur (url : *api/dataset/column*), passe le nom de colonne sélectionnée et l'id du dataset correspondant, obtient une liste des valeurs de la colonne sélectionnée.

function getAlgorithmList : Une fonction qui fait un appel GET à l'api *getAlgorithms* du seueur (url : */api/algorithm/list*), passe le format de dataset comme paramètre et obtient une liste d'algorithmes correspondants, chacun inclut le format, le titre, le type et l'id de l'algorithme.

function launchAnalysis : Une fonction qui fait un appel POST à l'api *launchAnalysis* du serveur (url : *api/analysis/launch*), passe un objet d'analyse (inclut le titre d'analyse, le description d'analyse et l'id de l'algorithme sélectionné), un objet d'input (inclut le nom du fichier correspondant, l'information des champs sélectionnés, le format du fichier et un préfixe pour exécuter la commande dans le serveur) et un objet paramètres (inclut un préfixe et une valeur) comme paramètres et lance un algorithme d'analyse dans le serveur.

function block_multiDataset : Une fonction qui reçoit une liste du dataset et génère des checkbox dans le bloc *selectBox_container*.

function block_algoSelector : Une fonction qui reçoit une liste d'algorithmes et génère un menu drop-down dans le bloc *algorithm_container*.

function block_controlPanel : Une fonction qui reçoit un id d'algorithme, appelle la fonction *getAlgoInfo*, selon le resultat génère un formulaire html dans le bloc *controlPanel_container* qui inclut différents éléments html (input, select, checkbox). Pour l'instant, il ne génère que les éléments html pour les datasets de format "table".

function filterAlgo : Une fonction pour montrer et cacher les algorithmes dans bloc *algorithm_container* selon l'option sélectionné par l'utilisateur dans le bloc *select#algoType*.

function input_select_builder : Une fonction pour générer les éléments html des select multiples selon les informations de champ / colonne et nom de fichier passé dans fonction *block_controlPanel*.

function input_select_add : Une fonction pour mettre à jour les options des colonnes dans les select multiples du bloc *options d'analyse*, ajoute des nouvelles colonnes dans les select multiples quand l'utilisateur coche un nouveau dataset checkbox.

function input_select_remove : Une fonction pour mettre à jour les colonnes dans élément les select multiples du bloc *options d'analyse* : enlève les colonnes correspondantes des select multiples quand l'utilisateur décoche un dataset.

function handleSubmit : Une fonction pour collecter tout les data entrées par utilisateur dans le formulaire et les paramètre nécessaires après appelle la fonction *launchAnalysis* pour transmettre la requête au serveur.

function params_input_builder : Une fonction servant à générer un élément html input selon le paramètre qu'il reçoit, utilisé par la fonction *block_controlPanel* quand il construit le bloc *options d'analyse* dans front-end.

function params_select_builder : Une fonction pour générer un élément html select selon le paramètre il reçoit, utilisé par la fonction *block_controlPanel* quand il construit le bloc *options d'analyse* dans front-end. (en cours)

function params_checkbox_builder : Une fonction pour générer un élément html checkbox selon le paramètre il reçoit, utilisé par la fonction "block_controlPanel" quand il construit le bloc *options d'analyse* dans front-end. (en cours)

function generatePeriodInput : Une fonction pour générer un élément html input type "range" selon le valeur de l'élément *select-period-unit* ("Hour", "Day" ou "Minute") sélectionné par utilisateur, qui permet de collecter l'information nécessaire à la création d'une tâche périodique.

function cleanBlock : Une fonction pour enlever tous les contenus dans un bloc html selon le sélecteur jQuery il reçoit

2.4.3 page_graphics.js

Dépendances : jquery.min.js, bootstrap.min.js, bootstrap-multiselect.js, js.cookie.js, echarts.min.js, ecStat.min.js.

