Documentation VAL3D

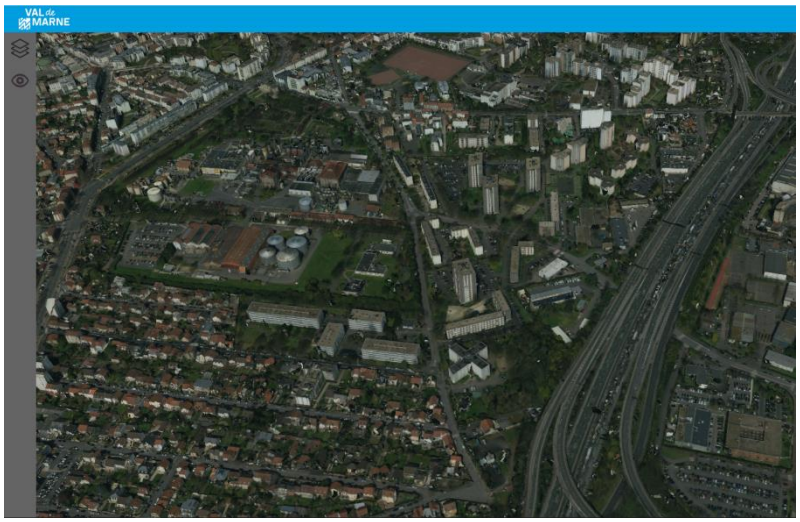
Réalisé par Jules Pierrat – v1.0.0 – 30 Septembre 2021

SOMMAIRE :

1. PRESENTATION :

VAL3D est un projet Open Source de viewer 3D Web permettant la consultation de données géographiques 3D et 2D à l'échelle d'un département ou d'une région. Il est aujourd'hui disponible sur GitHub (voir LIENS UTILES). VAL3D s'installe sur un serveur Web Apache et se configure via une base de données MySQL. Il permet de visualiser des données géographiques 2D vecteur et raster (geojson, geoTiff, WMS, WMTS), 2.5D (OSMBuilding) et 3D (terrain, photomaillage, BIM, glTF, …).

Je détaillerai les différentes étapes d'installation, de configuration et d'utilisation dans la suite de ce document. J'ajouterai également une liste des différentes corrections à apporter sur certains bugs détectés à ce jour ainsi que la liste des fonctionnalités intéressantes à développer selon moi.



2. INSTALLATION :

2.1. **Installation d'un serveur Web Apache et d'une base de données :**

La première étape pour l'installation de Val3D sur votre serveur est de configurer un serveur Web Apache (PHP 7.2 ou plus) ainsi qu'une base de données. Il est conseillé d'accéder à cette base de données via une interface graphique comme phpMyAdmin pour la suite. Les modules suivants sont conseillés d'être installés :

apache2handler	*curl*	*gmp*	*mbstring*	*Phar*	*sqlite3*	*xmlwriter*
bcmath	*date*	*hash*	*mysqli*	*readline*	*standard*	*xsl*
bz2	*dom*	*iconv*	*mysqlnd*	*Reflection*	*tokenizer*	*Zend*
calendar	*exif*	*imap*	*openssl*	*session*	*wddx*	*OPcache*
com_dotnet	*fileinfo*	*intl*	*pcre*	*SimpleXML*	*xdebug*	*zip*
Core	*filter*	*json*	*PDO*	*soap*	*xml*	*zlib*
ctype	*gd*	*ldap*	*pdo_mysql*	*sockets*	*xmlreader*	
	gettext	*libxml*	*pdo_sqlite*	*SPL*	*xmlrpc*	

Note : Tous ces modules ne sont pas nécessaires, mais étaient installés lors du développement de VAL3D

2.2. **Ajout du projet VAL3D dans le répertoire www :**

Une fois le serveur Web configuré, on peut ajouter le contenu du répertoire GitHub à la racine du serveur Web (généralement /var/www/html pour un serveur LAMP). Plusieurs choses sont alors à configurer.

Le fichier **PATH.txt** doit contenir le chemin d'accès à la racine du serveur Web (pour le serveur LAMP /var/www/html).

Le fichier **conf/clef_geoportail.txt** doit contenir la clef géoportail pour accéder aux différentes données. Pour le moment, le fichier peut être lu par n'importe quel client, ce qui pose un souci de sécurité. Dans une prochaine version de VAL3D ce problème sera résolu en déplaçant cette clef dans la base de données ou en encryptant le contenu du fichier.

Le fichier **conf/connectBDD.txt** doit contenir l'ensemble des informations de connexion à la base de données :

host=localhost
username=root
password=root
bd=val3d

Pour le moment, ce fichier est lisible par n'importe quel client ce qui pose également des soucis de sécurité. Le fichier sera encrypté dans une future version de VAL3D pour résoudre ce problème. Pour le moment cet utilisateur n'aura que des droits limités sur la base de données ce qui réduit les problèmes de sécurité.

