mviewer Documentation

Version 2

Communauté mviewer

avr. 24, 2024

Documentation utilisateur

1 Introduction 1

2 Documentation utilisateur 3

3 Documentation technique 17

4 Documentation dévelopeur 93

5 Documentation contribution 127

6 Auteurs et licence 147

Index 149

CHAPITRE 1

Introduction

Ci-dessous vous trouverez une documentation pour les utilisateurs ou administrateurs de la plateforme mviewer.

1.1 mviewer

mviewer est un Visualiseur géographique basé sur OpenLayers 6.3.1 et Bootstrap 3.3.6.

L'ensemble des sources de ce projet peut être trouvé sur le dépôt GitHub .

CHAPITRE 2

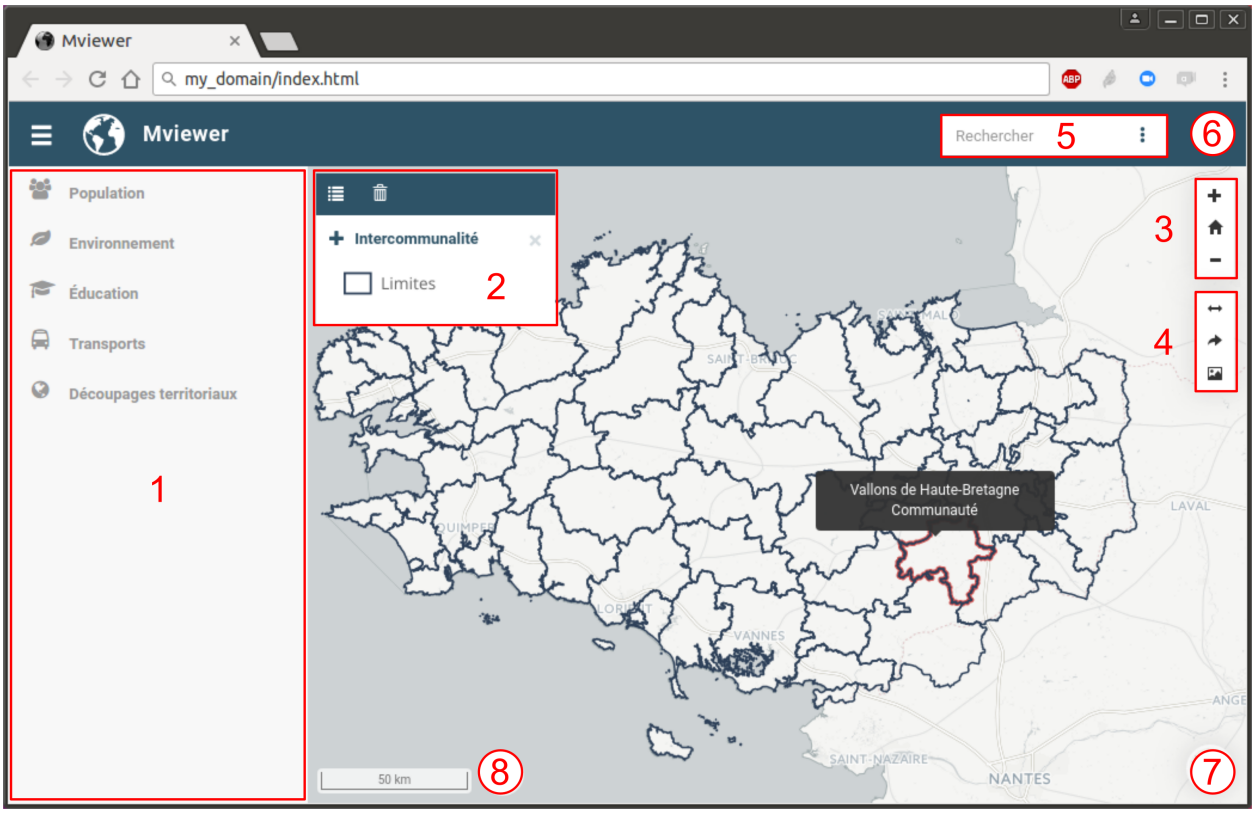
Documentation utilisateur

Cette partie est dédiée aux utilisateurs qui souhaitent prendre en main l'interface de mviewer.

2.1 Interface utilisateur

L'interface de mviewer peut être décomposée en 8 rubriques :

1. *Gestion des couches*
2. *Gestion de l'affichage*
3. *Outils de navigation*
4. *Outils aditionnels*
5. *Barre de recherche*
6. *Documentation*
7. *Fond de carte*
8. *Crédits*



2.1.1 Gestion des couches

Panneau listant l'ensemble des couches pouvant être chargées dans la carte. Pour en savoir plus, consulter la page « *Gestionnaire de couches* ».

2.1.2 Gestion de l'affichage

Panneau de gestion de l'affichage des couches sélectionnées. Pour en savoir plus, consulter la page « *Gestion de l'affichage* ».

2.1.3 Outils de navigation

Outils de zoom dans la carte. Pour en savoir plus, consulter la page « *Outils de navigation* ».

2.1.4 Outils aditionnels

Outils permettant :

- de mesurer des aires ou des distances sur la carte,
- de partager la carte,
- d'exporter la carte sous forme d'image.

Pour en savoir plus, consulter la page « *Outils additionnels* ».

2.1.5 Barre de recherche

Moteur de recherche de lieux (ville, département, région…). Pour en savoir plus, consulter la page « *Barre de recherche* ».

2.1.6 Documentation

Informations complémentaires permettant de décrire le contexte de la plateforme, les données diffusées, les points de contact… Pour en savoir plus, consulter la page « *Documentation* ».

2.1.7 Fond de carte

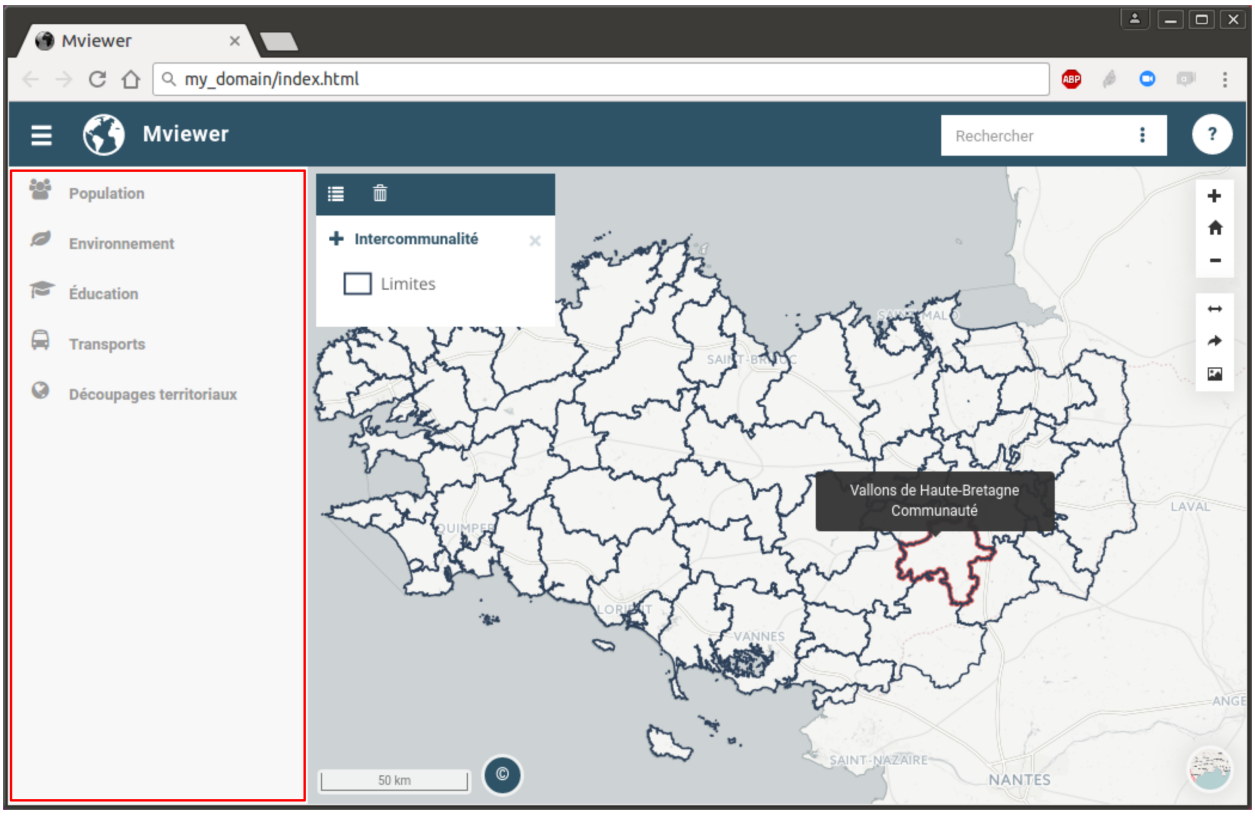
Outil permettant de changer le fond de carte parmi une liste prédéfinie. Pour en savoir plus, consulter la page « *Fonds de carte* ».

2.1.8 Crédits

Crédits relatifs aux fonds de cartes. Pour en savoir plus, consulter la page « *Crédits* ».

2.2 Gestionnaire de couches

Dans **mviewer**, la liste des données disponibles à l'affichage est visible dans le panneau de gauche.



2.2.1 L'arborescence

Comme décrit dans la page « *Configurer - Les couches* », les couches peuvent être organisées de la manière suivante :

- **un thème,**
  - **un (ou plusieurs) groupe(s) *(ce niveau est optionnel)*,**
    - une (ou plusieurs) couche(s).

Par défaut, l'arborescence est repliée et seuls les thèmes sont visibles. En cliquant sur un thème, son contenu *(groupe(s) ou couche(s))* apparaît. Il en va de même sur un groupe qui peut être déplié pour voir apparaître les couches qu'il contient. De la même façon, un clic sur un thème / groupe déplié va le replier.

2.2.2 Afficher / cacher des couches

Pour afficher une couche dans la carte, il vous suffit de cliquer dessus. Dès lors, l'icone ( ) situé devant le nom devient ( ).

Pour supprimer une couche de la carte, il vous suffit de cliquer de nouveau sur son nom.

2.2.3 Afficher / cacher le panneau

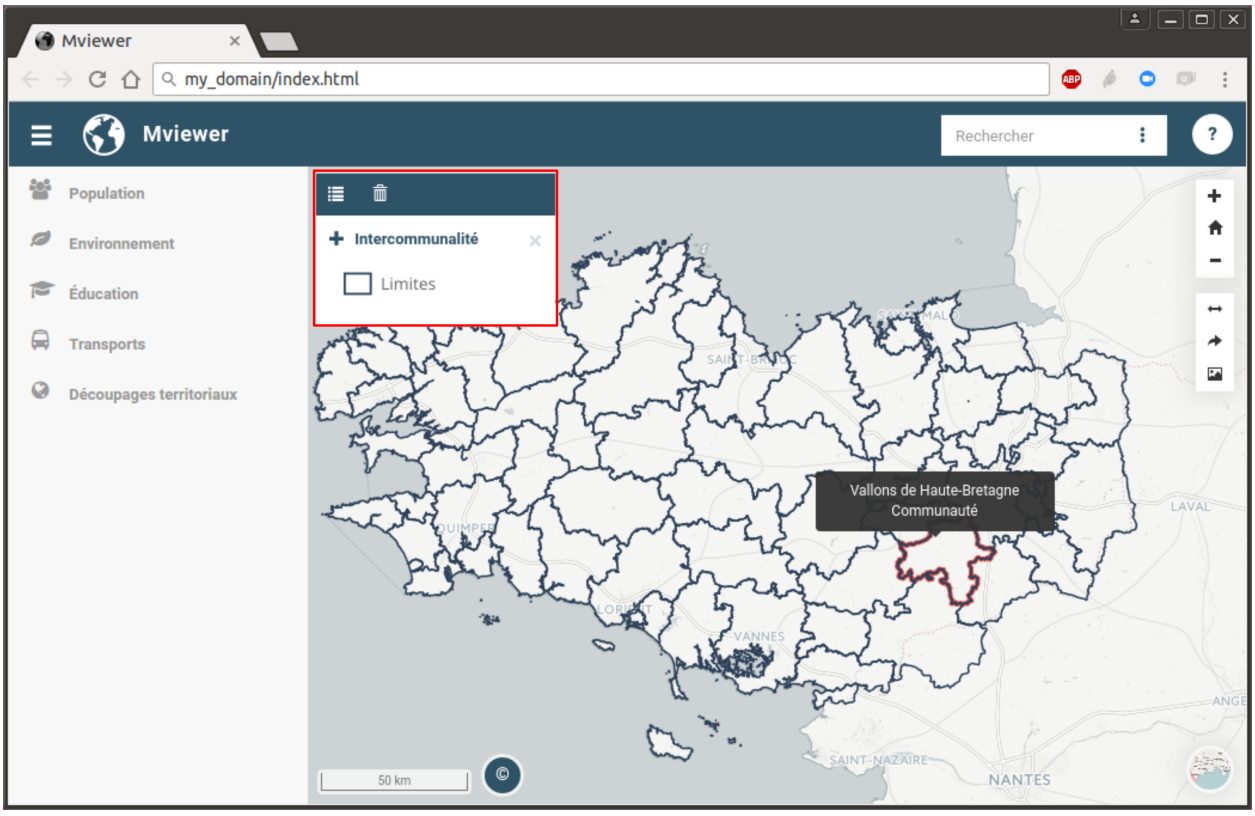
En cliquant sur l'icone ( ) vous avez la possibilité de plier / déplier le panneau.