Variables globales

datasets enregistrement des datasets renvoyés par le serveur

ALLOWED_DATATYPE_MAP un objet javascript pour faciliter le jugement du format permis des différents datasets

Fonctions

function setDatasetFormat : Une fonction pour contrôler les données affichées dans le bloc *selectBox_container* selon la sélection du format de données par utilisateur.

function getDatasetList : Une fonction qui fait un appel GET à l'api *getDatasetList* du serveur (url : *api/dataset/list*), passe le format de dataset comme paramètre et obtient une liste de datasets correspondant, incluant le titre et l'id du dataset.

function getData : Une fonction qui fait un appel GET à l'api *getDatasetMetadata* du serveur (url : *api/dataset/(?P<data_id>+)*), passe l'id de un dataset comme paramètre et obtient l'information du dataset correspondant, ce qui inclut le titre, l'id, et metadonnées détaillées du dataset, puis génère un bloc html contenant le graphique.

function getSelectedColumnData : Une fonction qui fait un appel POST à l'api *getDatasetColumns* du serveur (url : *api/dataset/column*), passe les informations des colonnes choisies pour l'axe X et l'axe Y (le nom de la colonne, l'id du dataset, et l'index de la colonne), et obtenir les données des colonnes.

function cleanBlock : Une fonction pour enlever tous les contenus dans un bloc html selon le sélecteur jQuery il reçoit.

function block_multiDataset : Une fonction qui reçoit une liste de datasets et génère des checkboxes dans front-end bloc *selectBox_container*.

function create_controlPanel : Une fonction qui reçoit un objet de dataset, selon le FORMAT du dataset ("T" pour la fonction *create_controlPanel_table*, "I" pour la fonction *create_controlPanel_image*, il y manque encore les fonctions pour traiter le type audio et le type video). Elle appelle la fonction correspondante pour filtrer les éléments html du bloc *controlPanel_container*.

function create_controlPanel_table : Une fonction qui reçoit un dataset de format TABLE et génère un bloc html dans le bloc *controlPanel_container*. Pour le moment, le seul type de graphique disponible est le scatterplot : le bloc inclut deux éléments html select, qui représentent l'axe X et l'axe Y. Selon l'information des columns du dataset objet, cette fonction génère les options des select, et un bouton qui permet d'appeler la fonction *generateGraphic* pour générer le graphique selon le choix de l'utilisateur.

function create_controlPanel_image : Une fonction qui reçoit un dataset de format IMAGE et génère un bloc html dans le bloc *controlPanel_container*. Ce bloc inclut un élément html select, qui permet l'utilisateur de choisir dans le dataset une série d'images à présenter dans bloc html *image-slider*. De plus cette fonction génère un bouton qui permet d'appeler la fonction *generateImageSlider* pour générer un carrousel d'images selon le choix d'utilisateur.

function update_controlPanel_add : Une fonction qui reçoit un dataset pour mettre à jour les options des colonnes dans un élément html select ou multi-select du bloc *Options de visualisations*. Selon le FORMAT du dataset, elle appelle la fonction correspondante ("T" pour la fonction *create_controlPanel_table*, "I" pour la fonction *create_controlPanel_image* mais il y manque encore les fonctions pour traiter le format audio ou video), et ajoute les nouveaux datasets dans les select ou multi-select quand l'utilisateur coche un nouvel dataset.

function update_controlPanel_add_table : Une fonction qui reçoit un objet de dataset de format TABLE et met à jour les colonnes des éléments html select ou multi-select du bloc *panneau de contrôle*, ajoute des nouvelles options selon les nouveaux datasets cochés .

function update_controlPanel_add_image : Une fonction qui reçoit un dataset au format IMAGE et met à jour les colonnes dans les éléments html select ou multi-select du bloc *panneau de contrôle*, ajoute des nouvelles options selon les nouveaux datasets cochés.

function update_controlPanel_remove : Une fonction pour mettre à jour les colonnes dans les éléments html multi-select du bloc *panneau de contrôle*. Elle enlève les colonnes correspondants dans les multi-select quand l'utilisateur décoche un dataset.

function generateGraphic : Une fonction qui collecte les informations des colonnes que l'utilisateur a sélectionnées dans le *panneau de contrôle* généré par la fonction *create_controlPanel_table*, et puis fait appel à la fonction *getSelectedColumnData* pour obtenir les data valeurs des colonnes correspondantes, ensuite

traite les données avec fonction *process_data*, puis génère le graphique dans le bloc *graphic_container* avec le fonction *chart_linear_regression*. Pour l'instant le seul graphique possible est le Scatterplot, et les colonnes choisies pour l'axe X et l'axe Y doivent être dans le même dataset.

function generateImageSlider : Une fonction qui collecte les informations des colonnes que l'utilisateur a sélectionnées dans le *panneau de contrôle* généré par la fonction *create_controlPanel_image*, et puis fait appel à la fonction *getSelectedColumnData* pour obtenir les données des colonnes (images) correspondantes, puis génère un bloc html *image-slider* dans le bloc *graphic_container*, qui permet de présenter les images dans un carrousel.

function process_data : Une fonction qui prends deux liste de données (axe X et axe Y), et génère une liste de points.

function chart_linear_regression : Une fonction qui prends une liste de points et une string (titre) pour générer un graphique de type Scatterplot.