Note : Dans le cas où l'utilisateur renseigné possèderait des droits trop poussés sur la BDD, cela pourrait poser quelques soucis. Une solution est de rendre la base de données accessible seulement depuis localhost.

2.3. **Création de la base de données :**

Dans l'interface phpMyAdmin, il faut créer une nouvelle base de données (par exemple val3d) et y importer le fichier SQL présent dans le répertoire bd (/var/www/html/bd/val3d.sql pour un serveur LAMP). On retrouve alors les tables suivantes au sein de cette base de données :

layer
plugin
users
poi
bim

Dans un deuxième temps, il est nécessaire de créer un nouvel utilisateur qui peut accéder à cette base de données depuis localhost (exemple : VAL3D_client@localhost) mais qui ne peut exécuter que la commande « SELECT ». C'est cet utilisateur qu'il faut renseigner dans le fichier de configuration conf/connectBDD.txt vu à la partie 2.2.

Note : Il est important de vérifier l'encryptage de la base de données car cela peut provoquer des erreurs si elle est remplie avec certains caractères comme les accents.

2.4 **Gestion du répertoire data :**

Le répertoire data (/var/www/html/data) permet de stocker toutes les données géographiques qui ne sont pas stockées sur un serveur extérieur. Dans le cas où le volume de données à stocker est très important, il est conseillé de déplacer le répertoire data sur une partition permettant de stocker un tel volume. Il suffit alors de créer un lien symbolique dans le répertoire souche du serveur web qui pointe sur le nouveau répertoire data.

Une fois toutes ces étapes réalisées, VAL3D est fonctionnel dans un navigateur Web. On peut retrouver la liste de tous les outils sur la gauche mais aucune donnée géographique n'a encore été ajoutée. Nous verrons ce point dans la partie configuration (3).

3. CONFIGURATION :

Dans cette partie nous allons voir comment ajouter manuellement des données géographiques et les configurer. Nous verrons également comment gérer les différents utilisateurs et leurs droits d'accès. Nous verrons comment ajouter ou supprimer des plugins et quels utilisateurs y auront accès. Nous verrons comment ajouter des Points of Interest (POI) et les configurer. Enfin nous verrons comment paramétrer un projet BIM pour l'afficher dans VAL3D.

Note : Aujourd'hui, la configuration se fait manuellement et nécessite quelques connaissances en json et la gestion de base de données. Dans une future version de VAL3D, cette configuration manuelle compliquée sera remplacée par une interface administrateur qui s'occupera de gérer l'ensemble de ces tâches de manière simplifiée.

3.1 Données géographiques :

3.1.1 Structure de la table layer

La base de données contient une table layer qui permet de décrire l'ensemble des données géographiques qu'un utilisateur peut visualiser. Pour le moment, tous les utilisateurs ont accès à toutes les couches de données. Dans une future version de l'application, il serait intéressant de gérer les accès à certaines couches en fonction des privilèges de l'utilisateur.

Voici comment s'organise cette table :

Attributs	*Type*	Définition
ID	*int*	Identifiant numérique unique de la couche Il permet de trier l'ordre d'affichage des couches dans le menu couche de VAL3D Ex : 1
Name	*char*	Nom de la couche Ce nom est affiché dans le menu des couches de VAL3D, il permet de faciliter la compréhension de la nature de cette couche par l'utilisateur. Ex : Val-de-Marne 3D
Path	*char*	Chemin d'accès à la donnée Permet de définir où VAL3D récupère les données de la couche (adresse http, adresse relative VAL3D, …) Ex : data/Photomaillage/tileset.json
Type	*char / list*	Type de la donnée à ajouter Mot clef qui permet à VAL3D de faire les bons traitements en fonction de la nature de la couche. Ce mot clef fait parti d'une liste bien définie décrite dans la suite de ce document. Ex : 3dtiles
Display	*boolean*	Boolean d'affichage par défaut Définit si la couche est visible par défaut dans le viewer VAL3D. Valeurs : true, false
Groupe	*char*	Nom du groupe auquel appartient la couche Ce nom de groupe permet de trier les types de couches par groupe. Si deux couches appartiennent à la même catégorie, il faut alors renseigner le même nom de groupe. Ex : Plan
Meta	*json*	Métadonnées pour configurer en détail la couche Elles sont renseignées au format json et spécifiques à chaque type de couche. Cet attribut est détaillé dans la suite de cette section pour chaque type de donnée. Ex : {''WMS'' :{},''style'' :{}}

