

# Web Mapping : Documentation programmation

Février 2022

Auteurs :

REVENEAU Hugo



## Contents

I.	Organisation des fichiers du projet.....	3
1.	Dossier CSS .....	3
2.	Dossier HTML .....	3
3.	Dossier IMG.....	3
4.	Dossier JS .....	3
5.	Dossier PHP .....	4
II.	Description des codes principaux .....	4
1.	Avant-propos : description des connaissances préalables.....	4
2.	main_js.js.....	5
3.	insert_in_db.php .....	7
4.	index.html .....	7
	Figure 1 : représentation UML de la classe Visualisateur .....	5
	Figure 2 : Schéma de structure de la page web .....	8

## I. Organisation des fichiers du projet

Avant de lire la présente documentation, il est conseillé d'avoir lu la documentation utilisateur ou d'avoir testé le site. En effet certains aspects tels que les différentes commandes sont supposés connus.

Le projet contient tous les fichiers nécessaires au fonctionnement de la page web et des échanges avec la base de données. Les fichiers sont répartis dans des dossiers correspondant à leurs types. Ainsi on note les dossiers suivants :

- CSS
- HTML
- IMG
- JS
- PHP

En racine du projet se trouve également un fichier ***index.html*** qui sera celui lancé lors de l'appel de l'URL. Il s'agit d'une page web basique appelant le fichier ***redirection.js*** qui va se charger de lancer la page principale qui se trouve dans le dossier **HTML > *index.html***. Ce mécanisme permet alors de bien structurer le code notamment si plusieurs pages HTML avait été présentes.

### 1. Dossier CSS

Il contient le fichier ***style.css***. C'est ce fichier qui implémente le style de la page. Il gère les couleurs, les animations des boutons mais également la mise en page des différentes divisions.

### 2. Dossier HTML


Il contient le fichier ***index.html***. C'est le fichier qui contient le contenu organisé de la page web. Il appelle également les différents fichiers JavaScript nécessaires au fonctionnement de la page.

### 3. Dossier IMG

Il contient les images utiles au projet. Ainsi il contient le logo de la page (***logo.png***) et un QR code permettant d'accéder directement à la page web.

### 4. Dossier JS

Il contient tous les fichiers JavaScript lancés par la page HTML principale. On en compte trois :

- ***aide.js*** contient l'EventListener affichant un popup d'aide en cas de clic sur le bouton d'aide 
- ***check\_unicity.js*** contient le code vérifiant si le nom saisi dans le formulaire existe déjà ou non dans la base de données. Il gère également l'activation ou non du bouton d'envoi.
- ***main.js.js*** est le fichier javascript principal. Il contient la classe Visualisateur qui contient l'intégralité des méthodes de modification de la vue, mais également de chargement depuis le flux WMTS et d'envoi dans la base de données du projet.

Les deux derniers fichiers évoqués échangent en relation étroite avec le Back-end par le biais de fetchs appelant en asynchrone des fichiers PHP.

## 5. Dossier PHP

Il contient tous les fichiers PHP utilisés par les fichiers **check\_unicity.js** et **main\_js.js**. On en compte également trois :

- **db\_link.php** est le fichier permettant la connexion à la base de données du projet hébergée sur les serveurs de AlwaysData.
- **check\_unicity.php** est le fichier contenant la requête SQL vérifiant l'existence ou non du nom en input dans la base de données. Pour cela, il compte le nombre de noms égaux à l'input et renvoie ce nombre en sortie. Ce nombre est ensuite interprété par le fichier **check\_unicity.js** qui se charge alors de bloquer ou non le bouton d'envoi.
- **insert\_in\_db.php** est le fichier contenant la requête SQL qui va remplir un enregistrement dans la base de données.

## II. Description des codes principaux

### 1. Avant-propos : description des connaissances préalables

Premièrement, cette page web se base sur l'utilisation d'un flux WMTS afin de récupérer des images directement depuis les serveurs du Géoportail. On peut accéder à ces images par une requête https via une adresse URL. Un exemple est présent ci-dessous :

URL :

<https://wxs.ign.fr/pratique/geoportail/wmts?SERVICE=WMTS&REQUEST=GetTile&VERSION=1.0.0&LAYER=ORTHOIMAGERY.ORTHOPHOTOS&TILEMATRIXSET=PM&TILEMATRIX=6&TILECOL=35&TILEROW=20&STYLE=normal&FORMAT=image/jpeg>

Résultat :



Cette requête se compose de trois parties :

- **https://wxs.ign.fr/pratique/geoportail/** réfère à l'adresse des serveurs et contient la clef d'accès. Nous utilisons ici la clef **pratique**.
- **wmts?SERVICE=WMTS&REQUEST=GetTile&VERSION=1.0.0** indique que l'on souhaite récupérer une tuile du service WMTS