2.4.4 page_dashboard.js

Dépendances : jquery.min.js, bootstrap.min.js, bootstrap-multiselect.js, typeahead.bundle.min.js.

Variables globales

Ce fichier ne présente aucune variable globale.

Fonctions

function substringMatcher : Donner un input type *string*, renvoie une fonction de détection utilisé par librairie *typeaheadjs* pour détecter l'input string, après affichier les resultats dans élément html *input#dataset-searchBar*.

function handleSearch : Obtient les contenus de barre de recherche du html input *dataset-searchBar*, et les filtre par les champs que l'utilisateur a choisis (Titre, Description, Mots-clés) , fait un appel au serveur pour rechercher les datasets correspondants, ensuite affiche les résultats dans le bloc *dataset-container*.

function handleSort : Trie les dataset dans le bloc *dataset-container* selon l'ordre choisi par utilisateur (Date, Titre, Kube, Taille). Par défaut les datasets sont triés par Date.

2.4.5 page_kubex.js

Dépendances : jquery.min.js, bootstrap.min.js, js.cookie.js.

Variables globales

ALLOWED_DATATYPE_MAP un objet javascript pour vérifier les différents formats permis des datasets

DATA_FORMAT_MAP un objet javascript pour faciliter le transfert entre les variables et le texte

Fonctions

function editKubeConfiguration : Selon l'événement reçu, récupère le numéro de série du Kubex cible, après fait un appel GET au serveur pour récupérer les informations basiques du Kubex (IP Kubex, IP Serveur et Capteurs), et en puis les afficher dans une fenêtre modale *#kubeConfigModal*

function editSensorConfiguration : Selon le l'événement reçu, récupère le numéro de série du Kubex cible et l'identifiant du capteur, puis fait un appel GET au serveur pour récupérer les informations basiques du capteur cible (titre, IP, et les informations des champs de données), avant de les afficher dans une fenêtre modale *#sensorConfigModal*

function updateSensor : Selon l'événement reçu, les informations basiques du capteur cible (IP, titre, et les informations des champs de données), puis faire un appel POST au serveur, et enfin passer les informations remplies pour mettre à jour la configuration du capteur.

2.5 Outils d'analyse

2.5.1 Scripts R

Les scripts R ont pour principale fonction de l'analyse statistique et le tracé de courbes. Ils requièrent l'installation d'une version fonctionnelle de R ainsi que des bibliothèques utilisées par les scripts.

Pour le moment, un seul script R *lm.R* existe à des fins de test, cependant suivant le même principe de fonctionnement, de nombreuses fonctionnalités peuvent encore être codées.

Régression linéaire

Synopsis

```
Rscript lm.R [-d dimension]
```

Description

Le script *lm.R* prend comme entrée une colonne de valeurs prédites, ainsi que une ou plusieurs colonnes de valeurs d'entrée. Il effectuera alors la régression linéaire des valeurs *prédites* en fonctions des valeurs *d'entrée*, puis renverra la courbe de régression et 4 courbes de statistiques sur la régression.

Améliorations possibles Ce modèle ne prend pour le moment en compte que des formules de la forme $y \sim x_1 + x_2 \dots$. Permettre de spécifier (de façon contrôlée) différents modèles prenant en compte l'interaction et l'intercept sera un atout futur non négligeable.

2.5.2 Scripts Python

Classification par apprentissage

Synopsis

```
python Python/ML_Classification/global_neural_net.py <training> <predict> [-d <dimension>]
```

Description

La classification par apprentissage permet de classer automatiquement un jeu de données de façon similaire à un autre jeu de données comparable déjà classé. Cet algorithme effectuera un apprentissage puis un test sur des échantillons du jeu classé, puis appliquera le réseau de neurones généré pour classer automatiquement le second jeu de données selon les mêmes critères que le premier.