3.1.2 Photomaillage / 3dtiles :

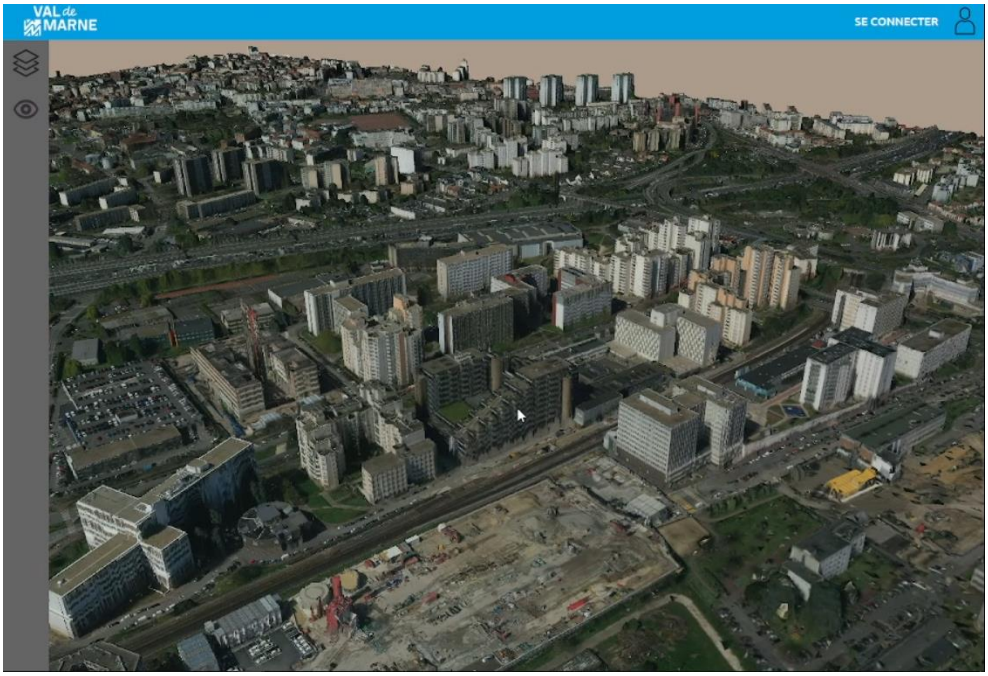
Dans cette partie, nous allons voir comment ajouter et configurer un photomaillage. Un photomaillage est un fichier au format 3dtiles. Cela signifie qu'il est structuré comme suit :

- **Un répertoire Data** contenant l'ensemble des tuiles au format .b3dm
- **Un fichier tileset.json** qui organise ces tuiles les unes par rapports aux autres.

Ce fichier 3dtiles peut être stocké dans le répertoire data comme vu dans la section Installation mais également sur un serveur extérieur.

Pour ajouter ce 3dtiles dans VAL3D, il suffit de remplir les champs de **la table layer** comme vu précédemment. Le type à renseigner est **3dtiles**. Il est inutile de renseigner l'attribut **Meta** car le photomaillage n'est pas paramétrable. Voici un exemple de photomaillage stocké sur le serveur web dans un répertoire Photomaillage/VM3D :

ID	Name	Path	Type	Display	Groupe	Meta
1	Val-de-Marne 3D	data/Photomaillage/VM3D/tileset.json	3dtiles	true	Cartes 3D	



3.1.3 Couche vectorielle 2D surfacique

Dans cette partie nous verrons comment ajouter et configurer une couche vectorielle 2D surfacique. Ce fichier doit forcément être au format **Geojson surfaces** et les attributs de cette couche doivent être nommés comme ils apparaîtront dans VAL3D. Ce fichier Geojson doit être stocké sur le serveur web ou sur un serveur distant.

Pour ajouter ce Geojson dans VAL3D, il suffit de remplir les champs de la table **layer** comme vu précédemment. Le type à renseigner est **vectS**. L'attribut **Meta** est au format json et il permet dans ce cas de paramétrer le style par défaut de la couche. Dans cette version de VAL3D, seule la couleur par défaut et l'opacité par défaut sont réglables. Dans une future version, Il serait intéressant de pouvoir donner un style attributaire par défaut ou même une étiquette.

PARAMETRAGE META:

```
{
    "style" :
    {
        "type": "uniq",
        "fill_color" :
        {
            "red" : 0,              // 0-255
            "green" : 255,          // 0-255
            "blue" : 0 ,            // 0-255
            "alpha" : 0.5           // 0-1
        }
    }
}
```

Voici un exemple de Geojson stocké sur le serveur web dans un répertoire Administratif et avec un style par défaut définit comme couleur verte et opacité 50% :

ID	Name	Path	Type	Display	Groupe	Meta
2	Forêts	data/Environnement/forets.geojson	vectS	true	Environnement	*comme ci-dessus

3.1.3 Couche vectorielle 2D contours

Dans cette partie nous verrons comment ajouter et configurer une couche vectorielle 2D surfacique. Ce fichier doit forcément être au format **Geojson surfaces** et les attributs de cette couche doivent être nommé comme ils apparaîtront dans VAL3D. Ce fichier Geojson doit être stocké sur le serveur web ou sur un serveur distant.