2.2.4 Configurer

Pour savoir comment modifier la liste des couches présentes dans ce panneau, vous êtes invités à consulter la page « *Configurer - Les couches* ».

2.3 Gestion de l'affichage

Le gestionnaire d'affichage de couche se trouve sur la partie gauche de la carte.

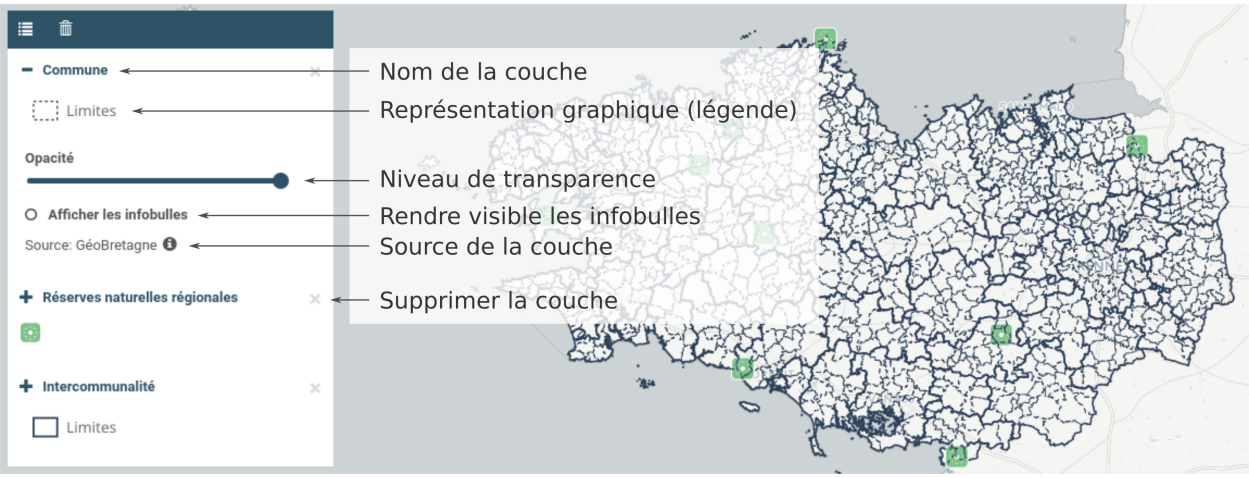


Lorsque dans le panneau latéral (voir « *Gestionnaire de couches* ») vous cliquez sur une couche, celle-ci apparaît dans le gestionnaire d'affichage.

2.3.1 Options sur une couche

Là, vous avez la possibilité :

- de visualiser le rendu graphique *(la légende)*,
- de modifier l'opacité *(transparence)*,
- d'afficher des informations sous la forme d'infobulles *(optionnel)*,
- de voir la source de la donnée *(optionnel)*,
- de supprimer la couche de la carte*(en cliquant sur l'icone* ✕ *)*



2.3.2 Autres options

De plus, l'utilisateur à la possibilité de :

- modifier l'ordre d'affichage : pour cela, il vous suffit de cliquer sur une couche et de la glisser / déposer à l'endroit désiré. Notez que la couche en haut dans la liste sera affichée au premier plan dans la carte.
- supprimer *(ne plus afficher)* l'ensemble des couches ( )
- plier / déplier le gestionnaire ( )

2.4 Outils de navigation

Pour naviguer dans la carte, l'utilisateur à la possibilité d'utiliser trois outils présents sur la droite de l'interface :

- Zoom + ( )
- Zoom sur la position initiale ( )
- Zoom - ( )

À noter qu'il est également possible de naviguer à l'aide de la souris et du clavier :

- Avec la roulette : Zoom + / - ,
- Double clic-gauche : Zoom +
- Maj + Double clic-gauche : Zoom -
- Clic-gauche : déplacement de la carte, sans changer le niveau de zoom.

Pour modifier la position de départ du zoom (et donc celle appliquée avec l'outil), veuillez vous référer à la page « *Configurer - les options de la carte* ».

2.5 Outils additionnels

Les outils additionnels se comptent au nombre de trois :

1. *Outils de mesure*
2. *Partage de carte*
3. *Export de la carte*

2.5.1 Outils de mesure

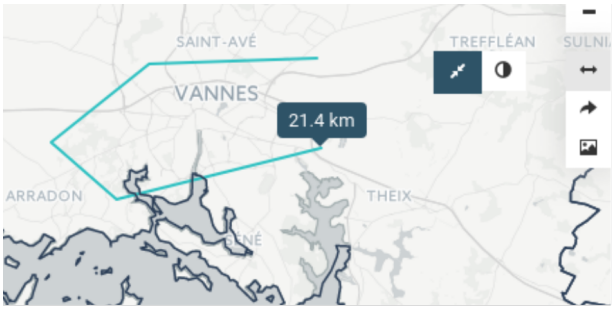
En cliquant sur l'icone ( ), deux nouveaux outils apparaissent et vous avez la possibilité de mesurer :

— une distance ( ),

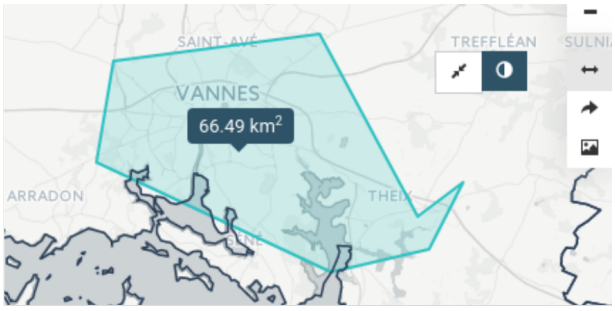
— une surface ( ).

Marche à suivre

Pour faire une mesure, il vous suffit de dessiner une ligne (ou une surface) à l'aide du clic-gauche de votre souris. Au fur et à mesure que vous avancez dans le dessin, la distance cumulée (ou bien la surface) est affichée.



Mesure d'une distance



Mesure d'une surface

Pour terminer une mesure, faites un double clic-gauche avec la souris.

Pour effacer une mesure (distance ou bien surface), il vous suffit de cliquer de nouveau sur l'outil que vous avez utilisé.

2.5.2 Partage de carte

Le partage de carte permet de générer un lien web fixant la situation actuelle de l'interface. Ainsi sont conservés :

— la position de la carte,

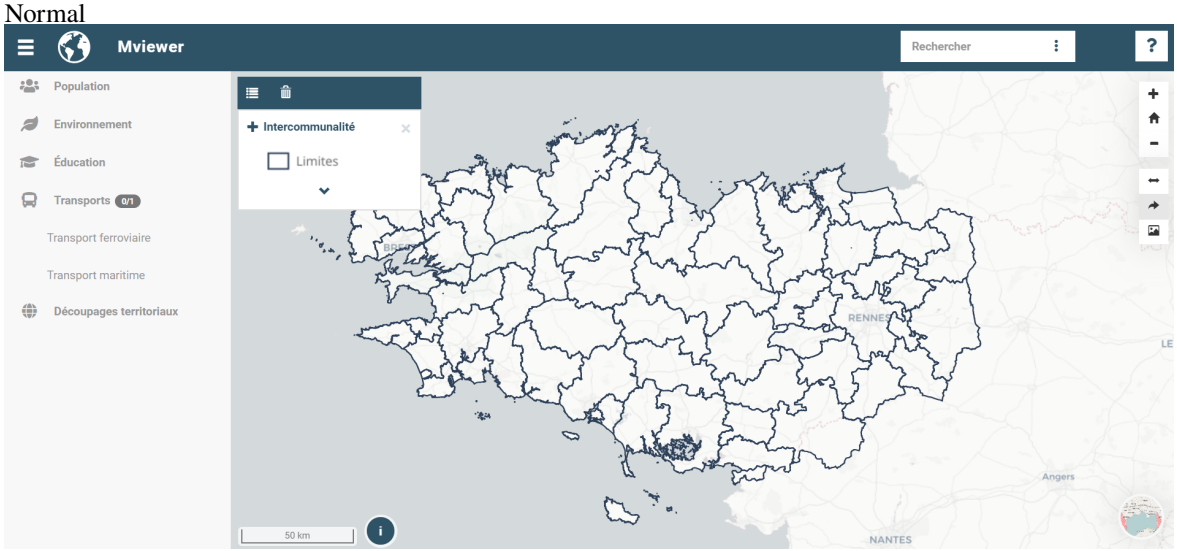
— le niveau de zoom,

— la liste des couches affichées avec leurs paramètres de transparence.

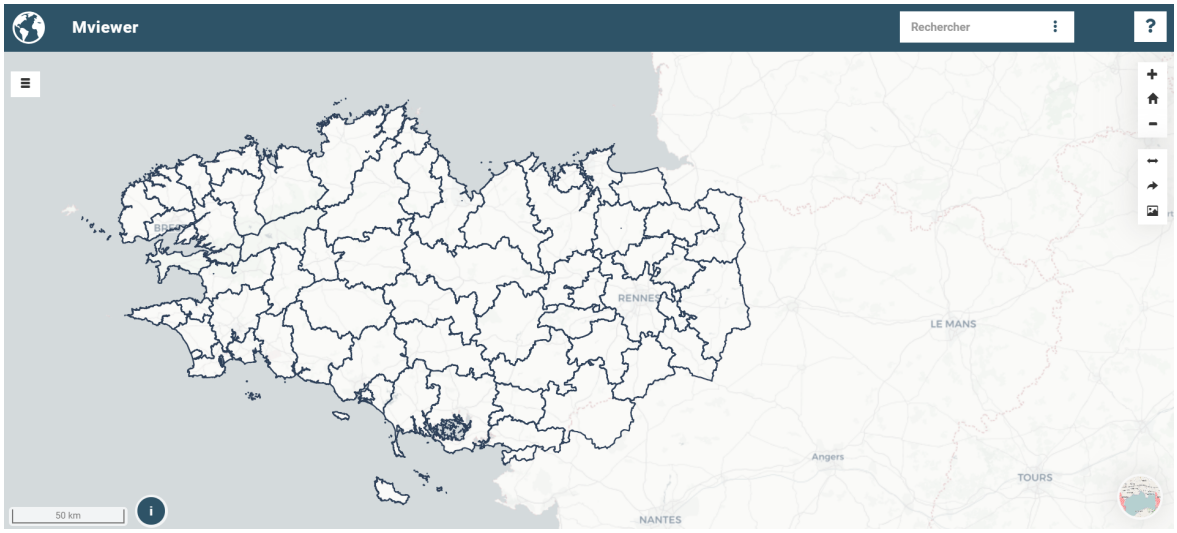
Une fois que vous avez cliqué sur l'icone ( ), une nouvelle fenêtre apparaît :

Choix du mode d'affichage du permalien

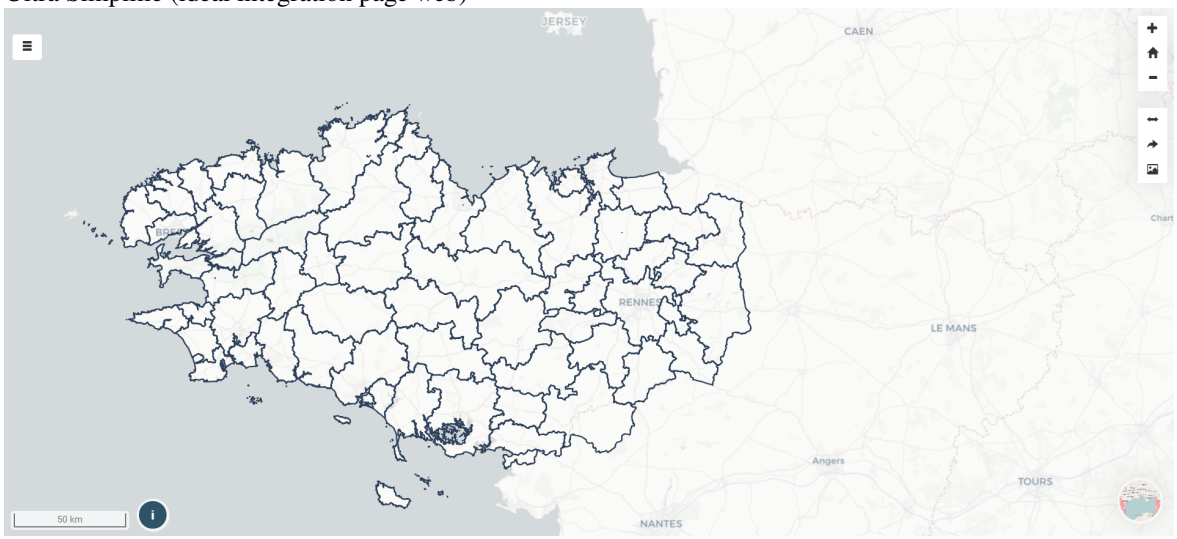
— Normal



— Simplifié



— Ultra Simplifié (idéal intégration page web)