- **LAYER=ORTHOIMAGERY.ORTHOPHOTOS&TILEMATRIXSET=PM&TILEMATRIX=6&TILECOL=35&TILEROW=20&STYLE=normal&FORMAT=image/jpeg** est une suite de paramètres, séparés par des **&** et décrivant les caractéristiques souhaitées pour la tuile demandée. On trouve alors :
  1. **LAYER** : le type de fond de carte demandé
  2. **TILEMATRIXSET** : le type de tuilage voulu
  3. **TILEMATRIX** : le niveau de zoom, entier, allant de 0 à un entier variable en fonction de la localisation de la tuile
  4. **TILECOL** : la colonne dans la grille de tuilage (allant de 0 à  $2^{TILEMATRIX-1}$ )
  5. **TILEROW** : la ligne dans la grille de tuilage (allant de 0 à  $2^{TILEMATRIX-1}$ )
  6. **STYLE** : le style voulu pour la tuile (restera sur normal pour ce projet)
  7. **FORMAT** : le format d'image voulu pour la tuile

## 2. main\_js.js

Ce code se divise en trois parties distinctes. La première est le codage de la fonction de filtrage des noms, qui permet de normaliser la saisie de l'utilisateur (ligne 1 – 15).

La seconde partie est le codage de la classe principale **Visualisateur**. Sa description UML est montrée en Figure 1 :

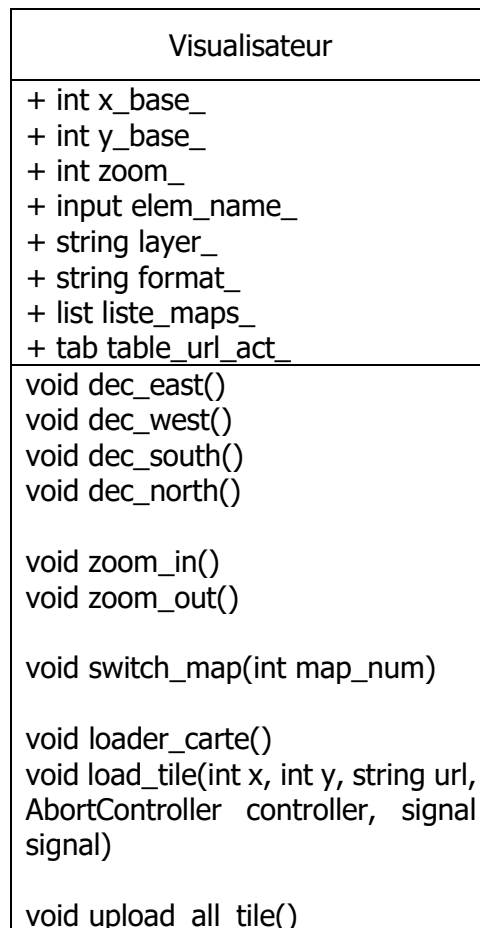


Figure 1 : représentation UML de la classe Visualisateur

***x\_base\_*** et ***y\_base\_*** sont les coordonnées de la tuile haut gauche à afficher par défaut. ***zoom\_*** est le niveau de zoom initial. Ces trois paramètres sont déclarés comme valant 0, 0 et 2, générant ainsi par défaut une carte mondiale complète. L'attribut ***elem\_name\_*** désigne l'input HTML de saisie du nom de jeu. Il est obtenu en récupérant l'élément par l'id ***nom\_jeu\_*** (cf ligne 232).

L'attribut ***liste\_maps\_*** est défini directement dans le constructeur de la classe. Elle contient une liste de dictionnaires. Un dictionnaire correspond à une carte et contient le nom de son ***LAYER*** d'URL sous la clef ***layer*** et le nom de son ***FORMAT*** d'URL sous la clef ***format***.

On définit aussi dans le constructeur le tableau associatif ***table\_url\_act\_***. Ce tableau a pour objectif de conserver en quasi-temps-réel les url des images affichées dans le navigateur. On associe alors à la clef ***X\_Y***, avec ***X*** la ligne et ***Y*** la colonne de l'image dans le navigateur, l'URL de l'image chargée. On actualise alors ce tableau associatif à chaque chargement d'image. Ce sont les URL stockées dans ce tableau qui seront envoyées dans la base de données lorsque le bouton d'envoi sera enclenché.

Les trois méthodes ***dec\_east()***, ***dec\_west()***, ***dec\_south()*** et ***dec\_north()*** permettent de décaler la vue affichée dans le navigateur. Pour cela, elles modifient les attributs ***x\_base\_*** et ***y\_base\_***, en faisant évidemment attention à ne pas dépasser les bords de l'image. On enclenche ensuite une actualisation de la carte par l'utilisation de la méthode ***loader\_carte()*** qui sera présentée plus tard.