Pour ajouter ce Geojson dans VAL3D, il suffit de remplir les champs de la table **layer** comme vu précédemment. Le type à renseigner est **vectL**. L'attribut **Meta** est au format json et il permet dans ce cas de paramétrer le style par défaut de la couche. Dans cette version de VAL3D, seule la couleur par défaut, l'opacité par défaut et l'épaisseur par défaut sont réglable. Dans une future version, Il serait intéressant de pouvoir donner un style attributaire par défaut ou même une étiquette.

PARAMETRAGE META:

```
{
    "style" :
    {
        "type": "uniq",
        "stroke_color" :
        {
            "red" : 255,          // 0-255
            "green" : 0,          // 0-255
            "blue" : 255,         // 0-255
            "alpha" : 0.8         // 0-1
        },
        "stroke_width" : 2
    }
}
```

Voici un exemple de Geojson stocké sur le serveur web dans un répertoire Environnement et avec un style par défaut définit comme couleur magenta, opacité 80% et une épaisseur de 2 pixels :

ID	Name	Path	Type	Display	Groupe	Meta
2	Forêts	data/Environnement/forets.geojson	vectL	true	Environnement	*comme ci-dessus

Note : Il est important de noter que les couches vectorielles surfaces et les couches vectorielles contours sont toutes les deux construites sur la base de fichier Geojson contenant des surfaces. Césium par défaut ne permet pas d'affecter un remplissage et un contour à une surface ce qui explique la séparation entre les deux actuellement. Il sera très utile de fusionner les deux dans une future version de VAL3D afin de simplifier la gestion des couches.
/ ! \ Attention pour le développement futur, Cesium additionne les couleurs lorsque des couches de différentes couleurs se superposent. A prendre en compte.

3.1.4 Couche vectorielle 2D Polylignes *:*

Pour le moment, il est impossible d'ajouter des couches vectorielles linéaires dans VAL3D. Cesium ne lit pas automatiquement les fichiers Geojson construits sur le modèle linéaire mais l'on pourra s'inspirer du plugin poi.js pour faire les traitements manuellement et ajouter des couches linéaires au format Geojson.

3.1.5 Couche vectorielle 2.5D Volume :

Pour le moment, il est impossible d'ajouter des volumes à partir de Geojson surface et d'un attribut hauteur. Cette fonctionnalité a été entièrement codée et est commentée dans le script VAL3DLayer.js. Cependant, il existe de nombreux cas où cette fonctionnalité peut faire crasher l'application et elles ne sont encore pas toute résolues. Cependant une grande partie du travail a quand même été réalisée.

3.1.6 OpenStreetMap Imagery

BUG SUR LE SERVEUR VAL3D – à venir

3.1.7 Geovaldemarne WMS

BUG SUR LE SERVEUR VAL3D – à venir

3.1.8 Géoportail Imagery :

Dans cette partie, nous allons voir comment ajouter des flux WMTS Géoportail à notre viewer Cesium. Il faut évidemment posséder une clef Géoportail. Pour ajouter ce flux WMTS à VAL3D, il est nécessaire de bien paramétrer les attributs **Path**, **Type** et **Meta** dans la table **layer.**

Pour l'attribut **Path**, il faut renseigner le lien d'accès aux géoservices de l'IGN en y ajoutant la clef Géoportail comme suit :

http://wxs.ign.fr/MACLEFGEOPORTAIL/geoportail/wmts

Comme précisé dans la section installation, il sera nécessaire de revoir le stockage de cette clef si son utilisation est réglementée.

Pour l'attribut **Type,** il suffit de rentrer **imageryGeoP.**

Pour l'attribut **Meta,** il permet de paramétrer quel type de couche on veut afficher (**WMTS**) et le style par défaut de cette couche (**style**).

PARAMETRAGE META :

```
{
    "style" :
    {
        "opacity" : 1                   // 0-1  défaut : 1
    },
    "WMTS" :
    {
        "layer"   : "CASSINI3",         // Voir géoservices IGN       défaut : ORTHOIMAGERY.ORTHOPHOTOS
        "style" :  "normal",            // Voir géoservices IGN       défaut : normal
        "format" :  "image/jpeg",       // Voir géoservices IGN       défaut : image/jpeg
        "tileMatrixSetID" :  "PM"       // Voir géoservices IGN       défaut : PM
    }
}
```

Voici un exemple de comment ajouter une couche WMTS Plan Cassini du Géoportail avec une opacité de 80% par défaut :

ID	Name	Path	Type	Display	Groupe	Meta
3	Cassini	*http://wxs.ign.fr/MACLEFGEOPORTAIL/geoportail/wmts*	imageryGeoP	false	Plan	*comme ci-dessus

