Doc Appli visualisation Espaces verts urbains

**Partie Front**

<https://github.com/ortance/Front_visualisationEV/tree/master/app/base/home>

(environnement ecorelevé) : critères :

* Ville
* Choix de la date (format aaaa-mm-jj)

Views et templates concernés :

leafletView.js et searchView.js, tpl-testMap.html et tpl-searchNDVI.html

ajout de leaflet-draw

**Partie Back :**

<https://github.com/ortance/visualisationEspacesVerts>

Pré-requis :

* **téléchargement** broker rabbitmq (<https://www.rabbitmq.com/#getstarted>)
* **configuration** d’un utilisateur rabbitmq (<http://docs.celeryproject.org/en/latest/getting-started/brokers/rabbitmq.html#broker-rabbitmq>)
* **base de données :** postgreSQL

Lancement de l’application :

* Lancement du broker dans le dossier C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.6.10\sbin avec la commande **rabbitmq-server** ; pour arrêter le serveur : **rabbitmqctl stop**
* Lancement de l’application flask : dans le dossier source, commande **python app.py**
* Lancement du worker celery, après avoir lancé rabbit : dans le même dossier que pour l’application flask, commande **celery worker -A app.celery --loglevel=info** (plus d’informations : <http://docs.celeryproject.org/en/latest/index.html>)

**Setup.py :** pour l’installation de ttes les dépendances

**Config.py :** config de la base de données et des critères de recherche des images sat (sunElevation, cloudCoverage, thresholdNDVI, limitScene)

**App.py :** initialisation de l’application flask et du worker celery et 3 routes pour le fonctionnement de l’appli :

* **getFormData** : récupération des données du front et lancement de la chaine de traitement via une tâche Celery (celery.task.test) (appel models.py)
* **checkceleryTask** : vérification de l’état d’avancement de la tâche celery
* **ndviAuto** : récupération des résultats stockés en base de données selon le nom de la table, récupération des coordonnées du centre de la ZI, mise en forme sous format json

**Models.py :** mise en placeconnexion à la base de données, appels functions.py

**Functions.py :** fonctions pour le traitement d’images satellites :

* main\_script : fonction principale, utilisation des critères de recherche, appel aux fonctions :
  + download\_sentinel2 : téléchargement des bandes des images sentinel2
  + fusion/conversion avec l’outil gdalbuildvrt
  + convert\_coords : conversion des coordonnées de la ZI (WGS84) à la projection de l’image
  + create\_polygon : création d’un polygone pour la découpe avec les coordonnées converties
  + write\_shapefile : création d’un shapefile avec le polygone précédent
  + projection : ajout d’une projection au shapefile
  + découpe des images en .vrt selon le shapefile, réduction du calcul sur la ZI
  + calcuate\_ndvi : calcul du ndvi avec les deux clips
  + binarize : binarisation selon le seuil ndvi choisi
  + polygonisation et insertion en base de données : outil gdal\_polygonize.py
* format\_geojson (pas utilisée pour l’instant)

**API Landviewer** (<https://lv.eosda.com/>) **:**

Images de type 1C (plus d’explications : <https://www.theia-land.fr/fr/projets/muscate>)

Contact pour tout problème sur la clef d’utilisation : [konstantin.nesterenko@eosda.com](mailto:konstantin.nesterenko@eosda.com)

Actuelle page de la documentation de l’API (assez peu fournie) : <https://docs.google.com/document/d/1HwhuZjeEUMhUuMwgdeOZllT0EPg1cgWR4o3IxNvrr-M/pub#h.2f4b9p8ec7kp>