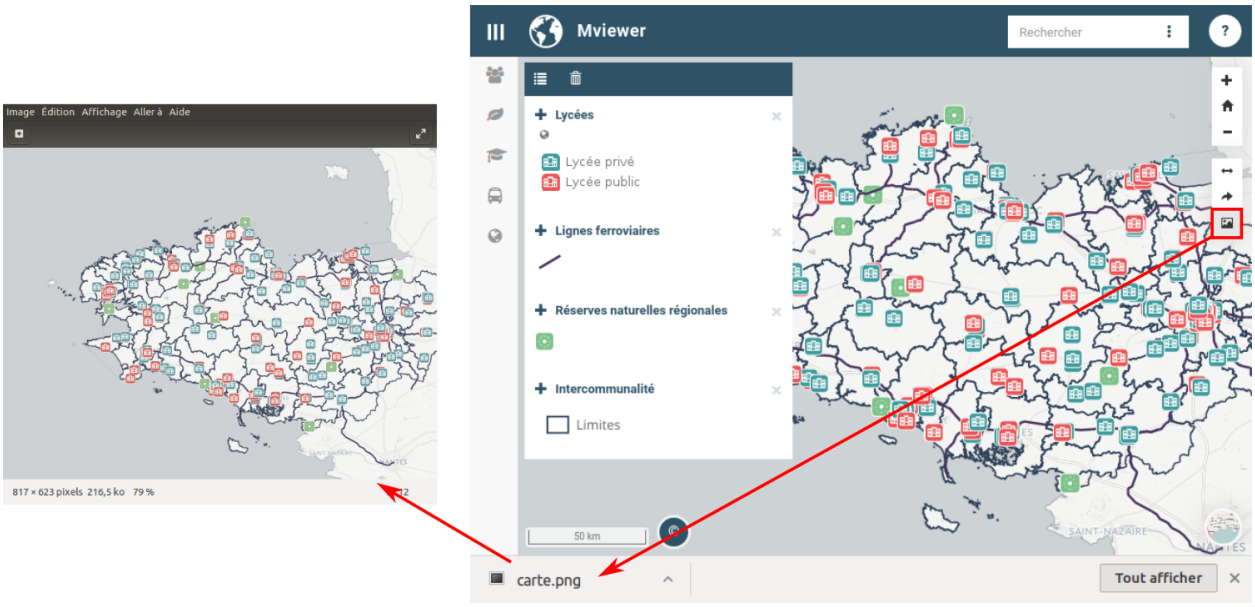
Choix du type de lien

— générer un lien hypertext *(icone de gauche)* : lorsque vous cliquez, un nouvel onglet de votre navigateur s'ouvre avec le lien permanent,
— utiliser un QR Code *(icone de droite)*.

Exemple de permalien : http://localhost/mviewer/?x=-220750&y=6144926&z=8&l=epci&lb=positron&mode=u
— x et y : coordonnées du centre de la carte
— z : niveau de zoom
— l : liste des couches
— lb : couche de fond par défaut
— mode : mode d'affichage (n,s ou u)

2.5.3 Export de la carte

En cliquant sur l'icone ( ) la carte est automatiquement exportée au format .png.



2.6 Barre de recherche

La barre de recherche, située en haut à droite de l'interface, permet de rechercher tout type de lieux, comme par exemple des noms de communes, de département ou de région ainsi que, si c'est paramètré, des entités.

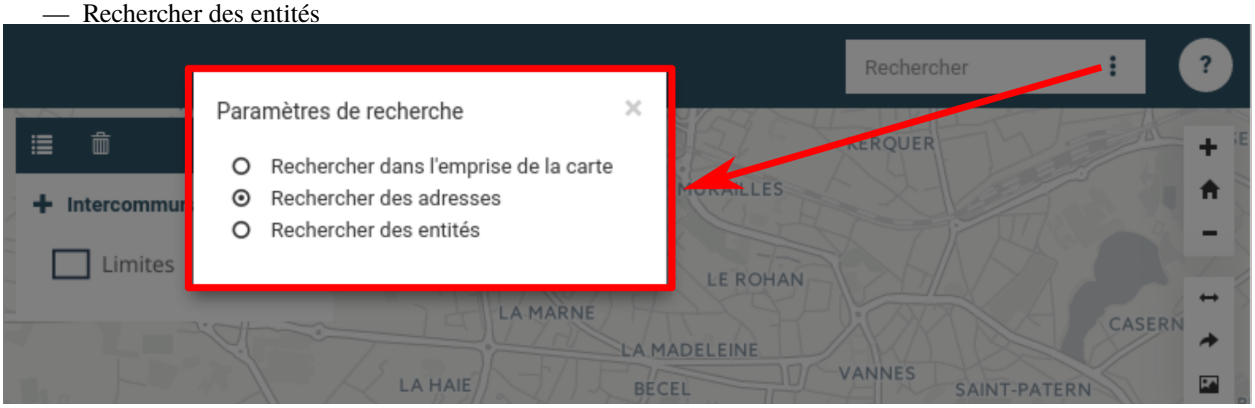
Au fur et à mesure que l'utilisateur écrit, le moteur de recherche affiche les propositions correspondantes. Là, l'utilisateur est invité à cliquer sur l'entrée qui correspond à son attente. Dès lors, le navigateur va zoomer sur l'entité sélectionnée.

En cliquant sur la **croix** *(à droite de la zone d'écriture)*, l'utilisateur efface le contenu de la zone de texte.

2.6.1 Options

En cliquant sur l'icone composée de 3 points, l'utilisateur à la possibilité d'affiner les options de recherche, avec les choix suivants :

— Rechercher dans l'emprise de la carte
— Rechercher des adresses
— Rechercher des entités

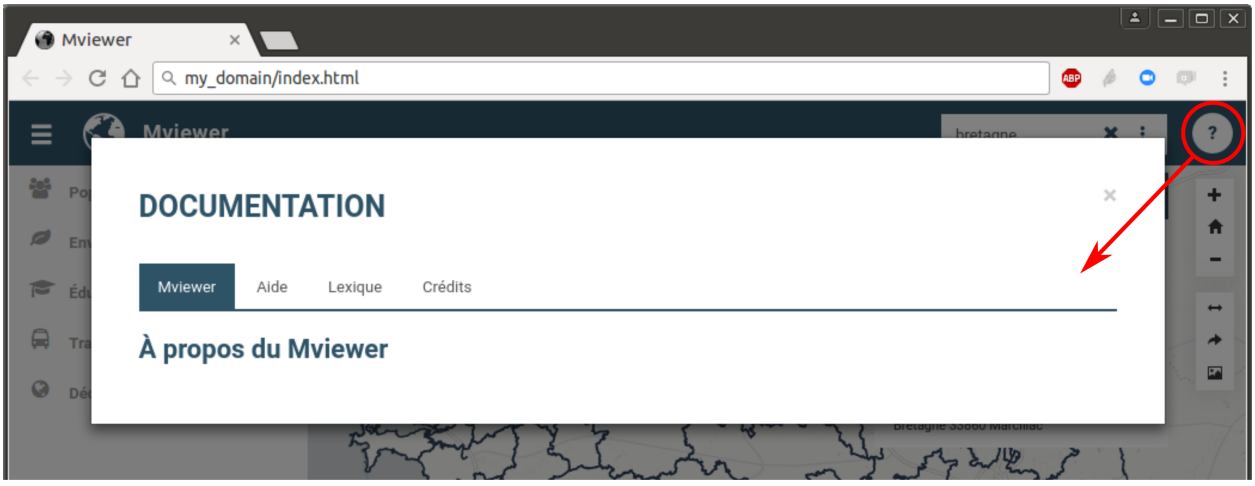


2.7 Documentation

Le panneau de documenation offre aux utilisateurs des informations complémentaires permettant de décrire le contexte de la plateforme, les données diffusées, les points de contact ou toutes autres données nécessaires.

2.7.1 Ouvrir et fermer le panneau

En cliquant sur le bouton « **?** », un nouveau panneau s'affiche au premier plan de l'écran.

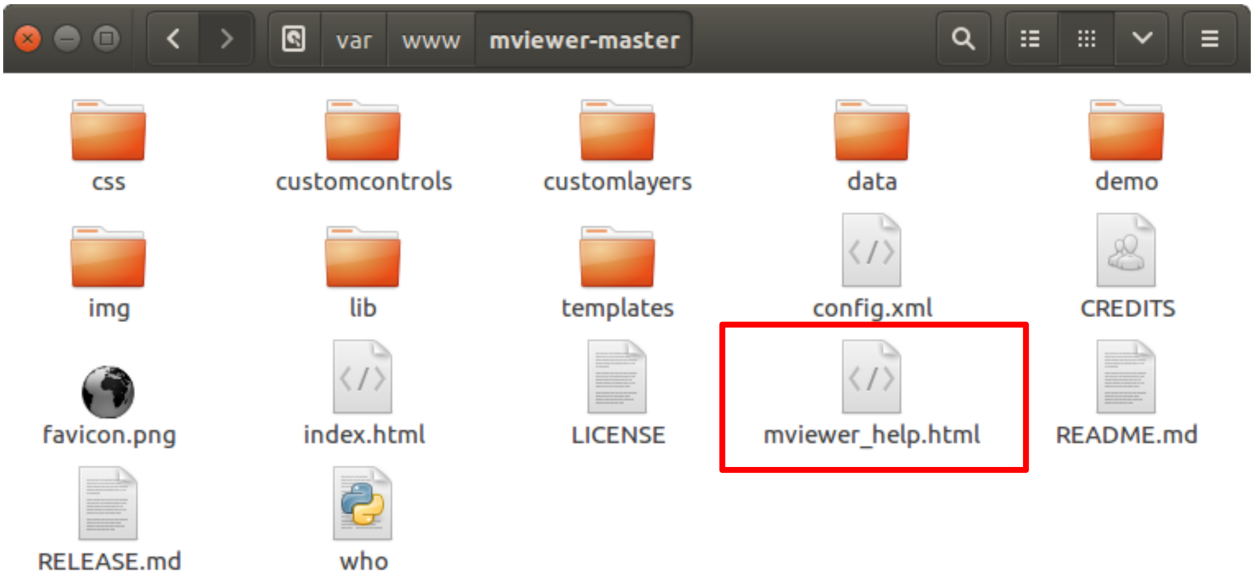


Pour fermer ce panneau, il vous suffit soit :

— de cliquer sur la croix en haut en droite du cadre,
— de cliquer en dehors du cadre.

2.7.2 Configurer le panneau

Techniquement, ce panneau de documentation se présente sous la forme d'un fichier .html que l'on retrouve à la racine du dossier de **mviewer** : **mviewer_help.html**.

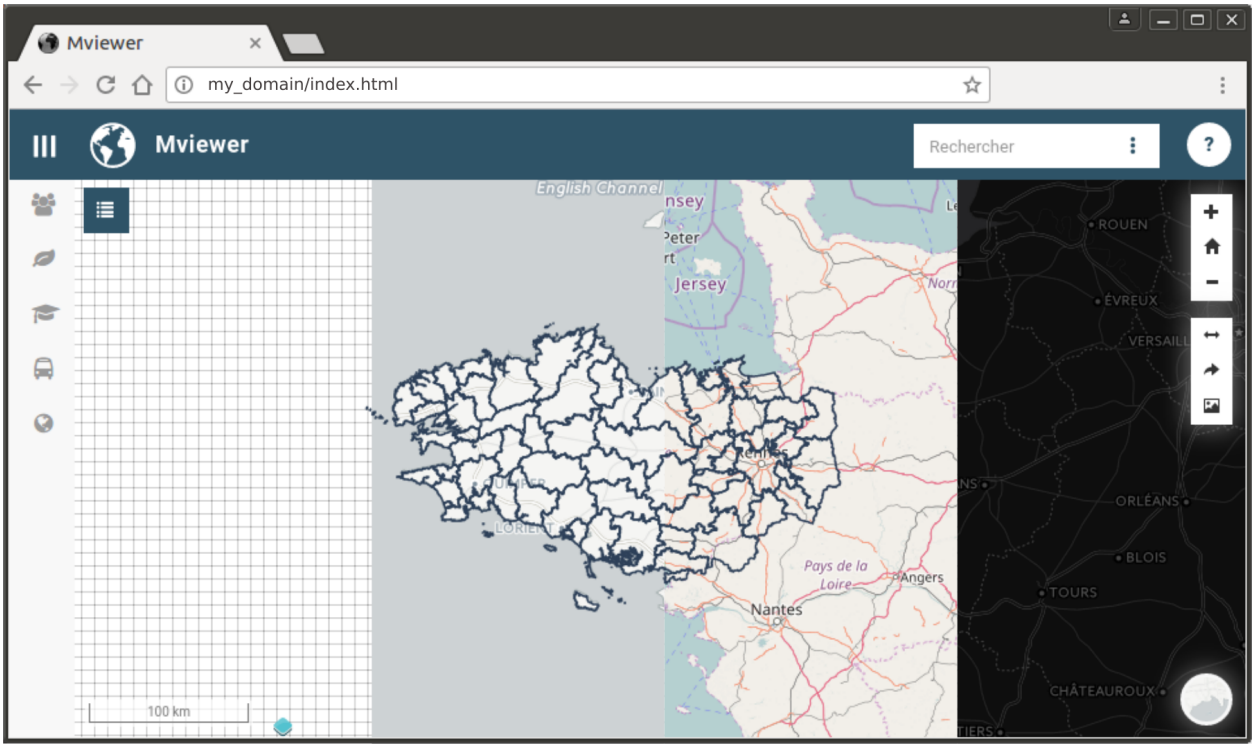


En éditant et modifiant ce fichier, vous aurez la possibilité de gérer (ajouter/supprimer/renommer) les onglets, ainsi que leur contenu. Pour plus d'information, veuillez consulter la page XXXX.