3.1.9 OSM Building

Pour ajouter OSM Building, il vous suffit d'ajouter un élément dans la table **layer** et renseigner l'attribut **Type** avec le mot-clef **buildingOSM.** Cette couche n'est pas paramétrable. Exemple :

ID	Name	Path	Type	Display	Groupe	Meta
4	Bâtiment OSM		buildingOSM	false	Bâtiments	

3.1.10 Skybox

La skybox est l'image de fond derrière le globe Cesium. Il suffit de fournir le path d'une image au format jpg carré et cohérente bord à bord. Une future version de VAL3D pourrait permettre de choisir 6 images différentes et de proposer un rendu plus intéressant. Exemple :

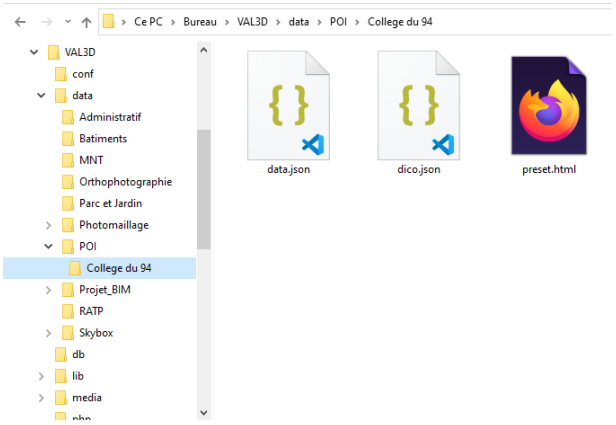
ID	Name	Path	Type	Display	Groupe	Meta
5	Ciel bleu	*http://unsiteweb/img/bluesky.jpg*	skybox	true	Ciel	

3.1.11 Point of Interest (POI) :

Les points of Interest ou POI sont la meilleure manière de décrire une donnée géographique ponctuelle. L'administrateur de VAL3D pourra personnaliser l'apparence, la position ou même la fiche de description d'un élément. Le principe est de cliquer sur un icone géoréférencé dans VAL3D et qu'il ouvre une fenêtre modale au sein de laquelle une page web peut s'afficher.

Un POI est donc un ensemble de trois fichiers au nom bien définit au sein d'un même répertoire comme ci-dessous :

- Un fichier **data.json** au format geojson ponctuel et qui décrit la géométrie et les attributs des POI.
- Un fichier **preset.html** qui décrit la page web à afficher dans la fenêtre modale. Elle décrit à la fois le style et la structure de la page. On peut y insérer des références aux attributs du fichier data.json.
- Un fichier **dico.json** qui permet de faire le lien entre les attributs du fichier geojson et certains textes du fichier html.



Exemple : Supposons qu'un fichier geojson ponctuel possède un attribut **nom** et **image** et que la page template.html possède une balise ** [NOM DE L'ETALISSEMENT] ** et une balise **<img src= '[PATH OF IMAGE]'>**. Alors, il faudra configurer le fichier **dico.json** de la sorte :

```
{
    "[NOM_ETABLISSEMENT]": "nom",
    "[PATH_OF_IMAGE]": "image"
}
```

Maintenant, si l'administrateur veut ajouter ses POI à VAL3D, il doit renseigner la base de données. Cette fois ci, il doit remplir la table **poi** et non la table layer. Elle est organisée comme suit :

Attribut	*Type*	Définition
id	*int*	Identifiant numérique unique de la couche de POI Il permet de trier l'ordre d'affichage des POI dans le menu POI de VAL3D Ex : 1
Nom	*char*	Nom de la couche POI Ce nom est affiché dans le menu des POI de VAL3D, il permet de faciliter la compréhension de la nature de cette couche par l'utilisateur. Ex : Collèges du 94
Path	*char*	Chemin d'accès à la donnée Permet de définir où VAL3D récupère les données du POI. Cela correspond au répertoire parents des trois fichiers décrits précédemment. Ex : data/POI/College du 94 (Par exemple pour le cas ci-dessus)
Meta	*json*	Métadonnées pour configurer en détail la couche Elles sont renseignées au format json et permettent de gérer l'icône de chaque POI et son emplacement par rapport au terrain. Cet attribut sera juste à la suite de ce tableau.