Les méthodes ***zoom\_in()*** et ***zoom\_out()*** ont pour objectif de modifier l'attribut ***zoom\_***, de recalculer les attributs ***x\_base\_*** et ***y\_base\_*** qui sont modifiés du fait du changement de grille, et de réactualiser la carte. Les attributs ***x\_base\_*** et ***y\_base\_*** sont calculés afin de conserver au mieux le centrage de la vue sur la zone observée. Mais lors d'un dézoom, avec une telle méthode, l'emprise de la vue peut être amenée à dépasser de l'emprise du fond de carte. Pour palier cette problématique, la méthode ***zoom\_out()*** détecte par des calculs de taille de grille ce genre de problématique et décale automatiquement la vue en conséquence.

La méthode ***switch\_map(int map\_num)*** permet de changer de fond de carte. Pour cela, elle prend en entrée le numéro de la carte, correspondant à l'indice de cette carte dans l'attribut ***liste\_maps\_***. Elle mute alors les attributs ***layer\_*** et ***format\_*** en conséquence et actualise la carte à l'aide de la méthode ***loader\_carte()***.

Les méthodes ***loader\_carte()*** et ***load\_tile()*** sont en étroite relation. La méthode ***loader\_carte()*** appelle en parallèle 16 méthodes ***load\_tile()***. Cela est possible car la méthode ***load\_tile()*** est dite asynchrone : on n'attend donc pas qu'elle se termine pour passer à la suite du code. On définit dans ***loader\_carte()*** un ***AbortController*** et son ***signal***. Le contrôleur et son signal sont transmis à chacun des 16 appels. Cette astuce permet de lier les 16 appels par un lien commun. Les méthodes ***load\_tile()*** essaient alors toutes de charger une image via une URL donnée en entrée. Si une des url renvoie une erreur, elle enclenche l'avortement du contrôleur. Ce dernier étant partagé par tous les appels de ***load\_tile()***, toutes les fonctions asynchrones vont être arrêtées. Cela évite alors de poursuivre 15 requêtes qui sont vouées à l'échec. Dans un tel cas, on exécute alors un ***zoom\_out()*** afin de revenir à une zone fonctionnelle. Si aucune erreur n'est renvoyée, chacun des 16 appels ajoute à la page web l'image

en sortie à son emplacement défini par les paramètres ***x*** et ***y*** et ajoute cette url à la ***table\_url\_act\_***.

La dernière méthode, ***upload\_all\_tile()***, permet l'envoi de toutes les images dans la base de données sous un nom saisi dans le champ de saisie du formulaire de la page. L'input du formulaire est récupéré par le biais de l'attribut ***value*** de l'attribut ***elem\_name\_*** de notre classe. Ce dernier est une chaîne de texte qui est filtrée à l'aide de la méthode de normalisation décrite en début de partie. On crée ensuite une variable ***data*** composée des différentes variables séparées par un ***&***, à savoir le nom du jeu de données et les 16 URLs de la ***table\_url\_act\_***. Les URLs contenant elle-même des ***&***, il convient de les remplacer par une chaîne de caractère spécifique. Il a été choisi ici de les remplacer au sein des URLs par la chaîne « ***|||*** ». On appelle alors un fetch vers le fichier ***insert\_in\_db.php*** avec comme entrée la variable ***data*** en méthode ***post***.

### 3. [insert\\_in\\_db.php](#)

Dans notre code, ***insert\_in\_db.php*** reçoit par la méthode ***post*** le nom du jeu de données ainsi que 16 URLs dont les ***&*** ont été remplacées par la chaîne « ***|||*** ». La première étape consiste donc à récupérer le nom et les URLs, tout en remplaçant dans ces dernières les « ***|||*** » par des ***&*** (cf. lignes 5 à 37). Toutes ces données sont alors envoyées dans la base de données en ligne à l'aide de 1+16 requêtes SQL étant générées automatiquement dans une boucle ***for*** à 16 itérations.

### 4. [index.html](#)

Le fichier ***index.html*** situé dans le dossier HTML contient la structure même de la page web. Elle est organisée en divisions permettant de hiérarchiser les différents éléments. Un schéma simplifié de la structure est montré ci-dessous :

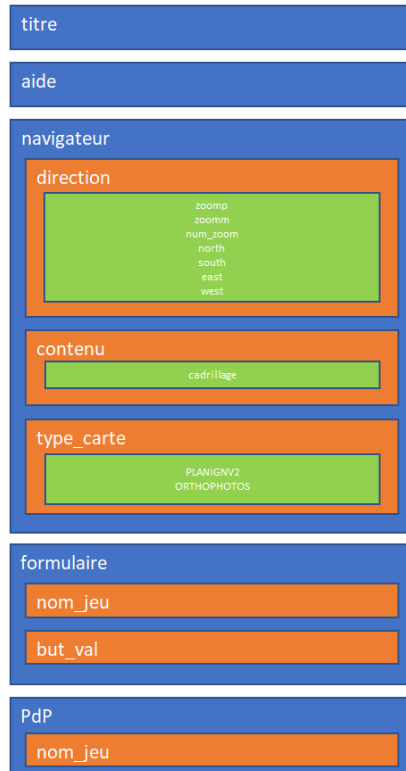


Figure 2 : Schéma de structure de la page web