2.8 Fonds de carte

En cliquant sur l'icone ( ) en bas à droite de la carte, l'utilisateur à la possibilité de changer le fond de carte. Par défaut, quatre fonds sont proposés :

- Positron *(actif au démarrage)*,
- OpenStreetMap,
- Carroyage,
- Dark Matter.



Carroyage *Positron* *OpenStreetMap* *Dark Matter*

Pour modifier la liste des fonds de carte, veuillez consulter la page « *Configurer - Les fonds de carte* ».

2.9 Crédits

En cliquant sur l'icone (©) l'utilisateur fait apparaître les crédits du *Fonds de carte* actuellement affiché. Lorsque ce dernier est changé, les crédits sont automatiquement mis à jour.

Pour replier la fenêtre de crédit, il vous suffit de cliquer sur l'icone » .

CHAPITRE 3

Documentation technique

Cette partie est dédiée aux personnes qui ont vocation à déployer et configurer la plateforme mviewer.

3.1 Installer MVIEWER

Mviewer est une application web développée en HTML / CSS / JAVASCRIPT. Elle nécessite simplement d'être déployée sur un serveur WEB qui peut être APACHE, NGINX, TOMCAT…

3.1.1 Avec un serveur web Apache

Avertissement : Prérequis : disposer d'une instance **Apache**

Clonage des sources et déploiement

L'objectif est de copier les sources depuis github.com dans le répertoire web d'Apache.

Avec git

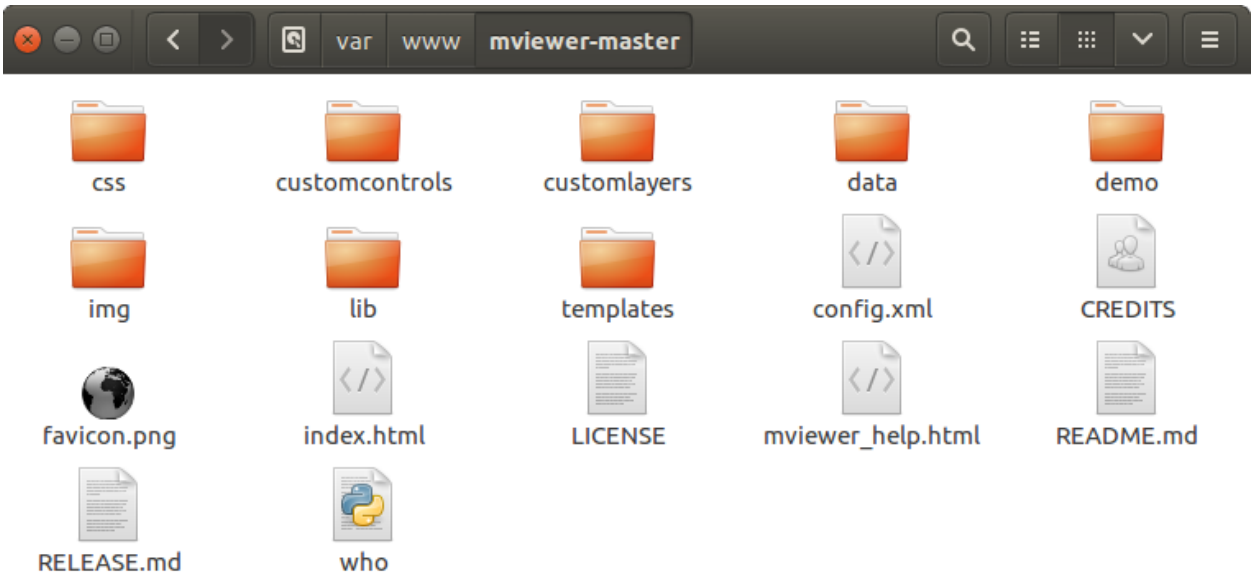
Dans un terminal, après vous être placé dans le dossier web Apache, exécuter la commande **git** suivante.

```
git clone https://github.com/mviewer/mviewer.git
```

Sans git

Télécharger ce fichier zip présent sur la page d'accueil du dépôt mviewer sur GitHub : https://github.com/mviewer/mviewer

Dézipper le contenu du zip dans le dossier web Apache **/var/www/** *(ou autres dossiers de déploiement Apache)*.



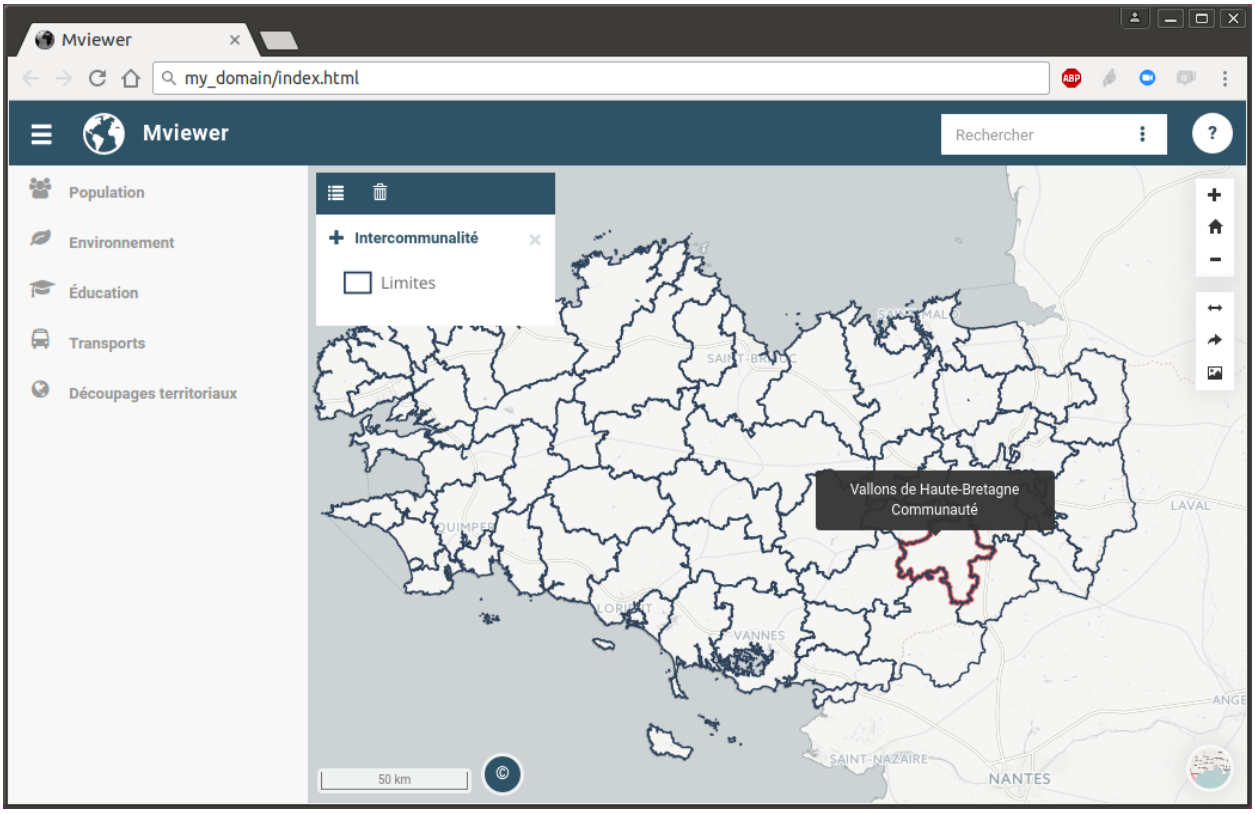
Test navigateur

Une fois déployée, l'application est accessible dans le navigateur internet en saisissant l'URL http://nom_du_serveur/nom_du_mviewer/demo/

exemples :

- http://localhost/mviewer/demo/
- http://localhost/miewer-master/demo/index.html

En lançant l'application racine **index.html**, vous avez maintenant un visualiseur géographique fonctionnel avec les couches de la Région Bretagne *(configuration par défaut disponible dans apps/default.xml)*.



3.1.2 Avec Docker

Avertissement : Prérequis : disposer de **docker**

mviewer en mode standalone

C'est la solution la plus simple. Il est ainsi possible de lancer mviewer et de visualiser toutes les démos disponibles dans l'application. Ce mode ne convient pas pour effectuer ses propres applications mviewer.

```
#Récupérer la dernière image buildée de mviewer
docker pull mviewer/mviewer
#Lancer le container docker mviewer
docker run --rm -p80:80 mviewer/mviewer
```

mviewer et un dossier d'applications

C'est la solution à privilégier pour créer ses propres applications. En parallèle de mviewer, un dossier web **apps** (volume) est monté. Ce dossier web contient vos applications mviewer ainsi que les ressources nécessaires.

```
# Lancer le container mviewer + le répertoire web apps qui pointe vers
# /chemin/vers/repertoire_de_configurations_xml
docker run --rm -p80:80 \
    -v/chemin/vers/repertoire_de_configurations_xml:/usr/share/nginx/html/apps \
    mviewer/mviewer
```

mviewer avec docker-compose

Cette solution permet de mettre en place les mêmes possibilités que la méthode précédante (mviewer + volume d'applications) en utilisant docker-compose et deux fichiers de paramétrage

Avertissement : Prérequis : disposer de **docker** et **docker-compose**

1. Création du fichier environemment

Code source 1 – settings.json

```
APPSPATH=/home/prod/mviewer-apps
```

2. Création du fichier de configuration

Code source 2 – docker-compose.yml

```
version: '3.2.1'
services:
  mviewer:
    container_name: mviewer
    build: .
    ports:
      - "80:80"
    image: mviewer/mviewer
    volumes:
      - '${APPSPATH}:/usr/share/nginx/html/apps'
```

3. Lancer mviewer

```
docker pull mviewer/mviewer
docker-compose up
```

Test navigateur

Une fois déployée, l'application est accessible dans le navigateur internet en saisissant l'URL http://nom_du_serveur/demo/

exemples :

- http://localhost/demo/
- http://localhost/index.html
- http://localhost/?config=apps/default.xml

3.1.3 Mettre à jour mon instance de mviewer

Lors d'une nouvelle release de mviewer, il est conseillé d'utiliser git pour mettre à jour son instance.

1. Tester la nouvelle version sur une version de test mviewer-dev

```
cd /var/www/mviewer-dev
git clone https://github.com/mviewer/mviewer.git
```

Puis, tester cette instance en pointant sur des applications en production. Exemple : https://localhost/mviewer-dev?config=prod/monappli.xml.

2. Mise à jour de la version de prod. On peut faire une sauvegarde de notre instance actuelle (ici avec le dossier mviewer-save)

```
cd /var/www/
cp -r mviewer mviewer-save
cd mviewer
git pull origin
```

Puis, tester cette instance. Exemple : https://localhost/mviewer?config=prod/monappli.xml.

3.1.4 Configuration et adaptations

Si vous souhaitez publier vos propres couches/thèmes ou bien ajouter/supprimer certaines fonctionnalités, veuillez consulter la page « *Configurer - Principe* ».

3.2 Configurer - Principe

Le visualiseur cartographique mviewer consomme des données et des services servis par le web (flux OGC WMS/WFS/WMTS, services de géocodage…). La connexion et le paramétrage de ces services se fait dans un fichier de configuration pivot qui permet en outre de personnaliser l'application (titre, logo, feuille de style…).

Si aucune configuration n'est indiquée dans l'URL, c'est la configuration par défaut qui s'applique (**apps/default.xml**).

Il est possible de créer n fichiers de configuration (config1.xml, config2.xml…) de façon à créer plusieurs applications thématiques à partir d'une seule instance mviewer. Pour appeler une configuration particulière, il suffit alors de rajouter le paramètre **config** dans l'url, exemple ?config=demo/geobretagne.xml .

Pour voir des **exemples d'applications et de fichiers de configuration XML**, rendez vous à cette page : démos.

En cliquant sur **Sources**, vous accèderez au fichier XML et le le bouton **Live !** vous permet de visualisation l'application.