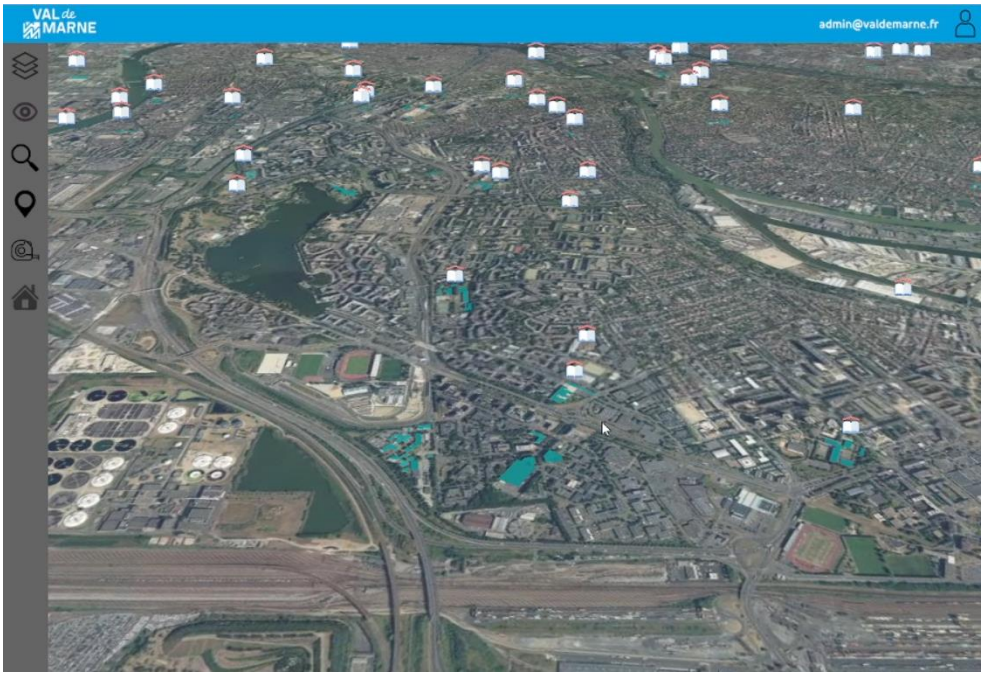
Pour configurer en détail l'apparence des POI dans le monde VAL3D en 3D, il est donc nécessaire de remplir correctement l'attribut Meta au format json :

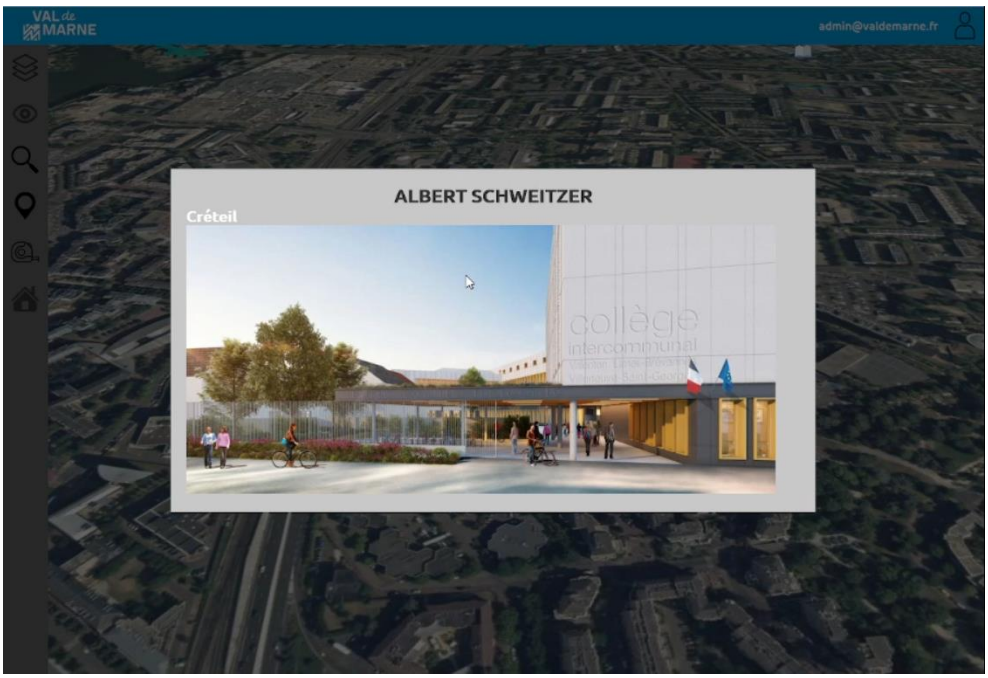
Il est composé de 3 sous-parties, **style**, **height** et **height_field**. Sachant que height et height_field ne sont pas compatibles. **style** est composé de **image** et **image_field** qui ne sont pas compatibles ainsi que **image_scale.**

```
{
"style": {                          // Contrôle en détail l'icone du POI (optionnel)
    "image": "media/img/logo.png",      //lien vers une image unique pour tout le groupe (prioritaire sur image_field)
    "image_field": "IMAGE",             //Attribut du geojson définissant le lien vers une image
    "image_scale": 1                    //Facteur d'échelle de l'image sur l'écran (0 - inf | défaut: 1)
},
"heigth": 200,                      // Définit la hauteur ellipsoïdale du POI (prioritaire sur height_field | optionnel | Défaut: 200)
"height_field": "ALTITUDE"          // Définit le champ du geojson qui définit la hauteur ellipsoïdale (optionnel)
}
```

Note : Il sera intéressant de pouvoir modifier la taille des icônes en fonction d'un champ de la couche geojson. Quelques bugs à corriger pour les attributs ***image_field*** *et* ***height_field.***

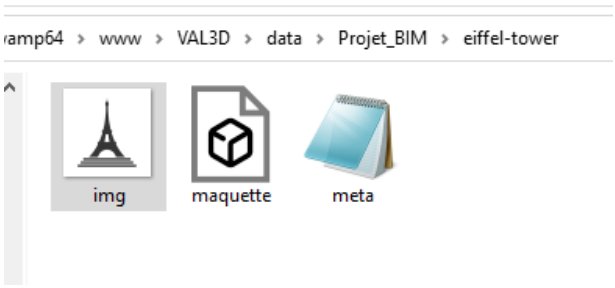
Une fois l'ensemble des attributs de la table renseignée, la couche de POI apparaît dans le menu POI.





3.1.12 Projet BIM

VAL3D permet aujourd'hui d'importer au sein de son monde 3D des maquettes BIM. Le fichier de partage du BIM est l'IFC (.ifc) mais pour des raisons d'optimisation, il est plus intéressant d'utiliser le format glTF. Pour le moment VAL3D possède des logiciels opensources permettant de réaliser cette conversion mais elle doit être faite manuellement. Une fois la maquette BIM convertie en glTF, il faut la placer dans son propre répertoire data avec en plus un fichier **meta.json** et une image carrée **img.jpg** comme illustré ci-contre. Ce fichier permet de géoréférencer la maquette BIM. Il doit être rempli manuellement car il n'existe pas de plugin et d'interface graphique pour générer ce fichier à ce jour.




```
{
    "longitude": 2.2933081,
    "latitude": 48.8582491,
    "height": 85.368,
    "scale":1,
    "rotation":225
}
```

Le fichier **meta.json** est configuré comme ci-dessous :

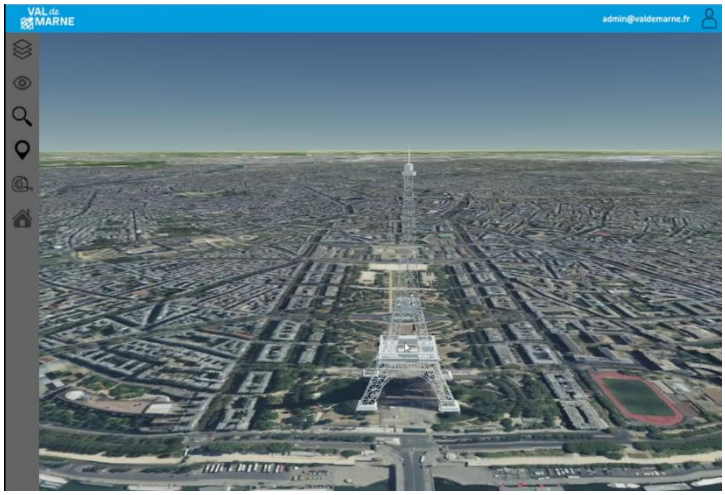
logitude : Longitude (°) en WGS84 (coordonnée de l'origine de la maquette)
latitude : Latitude (°) en WGS84 (coordonnée de l'origine de la maquette)
height : Hauteur ellipsoïdale (AIG GRS80)
scale : Echelle (Généralement 1)
rotation : Angle (°) de rotation entre l'axe x de la maquette et l'axe tangent à l'ellipsoïde dirigé vers le nord

L'image **img.jpg** doit être carrée et placée dans le même répertoire que les fichiers précédents.

Pour ajouter la maquette dans VAL3D, il faut remplir la table **bim** de la base de données. Elle est structurée de cette façon :

Attribut	*Type*	Définition
Id	*Int*	Identifiant numérique unique des projets BIM. Il permet de trier l'ordre d'affichage des projets BIM dans le menu BIM. Ex : 1
Name	*char*	Nom du projet BIM Ce nom est affiché dans le menu des projets BIM de VAL3D, il permet de faciliter la compréhension de la nature du projet par l'utilisateur. Ex : Gymnase Marc Valentin
Description	*char*	Description du projet BIM Cette description est libre, elle permet de donner plus de détails à l'utilisateur sur la nature du projet (adresse, architecte, prix, etc.) Ex : Adresse : X rue Xxxxxxx 94000 Creteil Architecte : ------
Path	*char*	Chemin d'accès à la donnée Permet de définir où VAL3D récupère les projets BIM. Cela correspond au répertoire parent des trois fichiers décrits précédemment. Ex : data/Projet_BIM/eiffel-tower (Par exemple pour le cas ci-dessus)

Une fois l'ensemble de ces champs complétés, le projet BIM est disponible dans le menu projet BIM de VAL3D.