3.2.1 Structure du fichier de configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <application />
    <extensions>
    <mapoptions />
    <baselayers>
        <baselayer />
    </baselayers>
```

(suite sur la page suivante)

(suite de la page précédente)

```
    <proxy />
    <olscompletion />
    <elasticsearch />
    <searchparameters />
    <themes>
        <theme>
            <layer />
        </theme>
    </themes>
</config>
```

3.2.2 Paramètres d'URL

Il est possible d'instancier un mviewer avec des **paramètres** de configuration transmis par URL

- `config` : Fichier de configuration à charger ex : `mviewer/?config=apps/mon_appli.xml`.
- `#` : Il s'agit d'un raccourci pour appeler une config présente dans le dossier apps. ex : `mviewer/#mon_appli`.
- `theme` : Thème css à utiliser ex : `?theme=geobretagne` pour charger le theme doit être dans css/themes/geobretagne.css.
- `wmc` : liste des contextes OGC WMC (séparés par des virgules) à charger afin d'alimenter le panel de gauche ex : `mviewer/?wmc=demo/hydro.wmc`
- `popup` : true ou false. Si true, une popup s'affiche sur la carte afin d'afficher le résultat de l'interrogation de couches.
- `lang` : Langue à utiliser pour l'interface. Passer exemple `?lang=en`.
- `mode` : Mode d'affichage à utiliser (d - default, s - simplifié, u - ultrasimplifié). Le mode simplifié ne dispose pas du panneau des thématiques et le mode ultra simplifié ne dispose pas de la barre de navigation.
- `title` : Titre à utiliser. Seulement exploité en mode défault et simplifié.
- `topics` : Thèmes à filtrer.
- `addLayer` : pour ajouter une couche WMS à la carte. ce paramètre prends comme valeur un objet **JSON** contenant
  - `url` : url du service
  - `name` : nom de la couche (layername)
  - `title` : label/titre à afficher dans mviewer

exemple pour le paramètre **addLayer** : `&addLayer={\%22url\%22:\%22https://www.geo2france.fr/geoserver/hdf_common/ows\%22,\%22name\%22:\%22Antennes__HdF_EnService_Agreg\%22,\%22title\%22:\%22Antennes_test\%22}`

Paramètres d'URL utilisés pour les permaliens

- `x` : Coordonnées x du centre de la carte dans le système de projection utilisé par l'application.
- `y` : Coordonnées y du centre de la carte dans le système de projection utilisé par l'application.
- `z` : Zoom de la carte (1 à 20).
- `lb` : Identifiant de la couche de fond affichée.
- `c_[monparam]` : Où mon param est l'identifiant du composant personnalisé ou de la couche personalisée. La valeur du paramètre peut ensuite être utilisée par le composant ou la couche concernée. exemple c_mycustomlayer=red,blue,green
- Plus d'informations sur les permaliens « *Outils additionnels* » menu Partage de carte

3.2.3 Sections de configurations

— Configurer l'application « *Configurer - Application* ».
— Configurer les extensions « *Configurer - Extensions*
— Configurer la carte « *Configurer - les options de la carte* ».
— Configurer les couches de fonds « *Configurer - Les fonds de carte* ».
— Configurer les couches thématiques « *Configurer - Les couches* ».
— Configurer la recherche « *Configurer - La recherche* ».
— Configurer le proxy « *Configurer - Le proxy* ».

3.3 Bien commencer avec mviewer

3.3.1 Préparer votre environnement de travail

Si vous débutez, nous vous conseillons de créer un dossier unique nommé « git ». Nous installerons ensuite Git et le terminal Git Bash pour passer les commandes à Git.

— Sur Windows :

```
C:\Users\jean\Documents\git
```

— Sur Ubuntu/Debian (autre) :

```
/home/user/jean/git
```

— Le code mviewer sera par la suite placé dans un répertoire « mviewer » :

```
Exemple: C:\Users\jean\Documents\git\mviewer
```

— Télécharger ensuite Git et installez le tout. Le terminal Git Bash sera installé en même temps.

Git Bash vous permettra de réaliser les commandes qui suivront.

3.3.2 Travailler avec un fork

Règle générale : ne jamais modifier la branche MASTER.

La branche master est une branche « mirroir » de la branche master du code initial (mviewer/mviewer).

C'est votre point de départ pour tout nouveau travail et tout nouveau travail doit repartir d'une base à jour et propre.

Nous recommandons de créer une branche à partir de la branche master à jour pour chaque nouveau travail.

La section qui suit vous donnera la procédure pour obtenir votre fork.

3.3.3 Récupérer les sources (fork)

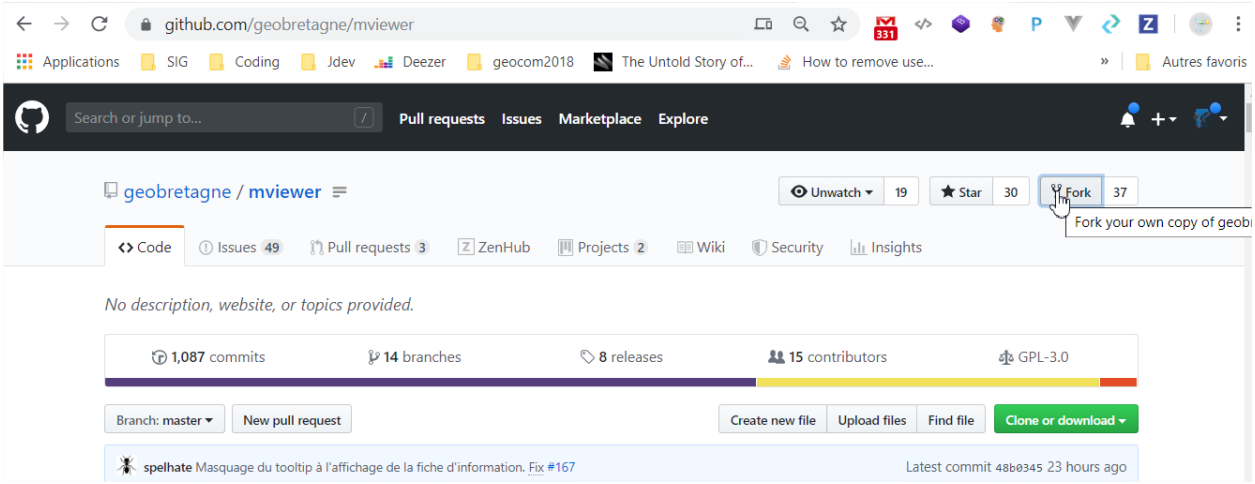
Pour bien débuter, nous vous recommandons de réaliser un fork vers votre espace GitHub.

Prérequis

— Disposer d'un compte GitHub
— Avoir les droits de création sur ce compte
— Être connecté sur GitHub

Procédure

Sur la page GitHub du mviewer cliquer sur « Fork » en haut à droite.



Choisissez ensuite le compte vers lequel réaliser votre fork.

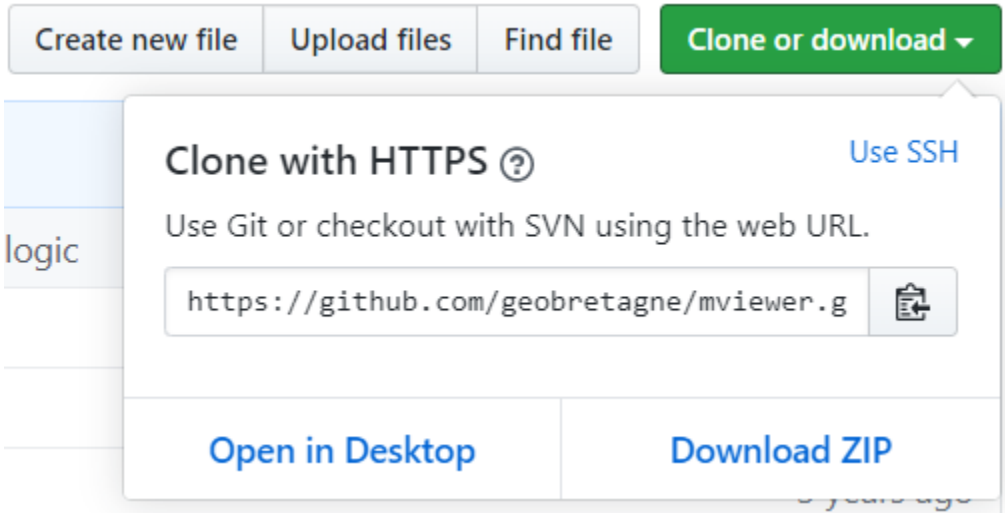
Vous détenez maintenant un fork disponible à l'adresse : https://github.com/MON_ORG/mviewer *MON_ORG étant à remplacer par le nom de votre compte GitHub.*

Votre fork contient nativement les mêmes branches à l'identique, dont la branche **master**.

Vous pourrez créer des nouvelles branches et modifier le code sans impacter le code natif du repository inital (mviewer/mviewer).

— Ouvrez Git Bash
— Allez sur le repository mviewer GitHub et cliquez sur « clone or download ». Copier ensuite l'URL dans la commande suivante.
— Réalisez un clone (copie) du code mviewer sur votre ordinateur en collant l'URL précédente :

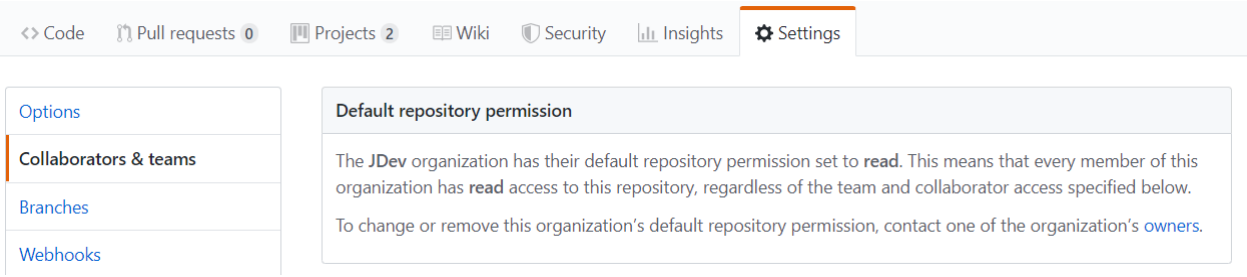
```
git clone https://github.com/MON_ORG/mviewer.git
```



3.3.4 Gestion des droits

Vous pouvez gérer les droits de votre repository pour les utilisateurs, développeurs et autres personnes disposant d'un compte GitHub.

Pour cela, cliquez en haut dans « Settings » et accédez à gauche à l'onglet « Collaborators & team ».

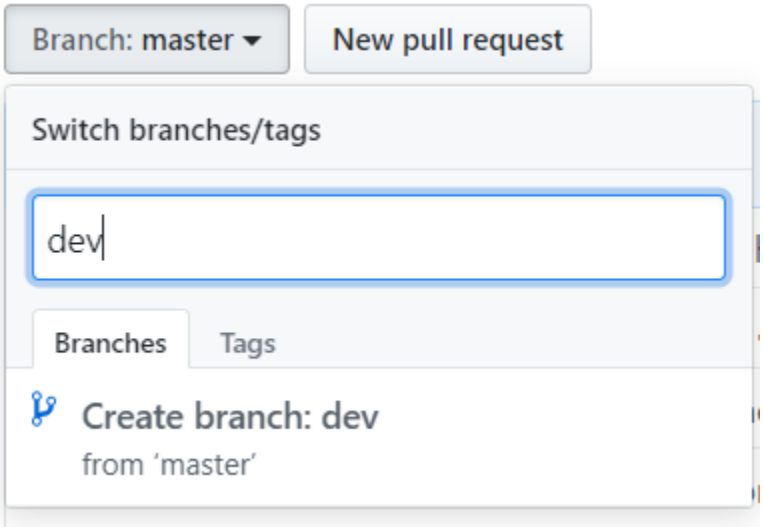


3.3.5 Créer une branche

Vous pouvez créer une branche sur GitHub directement ou avec Git.

Pour créer une branche su GitHub.

— Cliquez sur le bouton de la liste Branch :(nom de branche)
— Choisissez la branche de départ (master) dans la liste des branches disponibles
— Cliquez à nouveau sur le bouton de la liste Branch :(nom de branche)
— Dans le champ de recherche, saisissez le nom de votre nouvelle branche (*).



— Cliquez ensuite sur « Create Branch : (nom de votre branche) »

Vous avez maintenant une nouvelle branche.

Pour récupérer cette nouvelle branche dans votre dépôt Git, suivez la procédure qui suit :

— Positionnez-vous dans votre dossier ../git/mviewer/ :
— Synchronisez votre dossier git :

```
git fetch
```

— Positionnez-vous dans votre dossier mviewer (le dossier créé par la commande "git clone") :

```
git pull
```

— Si vous souhaitez changer de branche, saisissez cette commande :

```
git checkout NOM_DE_MA_BRANCHE
```

3.3.6 Récupérer les nouveautés de la branche

Pour mettre à jour votre dépôt (dossier) local mviewer (/git/mviewer) réalisez la commande :

```
git pull
```

Vous travaillerez maintenant sur la nouvelle branche de votre choix. Chaque nouvelle mise à jour de la branche master de votre fork devra être reportée sur cette branche aussi souvent que possible.

Pour mettre à jour la branche master de votre fork, nous devons définir en premier une « source distante » (upstream).

(*) *Attention : Choisissez un nom permettant d'identifier rapidement cette branche pour vous et votre équipe.*

3.3.7 Définir un upstream

Pour mettre à jour la branche master depuis le code de mviewer, vous devrez indiquer quelle est la « source distante » (upstream). Votre « origin » sera votre votre repository mviewer (fork).

Voici la manipulation.

— Définir un upstream :

```
git remote add upstream https://github.com/mviewer/mviewer
```

— Observer que vous avez bien un upstream :

```
git remote -v
> origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
> origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
> upstream  https://github.com/mviewer/mviewer.git (fetch)
> upstream  https://github.com/mviewer/mviewer.git (push)
```

Bravo ! Mettons maintenant à jour votre branche master.

3.3.8 Mettre à jour votre fork - master

Attention : assurez-vous d'avoir réalisé l'étape précédente avant celle-ci.

Vous devrez un jour mettre à jour votre branche master au sein de votre fork. Faites ainsi :

— Avec Git Bash ou votre terminal, positionnez-vous dans votre dossier mviewer (dossier récupéré via le clone) :