3.2 Ajout, Suppression et modification des profils utilisateurs :

L'administrateur de VAL3D a la possibilité de créer plusieurs profils utilisateurs dans VAL3D. Cela lui permet de donner accès ou non à certains plugins.

Note : Il sera très intéressant de pousser ce profil utilisateur pour l'accès aux données couches, POI et BIM afin que certaines informations ne soient réservées qu'à certains utilisateurs.

La gestion des profils utilisateurs se fait à l'intérieur de la base de données, dans la table **user.** Elle est structurée comme ci-dessous :

Attributs	*Type*	Définition
ID	*int*	Identifiant numérique unique à chaque utilisateur. Il permet d'organiser les utilisateurs pour leur gestion. Ex : 1
email	*char*	Email de l'utilisateur. L'utilisateur se connecte à Val3D grâce à cet identifiant. Pour le moment, il peut être factice. Ex : admin@valdemarne.fr
password	*char*	Mot de passe de l'utilisateur. L'utilisateur se connecte à son profil grâce à ce mot de passe. Pour le moment il est écrit en clair dans la base de données pour le moment. Ex: motdepasse
class	*char*	Groupe de l'utilisateur Les utilisateurs sont regroupés par groupes afin de gérer plus facilement leurs accès. Le mot BASIC est réservé pour l'utilisateur de base qui n'a pas de compte. Ex : BIM_MANAGEUR

Pour se connecter, le client possède en haut à droite de l'interface un bouton SE CONNECTER. En remplissant le formulaire avec les données nécessaires, la page est rechargée avec les plugins correspondant aux droits de l'utilisateur.

3.3 Gestion des plugins :

VAL3D est organisé autour de plugins qui sont modulables. La gestion de ces plugins se fait également depuis la base de données VAL3D grâce à la table plugin structurée comme ci-dessous :

Attribut	*Type*	Définition
Id	*int*	Identifiant numérique unique à chaque plugin. Il permet d'organiser l'ordre dans lequel les plugins apparaissent dans le menu de gauche. Ex : 1
Name	*char*	Nom du plugin. C'est le nom qui apparaît en haut du menu (Menu + …) Ex : Mesures
Access	*char*	Liste des groupes d'utilisateurs ayant accès au plugin C'est là que le groupe associé à un utilisateur est utilisé. Chaque groupe est séparé d'un point-virgule. Ex: BASIC;BIM_MANAGER;ADMIN;
Path	*char*	Chemin d'accès au script JS Permet de définir où VAL3D récupère le script du plugin. Cela correspond à l'adresse du script js (généralement dans le répertoire scripts) Ex : scripts/mesures.js
Code	*char*	Le code du plugin (unique) Il est fourni par le développeur du script et permet de faire certains liens entre l'interface VAL3D et les outils créés par ce script. Ex : mesures

Note : Un script « template.js » est disponible dans le répertoire script. Il correspond à la structure de base d'un script pour l'intégrer. Il nécessite cependant des connaissance JS et de la librairie CesiumJs pour le compléter.

4. UTILISATION :

A COMPLETER

5. CORRECTIONS A APPORTER

Bug / Problème	**Correction**
Accès à la base de données non sécurisée, information de connexion en clair sur le réseau.	Encryptage des informations de connexion et/ou limité l'accès aux informations.
OSM, Geovaldemarne	Correction des bugs
Bug emprise BIM,	Correction de bugs
Information accessible à tous	Rendre les couches, Poi et BIM dépendant de l'utilisateur
Couches 2.5D vectorielles	Dé commenté et correction de bug

6. FONCTIONNALITES A AJOUTER

Difficultés	**Solution**
Configuration de la base de données.	Création d'un plugin administrateur et d'une interface de gestion de la base de données.
Gestion difficile des couches vectorielles surfaciques (remplissage et contours)	Fusion des données vectS et vectL
Pas possible d'ajouter des couches vectorielles linéaires	Ajout des couches geojson linéaire.
Menu mesure incomplet	Ajout des mesures de distances et surface
Poi, taille des icones fixe	Ajout d'un champ dans la BDD
Surface d'un projet BIM non paramétrable	Modifier le script pour la rendre paramétrable grâce à une surface
Gestion de vue	Plugin pour gérer des vues préconfigurer (Dep, communes, lieux, etc.)

POUR PLUS D'INFO :

Jean-Michel DZIUBICH
Chef de projet
jean-michel.dziubich@valdemarne.fr

Jules PIERRAT
Développeur v1.0.0
jules.pierrat.eirl@gmail.com

LIENS UTILES :

GitHub :
https://xxxxxxxxxxxxxxxxxxxxxxxx