```
cd C:\Users\jean\Documents\git\mviewer
```

— Vérifiez que vous avez bien un upstream qui pointe vers https://github.com/mviewer/mviewer.git (voir l'étape précédente).

— Positionnez vous sur la branche master :

```
git checkout origin/master
```

— Synchronisez vous avec la source distante :

```
git fetch upstream
```

— Remplacez votre branche master (origin) par celle de mviewer (upstream) :

```
git reset --hard upstream/master
```

— Poussez ensuite ce code récupéré depuis mviewer (upstream) vers votre branche master (origin) :

```
git push origin master --force
```

3.3.9 Organisation des fichiers de carte

Rgèle générale

Ne **JAMAIS** modifier les fichiers du coeur.

Les fichiers du coeur sont tous les fichiers que vous obtenez nativement avec un clone de départ.

Nous vous recommandons d'intégrer la structure décrite dans cette section afin de simplifier vos manipulations de fichier :

— Créer un répertoire « apps » à la racine du mviewer.
— Positionner tous les fichiers de configuration XML à la racine du répertoire apps :

```
Exemple : C:\Users\jean\Documents\git\mviewer\apps\ma_carte.xml
```

— Créer un dossier par fichier de configuration que nous appellerons « dossiers de carte » :

```
Exemple : C:\Users\jean\Documents\git\mviewer\apps\ma_carte\
```

— Pour chaque dossier de carte, vous devrez créer les dossiers : templates, customcontrols, customlayers, data, sld, css, img.

Pour notre fichier de config « ma_carte.xml », nous aurons donc cette structure :

```
/apps
├── ma_carte.xml
└── ma_carte
    ├── customcontrols
    ├── customlayers
    ├── data
    ├── css
    ├── sld
    ├── img
    └── templates
```

Vous placerez dans ces dossiers les données (geojson), les customcontrols (js), les cunstomlayers (js) ainsi que les template mustache (js). Vous prendrez en compte la localisation de ces fichiers dans le fichier de configuration XML en donnant les bons chemins d'accès.

3.3.10 Organisation des autres fichiers

— Créer un répertoire « common » à la racine du répertoire « apps » (/apps/common/)
— Créer un dossier js, css, img, lib
— Créer un dossier basemaps, logo, legend, credit dans /img (/apps/common/img/)

On obtiendra donc cette structure :

```
/apps
├── common
    └── js/
```

(suite sur la page suivante)

(suite de la page précédente)

```
    ├── css/
    ├── lib/
    └── img/
        ├── legend/
        ├── logo/
        ├── credit/
        └── basemap/
├── ma_carte.xml
└── ma_carte
    ├── customcontrols
    ├── customlayers
    ├── data
    └── templates
```

Vous placerez tous les fichiers que vous avez créés ou modifiés dans ces dossiers au sein de /apps/common. Vous prendrez en compte la localisation de ces fichiers dans le fichier de configuration XML en donnant les bons chemins d'accès.

3.3.11 URL de carte

Il vous faudra prendre en compte le dossier « apps » dans vos URLs de carte ainsi :

```
http://kartenn.region-bretagne.fr/kartoviz/?config=apps/aide-droit-carte.xml
```

3.3.12 Addons

Si vous souhaitez enrichir vos cartes de fonctionnalités (isochrones, recherches, filtres temporels…) vous pouvez dupliquer cet addon dans tous les dossiers de carte.

Vous pouvez aussi créer un dossier « addons » dans le répertoire common et y ajouter la structure nécessaire (customlayers, customcontrols) pour être réexploitable par toutes les cartes :

Voici exemple d'organisation de fichier avec un addon « Isochrone » :

```
/apps
├── common
    └── js/
    ├── css/
    ├── lib/
    ├── addons/
        └── isochrone
            ├── customlayers
            ├── customcontrols
    └── img/
        ├── legend/
        ├── logo/
        ├── credit/
        └── basemap/
```

Le dossier « apps » étant votre dossier de travail, vous pouvez l'organiser selon vos besoins.

3.3.13 Participer à la communauté

Pour apporter une correction d'anomalie ou une évolution, nous vous recommandons d'aller à la la page « *Contribuer* ».

3.3.14 Bonnes pratiques de développements

Commits

Lorsque vous réalisez des commits, séparez les commits de style des commits de code.

Un commit de style comprend :
- Suppression / ajout d'un espace
- Suppression / ajout d'un saut de ligne
- Tout ce qui n'est pas du code

Un commit de code comprendra à l'inverse :
- Une modification sur une syntaxe
- Une modification sur une fonction
- Une modification sur un nom de variable
- Tout ce qui n'est pas du style

Encodage

L'encodage a utiliser est l'UTF-8.

Formatage

Lors de vos développements, inspectez le formatage du code initial :
- Le nombre d'espace pour indenter
- La présence d'espace avant et après les parenthèses
- La présence d'espace avant ou après les opérateurs logiques (==, <, >, ||, &&)
- Le nombre de saut de lignes avant ou après une fonction, un bloc de clode, etc…
- autres

Méfiez-vous de votre éditeur de code. Pensez à désactiver les plugins ou à configurer les règles de formatage.

Respectez ensuite ce formatage.

Commentaires

Un code commenté est un code compréhensible par tous. Nous recommandons très fortement d'utiliser les commentaires. Mieux vaut trop de commentaires que pas assez.
- Commentaires JavaScript :

```
// ceci est un commentaire sur une ligne
/* Ceci est
un commentaire sur plusieurs lignes*/
```

- Commentaire CSS :

```
/*Je suis un commentaire CSS*/
```

- Commentaire HTML :

```
<!--Je suis un commentaire HTML-->
```

Pour les fonctions ou méthodes JavaScript, nous recommandons d'ajouter en commentaire :
- Ce que fait cette fonction ou méthode
- Les paramètres en entrée
- Le résultat attendu et ce qui est retourné en sortie

Les indésirables

Nous déconseillons les affichages d'informations qui ne sont utiles qu'aux développeurs (console.log, alert…).

Editeur de code

Il n'y a pas d'obligation et vous êtes libre d'en choisir un.

— Notepadd++
— Sublime
— Visual Studio Code
— Atome
— autre

3.3.15 Informations Git & GitHub

Vous trouverez plus d'information sur la page « *Travailler avec Git et GitHub* ».

Vous trouverez notamment de la documentation dans la partie « *Documentation* ».

3.3.16 Documentation

1. Configurer un remote pour un fork
2. Maintien un fork avec l'upstream

3.4 Configurer - Application

Personnalisation de l'application

3.4.1 Syntaxe

```
<application
        id=""
        htmltitle=""
        title=""
        logo=""
        help=""
        addlayerstools=""
        showhelp=""
        titlehelp=""
        iconhelp=""
        style=""
        exportpng=""
        mapprint=""
        measuretools=""
        zoomtools=""
        initialextenttool=""
        stats=""
        statsurl=""
        coordinates=""
        coordinatestype=""
```

(suite sur la page suivante)

(suite de la page précédente)

```
        geoloc=""
        mouseposition=""
        togglealllayersfromtheme=""
        templaterightinfopanel=""
        templatebottominfopanel=""
        studio=""
        home=""
        mapfishurl=""
        lang=""
        langfile=""
        favicon=""
sortlayersinfopanel=""
/>
```

3.4.2 Paramètres principaux

- `title` *studio* : paramètre optionnel de type texte qui définit le titre de l'application. Valeur par défaut **mviewer**.
- `logo` *studio* : paramètre optionnel de type url qui définit l'emplacement du logo de l'application. Valeur par défaut **img/logo/earth-globe.svg**.
- `id` : identifiant de l'application. Il est utilisé dans l'extension filtre pour faire référence à l'application.
- `help` *studio* : paramètre optionnel de type url qui définit l'emplacement du fichier html de l'aide.
- `showhelp` *studio* : paramètre optionnel de type booléen (true/false) précisant si l'aide est affichée en popup au démarrage de l'application. Valeur par défaut **false**.
- `style` *studio* : paramètre optionnel de type url précisant la feuille de style à utiliser afin de modifier l'apparence de l'application (couleurs, polices…). Valeur par défaut **css/themes/default.css**. Voir : « *Configurer - Apparence* ».
- `exportpng` *studio* : paramètre optionnel de type booléen (true/false) activant l'export de la carte en png. Valeur par défaut **false**. Attention l'export ne fonctionne qu'avec des couches locales (même origine) ou avec des couches servies avec CORS activé.
- `mapprint` : paramètre optionnel de type booléen (true/false) activant l'impression de la vue courante depuis le navigateur. Valeur par défaut **false**.
- `measuretools` *studio* : paramètre optionnel de type booléen (true/false) activant les outils de mesure. Valeur par défaut **false**.
- `zoomtools` *studio* : paramètre optionnel de type booléen (true/false) activant les outils de zoom +/-. Valeur par défaut **true**.
- `initialextenttool` *studio* : paramètre optionnel de type booléen (true/false) activant le bouton de retour à l'étendue initiale. Valeur par défaut **true**.

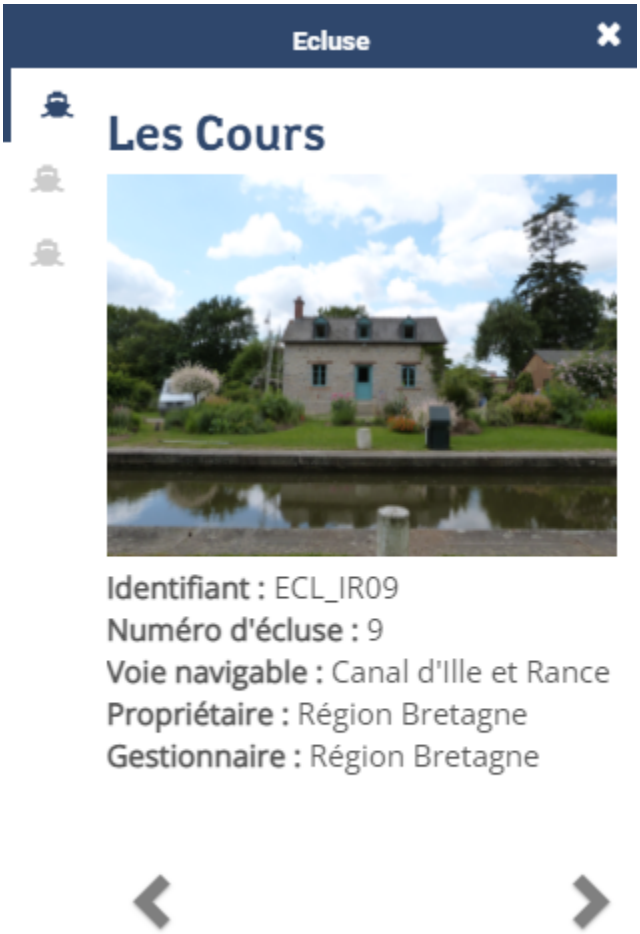
3.4.3 Paramètres secondaires

- `htmltitle` *studio* : optionnel de type texte, il permet d'utiliser du HTML uniquement pour le titre de l'application. Utiliser **title** avec ce paramètre pour le titre de l'onglet et la page de chargement. Il faut encoder pour une lecture en XML.
- `titlehelp` *studio* : paramètre optionnel de type texte qui définit le titre de la popup d'aide. Valeur par défaut **Documentation**.
- `iconhelp` *studio* : paramètre optionnel de type texte qui précise l'icône à utiliser afin d'illustrer la thématique. Le nom de l'icône doit être renseigné sous cette forme fab fa-apple ou fas fa-mobile. Les valeurs possibles sont à choisir parmi cette liste (cliquez sur l'icône souhaité pour obtenir la syntaxe) sur le site Fontawesome : https://fontawesome.com/v5/search?m=free
- `stats` : paramètre optionnel de type booléen (true/false) activant l'envoi de stats d'utilisation l'application. Valeur par défaut **false**.

— `statsurl` : paramètre optionnel de type url précisant l'url du service reccueillant les données d'utilisation de l'application (ip, application title, date). Ce service n'est pas proposé dans mviewer.
— `coordinates` *studio* : paramètre optionnel de type booléen (true/false) activant l'affichage des coordonnées GPS en degrés décimaux (navbar) lors de l'interrogation. Valeur par défaut **false**.
— `coordinatestype` : paramètre optionnel de type texte permettant de modifier l'unité des coordonnées affichés grâce à l'option coordinate. La valeur dms permet afficher les coordonnées en degrés sexagésimale (degré minute seconde).
— `geoloc` *studio* : paramètre optionnel de type booléen (true/false) activant la géolocalisation. Nécessite une connection **https**. Valeur par défaut **false**.
— `mouseposition` *studio* : paramètre optionnel de type booléen (true/false) activant l'affichage des coordonnées correspondant à la position de la souris. Les coordonnées sont affichées en bas à droite de la carte. Valeur par défaut **false**.
— `togglealllayersfromtheme` *studio* : Ajoute un bouton dans le panneau de gauche pour chaque thématique afin d'afficher/masquer toutes les couches de la thématique.Valeur : true/false. Valeur par défaut **false**.
— `templaterightinfopanel` : Template à utiliser pour le rendu du panneau de droite. Valeur à choisir parmi les templates de mviewer.templates.featureInfo (default|brut|accordion|allintabs). Valeur par défaut **default**.
— `templatebottominfopanel` : Template à utiliser pour le rendu du panneau du bas. Valeur à choisir parmi les templates de mviewer.templates.featureInfo (default|brut|accordion|allintabs). Valeur par défaut **default**.
— `studio` *studio* : Lien vers le mviewerstudio pour modifier la carte en cours.
— `home` *studio* : Lien vers le site parent de mviewer
— `mapfishurl` : Lien permettant d'afficher les couches courantes visibles vers un mapfishapp (geOrchestra) cible
— `hideprotectedlayers` : Indique si les couches protégées doivent être masquées dans l'arbre des thématiques lorsque l'utilisateur n'y a pas accès. Valeur : true/false (true par défaut).
— `lang` : Langue à utiliser pour l'interface. Passer « ?lang=en » dans l'url pour forcer la langue et ignorer la config. Par défaut, lang n'est pas activé. Le fichier mviewer.i18n.json contient les expressions à traduire dans différentes langues. Pour traduire le texte d'un élément html, il faut que cet élément dispose d'un attribut i18n=texte.a.traduire. En javascript la traduction s'appuie sur la méthode mviewer.tr(« texte.a.traduire »).
— `langfile` : URL du fichier de traduction supplémentaire à utiliser en complément de mviewer.i18n.json.
— `favicon` *studio* : URL du fichier image à utiliser comme favicon de l'application.
— `addlayerstools` : paramètre optionnel de type booléen (true/false) activant le panneau pour ajouter des couches WMS à la carte.
— `sortlayersinfopanel` : mode de tri des couches dans le panneau d'information en suivant la légende qui suit l'ordre des couches de la map (valeur **default**) ou la toc (valeur **toc**). Valeur par défaut **default**.

3.4.4 Modes de templates

Modes d'affichage des templates soit à droite (templaterightinfopanel), soit en bas (templatebottominfopanel).
— `default` : Une entrée par couche avec un carroussel pour navigation par entités.

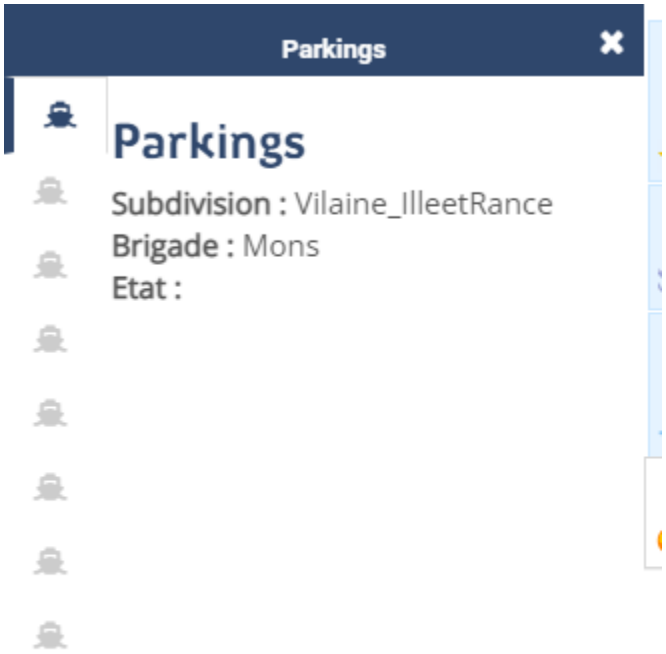


— `brut` : Affichage à la suite de toutes les entités avec encapsulation par couche.

Informations
Ecluse
Apigné
Identifiant : ECL_V03
Numéro d'écluse : 3
Voie navigable : Vilaine
Propriétaire : Région Bretagne
Gestionnaire : Région Bretagne
Parkings
Parkings
Propriétaire : Commune
Gestionnaire : Commune
Subdivision : Vilaine_IlleetRance
Brigade : Mons
Etat : Bon
Observations : + de 50 places

— `accordion` : Affichage par couche avec carroussel et pliage/dépliage lors du changement de couche.

— `allintabs` : Affichage à la suite avec une entrée par entité.



3.4.5 Exemple

```
<application title="Exemple"
        logo="img/logo/g.png"
        help="help/aide.html"
        exportpng="false"
        measuretools="true"
        favicon="https://www.bretagne.bzh/app/themes/bretagne/dist/img/icons/favicon.ico
↪"
        style="css/themes/blue.css"/>
```

3.5 Configurer - les options de la carte

3.5.1 Syntaxe

```
<mapoptions
maxzoom=""
projection=""
center=""
zoom=""
projextent=""
maxextent=""
```

(suite sur la page suivante)

(suite de la page précédente)

```
rotation=""
scalebar=""
scaleunits=""
scalesteps=""/>
```

3.5.2 Paramètres

- `maxzoom` : paramètre optionnel de type entier qui définit le seuil maximum de zoom de l'application. Valeur par défaut **19**. Plus d'info sur les seuils de zooms https://wiki.openstreetmap.org/wiki/Zoom_levels
- `projection` : paramètre obligatoire de type texte définissant la projection (code EPSG) utilisée par la carte. Exemple **EPSG :3857**.
- `center` *studio* : paramètre optionnel de type numérique définissant les coordonnées géographiques du centre de la carte dans la projection choisie. Exemple **-220750.13,6144925.57**.
- `zoom` *studio* : paramètre optionnel de type entier définissant le zoom initial de la carte. Valeur par défaut **8**.
- `projextent` : paramètre obligatoire de type texte définissant les limites géographiques de la projection utilisée. Ce paramètre n'est pas obligatoire pour les projections EPSG :4326 et EPSG :3857.
- `maxextent` *studio* : paramètre optionnel de type texte définissant les limites géographiques pour la vue cartographique
- `scalebar` : paramètre optionnel de type texte pour afficher l'échelle en barre et non en ligne (« false » par défaut).
- `scaleunits` : paramètre optionnel de type texte pour préciser l'unité de mesure de l'échelle (« metric » par défaut)
- `scalesteps` : paramètre optionnel de type texte pour préciser le nombre de pas de l'échelle (« 2 » par défaut).
- `scaletext` : paramètre optionnel de type texte pour préciser si on souhaite afficher le texte au dessus d'une échelle en barre (« true » par défaut).

3.5.3 Exemple

```
<mapoptions
      scalebar="true"
      scaletext="true"
      scaleunits="metric"
      scalesteps="3"
      maxzoom="18"
      projection="EPSG:3857"
      center="-161129,6140339"
      zoom="9"
      projextent="-20037508.342789244, -20037508.342789244, 20037508.342789244,␣
↪20037508.342789244"
      maxextent="-550346.603653, 5975541.123222, -45250.720745, 6262944.349574" />
```

Astuce : La carte dispose également d'un marker qui s'affiche au clic sur la carte. Il est possible de modifier ce marker via un paramétrage dans « *Configurer - La recherche* ».

```
<searchparameters imgurl='img/map_marker.png' imgwidth='50px' svgcolor='orange'
```

3.6 Configurer - Les fonds de carte

3.6.1 Paramètres généraux <Baselayers>

Paramétrages de la liste des fonds de carte.

Syntaxe

```
<baselayers style="">
    <baselayer />
    <baselayer />
</baselayers>
```

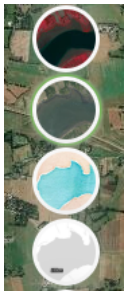
Paramètre

— `style` *studio* : paramètre optionnel de type texte à choisir parmi (default/gallery) et définissant le style du contrôle permettant de changer de fond de carte. Valeur par défaut **default**.

Le mode **« default »** active le contrôle ci-dessous. Le fond de carte affiché dans la vignette est celui qui s'affichera dans la carte après un clic.

Le mode **« gallery »** active une liste à deux états :

un état déplié lors du premier clic affichant tous les fonds de carte disponibles :



un état replié lors du second clic (choix du fond de carte à afficher) :

3.6.2 Paramètre par fond de cat <baselayer>

Elément enfant de <baselayers> permettant le paramétrage de chaque fond de carte.

Syntaxe

```
<baselayer type=""
        owsoptions=""
        id=""
        label=""
```

(suite sur la page suivante)

(suite de la page précédente)

```
        title=""
        maxscale=""
        thumbgallery=""
        url=""
        layers=""
        format=""
        visible=""
        fromcapacity=""
        attribution=""
        style=""
        styleurl=""
        matrixset=""
        maxzoom=""
        opacity=""
/>
```

Paramètres principaux

- `type` : paramètre obligatoire de type texte qui définit le type de la couche. Les options sont OSM, WMTS, WMS, vector-tms et fake. Fake permet de disposer d'un fond vierge. C'est alors le motif ou la couleur du fond de l'application qui s'affiche.
- `id` : paramètre obligatoire de type texte pour attribuer un identifiant unique et interne à la couche
- `label` : paramètre obligatoire de type texte pour définir le nom du fond de carte
- `title` : paramètre obligatoire de type texte pour définir le sous-titre du fond de carte. Utilisé avec le mode « gallery »
- `thumbgallery` : paramètre obligatoire de type url permettant de sélecionner l'imagette à associer au fond de carte.
- `url` : paramètre obligatoire de type url définissant l'URL du service web OSM, WMTS, WMS ou vector-tms.
- `styleurl` : paramètre optionnel de type url définissant le fichier de style au format JSON à utiliser. (Obligatoire pour les couches de type vector-tms)
- `layers` : paramètre optionnel de type texte définissant l'identifiant technique de la couche. (Obligatoire pour les couches de type WMS et WMTS)
- `format` : paramètre optionnel de type texte définissant le Format d'image retourné par le serveur. (Obligatoire pour les couches de type WMS et WMTS)
- `visible` *studio* : paramètre obligatoire de type booléen (true/false) précisant si la couche est visible au démarrage. Il s'agit d'un paramètre exclusif. Une seule couche de fond peut être affichée sur la carte. Attention un baseselayer et un seul doit disposer du paramètre visible= »true ».
- `attribution` : paramètre obligatoire alimentant le contrôle attributions de la carte (©).
- `style` : paramètre optionnel précisant le style à associer à la couche. (Obligatoire pour les couches de type WMTS et vector-tms. Pour le type vector-tms, le style correspond à la valeur indiquée en tant que première clé de la propriété « sources » du fichier de style au format JSON).

Paramètres secondaires

- `owsoptions` : pour une couche WMS, permet de forcer certains paramètres des requêtes GetMap. Exemple : « VERSION :1.3.0 ».
- `maxscale` : paramètre optionnel définissant l'échelle max du fond de carte.
- `fromcapacity` : paramètre optionnel de type booléen (true/false)spécifique aux fonds de carte WMTS. Permet la construction de la couche à partir des capacités du service WMTS.
- `matrixset` : paramètre optionnel précisant le style à associer à la couche. Paramètre obligatoire pour les couches de type WMTS si le paramètre **fromcapacity** n'est pas activé.
- `maxzoom` : paramètre optionnel de type numérique définissant le zoom maximum pour la couche (pas géré pour le type vector-tms).
- `opacity` : opacité du fond de carte . Valeur numérique de 0 à 1. Défaut = 1.

Exemple

```
<baselayer
        type="OSM"
        id="osm1"
        label="OpenStreetMap"
        title="OpenSTreetMap"
        thumbgallery="img/basemap/osm.png"
        url="http://{a-c}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution="Données : les contributeurs d'&lt;a href='http://www.openstreetmap.org/' target='_blank'>OpenStreetMap &lt;/a>,  &lt;a     href='http://www.openstreetmap.org/copyright' target='_blank'>ODbL &lt;/a>"
        visible="true"/>
```

3.7 Configurer - Les thématiques

La configuration des thématiques qui seront visibles dans l'interface se fait dans le fichier **config.xml**.

3.7.1 Principe général

Ici, les *couches* <*configlayers*> sont organisées de la manière suivante :

- une **couche** (layer) est intégrée à un groupe ou à un thème,
- un **groupe** est intégré à un thème et peut contenir entre 1 et *n* couches,
- un **thème** est intégré au parent « themes » et peut contenir entre 1 et n groupes ainsi que 1 et *n* couches,
- le parent « **themes** » peut contenir entre 1 et *n* thème,

Ceci peut être résumé avec l'arborescence suivante :

```
<themes>
        <theme>
                <layer> </layer>
        </theme>
        <theme>
                <group>
                        <layer> </layer>
                        <layer> </layer>
                </group>
        </theme>
</themes>
```

3.7.2 Structure par défaut

Par défaut, le contenu suivant est proposé :

```
<themes>
        <theme name="Population"  collapsed="false" id="habitant" icon="group">
                <layer id="rp_struct_pop_geom" name="Densité de population (hab/km²)" visible="false" tiled="false"
                searchable="false" queryable="false" attributefilter="true" attributefield="level" attributevalues="Commune,EPCI,Pays"  attributelabel="Échelle" attributestylesync="true" attributefilterenabled="true" infopanel="bottom-panel" infoformat="application/vnd.ogc.gml" featurecount="5" timefilter="true" timeinterval=
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪"year" timecontrol="slider" timemin="1999" timemax="2013" timevalues="1999,2008,2013"␣
↪style="rphab_densite@commune" stylesalias="" url="http://ows.region-bretagne.fr/
↪geoserver/rb/wms" attribution="Sources: INSEE (RP) - OpenStreetMap | Traitements:␣
↪Région Bretagne - Service connaissance, observation, planification et prospective"␣
↪metadata="http://kartenn.region-bretagne.fr/geonetwork/?uuid=26324529-e0b7-450c-9506-
↪2dcdca608f5f" metadata-csw="http://kartenn.region-bretagne.fr/geonetwork/srv/eng/csw?
↪SERVICE=CSW&amp;VERSION=2.0.2&amp;REQUEST=GetRecordById&amp;elementSetName=full&amp;
↪ID=26324529-e0b7-450c-9506-2dcdca608f5f">
5               </layer>
6       </theme>
7       <theme name="Environnement"  collapsed="false" id="environnement" icon="leaf">
8               <layer id="reserve_naturelle_regionale" name="Réserves naturelles␣
↪régionales"  visible="false" tiled="false" searchable="false" queryable="true" fields=
↪"axe" aliases="axe" infoformat="application/vnd.ogc.gml" featurecount="20" sld="http://
↪kartenn.region-bretagne.fr/styles/reserve_naturelle.sld" url="http://ows.region-
↪bretagne.fr:80/geoserver/rb/wms" legendurl="http://kartenn.region-bretagne.fr/doc/
↪icons_region/reserve_naturelle.svg" attribution="Source: Région Bretagne" metadata=
↪"https://geobretagne.fr/geonetwork/apps/georchestra/?uuid=77f8fc52-ae57-41d1-8f08-
↪7b121b013f51" metadata-csw="https://geobretagne.fr/geonetwork/srv/eng/csw?SERVICE=CSW&
↪amp;VERSION=2.0.2&amp;REQUEST=GetRecordById&amp;elementSetName=full&amp;ID=77f8fc52-
↪ae57-41d1-8f08-7b121b013f51" >
9               <template url="templates/global.reserve_naturelle_reg.mst"></template>
10              </layer>
11      </theme>
12      <theme name="Éducation"  collapsed="false" id="education" icon="graduation-cap">
13              <layer id="lycee" name="Lycées"  visible="false" tiled="false"␣
↪searchable="false" queryable="true" fields="axe" aliases="axe" attributefilter="true"␣
↪attributefield="secteur_li" attributevalues="Public,Privé sous contrat avec l
↪'éducation nationale"  attributelabel="Filtre" attributestylesync="false"␣
↪attributefilterenabled="false" infoformat="application/vnd.ogc.gml" featurecount="20"␣
↪sld="http://kartenn.region-bretagne.fr/styles/lycee_secteur.sld" url="http://ows.
↪region-bretagne.fr/geoserver/rb/wms" attribution="Source: Région Bretagne" metadata=
↪"http://kartenn.region-bretagne.fr/geonetwork/?uuid=99e78163-ce9a-4eee-9ea0-
↪36afc2a53d25" metadata-csw="http://kartenn.region-bretagne.fr/geonetwork/srv/eng/csw?
↪SERVICE=CSW&amp;VERSION=2.0.2&amp;REQUEST=GetRecordById&amp;elementSetName=full&amp;
↪ID=99e78163-ce9a-4eee-9ea0-36afc2a53d25" >
14              <template url="templates/global.lycee.mst"></template>
15              </layer>
16      </theme>
17      <theme name="Transports"  collapsed="false" id="transport" icon="bus">
18              <group name="Transport ferroviaire" id="grp1" >
19                      <layer id="troncon_ferroviaire" name="Lignes ferroviaires" ␣
↪visible="false" tiled="false" searchable="false" queryable="true" fields="axe" aliases=
↪"axe" infoformat="application/vnd.ogc.gml" featurecount="20" style="ligne_ferroviaire_
↪defaut" stylesalias="Par défaut" url="http://ows.region-bretagne.fr/geoserver/rb/wms"␣
↪attribution="Source: Région Bretagne" metadata="http://kartenn.region-bretagne.fr/
↪geonetwork/?uuid=0da27e88-4da6-423e-ba4c-dbcad9128cd2" metadata-csw="http://kartenn.
↪region-bretagne.fr/geonetwork/srv/eng/csw?SERVICE=CSW&amp;VERSION=2.0.2&amp;
↪REQUEST=GetRecordById&amp;elementSetName=full&amp;ID=0da27e88-4da6-423e-ba4c-
↪dbcad9128cd2">
20                      <template url="templates/transport.ligne_ferroviaire.mst"></
↪template>
```

(suite sur la page suivante)

(suite de la page précédente)

```
21                          </layer>
22                          <layer id="arret_ferroviaire" name="Arrêts ferroviaires␣
↪régionaux"  visible="false" tiled="false" searchable="true" queryable="true" fields=""␣
↪aliases="" infoformat="application/vnd.ogc.gml" featurecount="20" style="arret_
↪ferroviaire_defaut, arret_ferroviaire_nature" stylesalias="Par défaut,Nature des␣
↪arrêts ferroviaires" legendurl="http://kartenn.region-bretagne.fr/doc/icons_region/
↪gare_ter.svg" url="http://ows.region-bretagne.fr/geoserver/rb/wms" attribution=
↪"Source: Région Bretagne" metadata="http://kartenn.region-bretagne.fr/ geonetwork/?
↪uuid=4a9d13f7-17be-4a98-9f8f-907cf223072f" metadata-csw="http://kartenn.region-
↪bretagne.fr/geonetwork/srv/eng/csw?SERVICE=CSW&amp;VERSION=2.0.2&amp;
↪REQUEST=GetRecordById&amp;elementSetName=full&amp;ID=4a9d13f7-17be-4a98-9f8f-
↪907cf223072f" >
23                          <template url="templates/global.arret_ferroviaire.mst"></
↪template>
24                          </layer>
25                  </group>
26                  <group name="Transport maritime" id="grp2" >
27                          <layer id="gare_maritime" name="Gares maritimes"  visible="false
↪" tiled="false" searchable="false" queryable="true" fields="axe" aliases="axe"␣
↪infoformat="application/vnd.ogc.gml" featurecount="20" sld="http://kartenn.region-
↪bretagne.fr/styles/gare_maritime.sld"
28                          url="https://geobretagne.fr/geoserver/dreal_b/ows" legendurl=
↪"http://kartenn.region-bretagne.fr/doc/icons_region/gare_maritime.svg" attribution=
↪"Source: DREAL Bretagne"
29                          metadata="https://geobretagne.fr/geonetwork/apps/ georchestra/?
↪uuid=ffcb4e72-a01b-44f0-8da3-95a5b13c6e42" metadata-csw="https://geobretagne.fr/
↪geonetwork/srv/eng/csw?SERVICE=CSW&amp;VERSION=2.0.2&amp;REQUEST=GetRecordById&amp;
↪elementSetName=full&amp;ID=ffcb4e72-a01b-44f0-8da3-95a5b13c6e42" >
30                          <template url="templates/global.gare_maritime.mst"></template>
31                          </layer>
32                          <layer id="port" name="Ports"  visible="false" tiled="false"␣
↪searchable="false" queryable="true" fields="axe" aliases="axe" infoformat="application/
↪vnd.ogc.gml" featurecount="20" sld="http://kartenn.region-bretagne.fr/styles/port.sld"␣
↪url="http://ows.region-bretagne.fr:80/geoserver/rb/wms" legendurl="http://kartenn.
↪region-bretagne.fr/doc/icons_region/port.svg" attribution="Source: Région Bretagne"␣
↪metadata="https://geobretagne.fr/geonetwork/apps/georchestra/?uuid=c55c4fba-6a37-48ea-
↪8754-a1bf770a684b" metadata-csw="https://geobretagne.fr/geonetwork/srv/eng/csw?
↪SERVICE=CSW&amp;VERSION=2.0.2&amp;REQUEST=GetRecordById&amp;elementSetName=full&amp;
↪ID=c55c4fba-6a37-48ea-8754-a1bf770a684b" >
33                          <template url="templates/global.port.mst"></template>
34                          </layer>
35                  </group>
36          </theme>
37          <theme name="Découpages territoriaux"  collapsed="true" id="territoire" icon=
↪"globe">
38                  <layer id="commune" name="Commune" visible="false" queryable="false"␣
↪fields="nom_geo" aliases="Nom" type="customlayer" style="" opacity="1" legendurl="img/
↪legend/commune.png" url="customlayers/commune.js" tooltip="true" attribution="Source:␣
↪GéoBretagne" metadata="https://geobretagne.fr/geonetwork/apps/ georchestra/?
↪uuid=b08e6cb1-236c-49b7-88f9-42b500d274d5" metadata-csw="https://geobretagne.fr/
↪geonetwork/srv/eng/csw?SERVICE=CSW&amp;VERSION=2.0.2&amp;REQUEST=GetRecordById&amp;
↪elementSetName=full&amp;ID=b08e6cb1-236c-49b7-88f9-42b500d274d5"/>
```

(suite sur la page suivante)

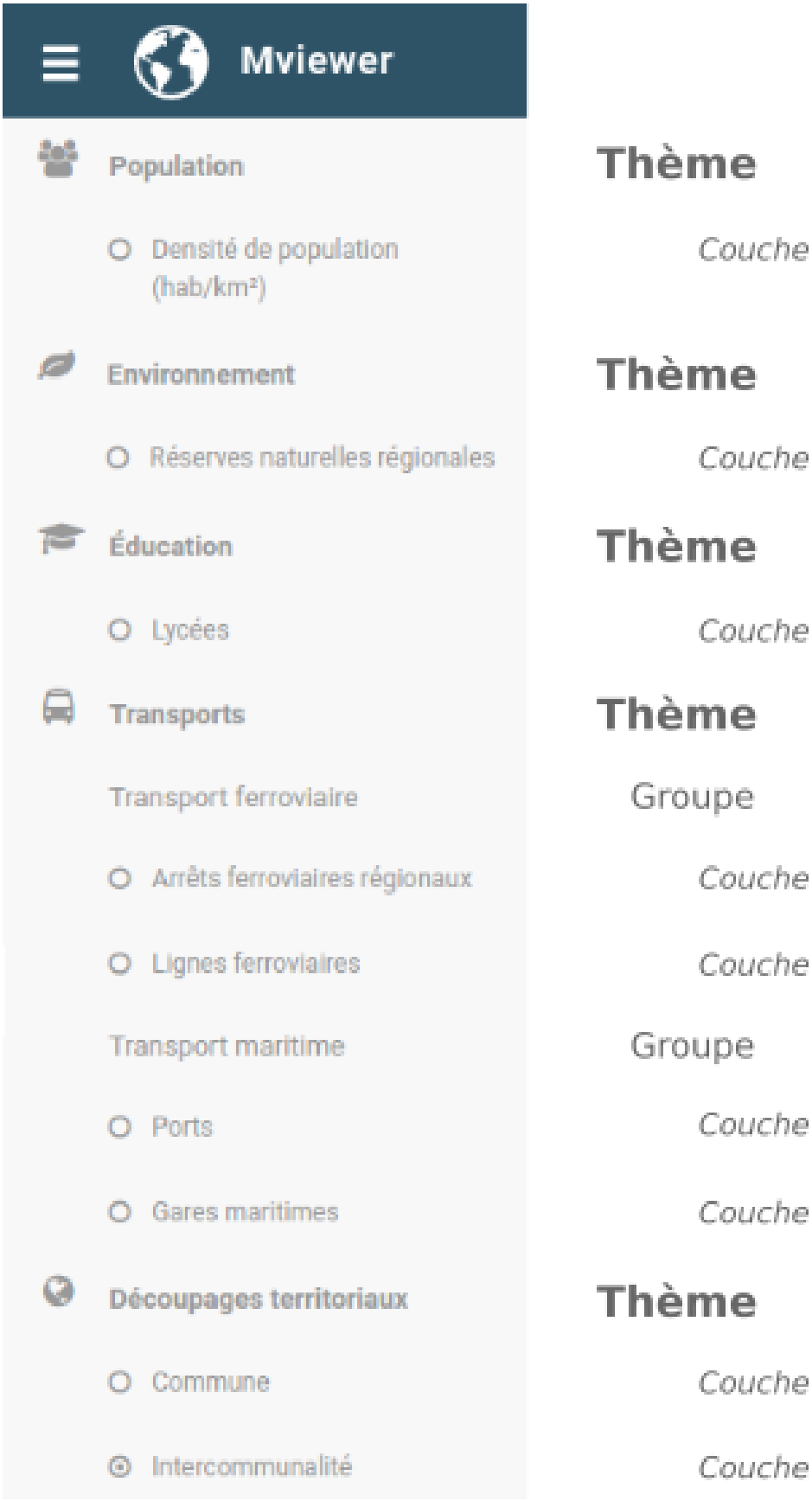
(suite de la page précédente)

```
                <layer id="epci" name="Intercommunalité" visible="true" queryable="false
↪" fields="nom_geo" aliases="Nom" customcontrol="true" type="customlayer" style=""␣
↪opacity="1" legendurl="img/legend/epci.png" url="customlayers/epci.js" tooltip="true"␣
↪tooltipenabled="true" attribution="Source: GéoBretagne" metadata="https://geobretagne.
↪fr/geonetwork/apps/ georchestra/?uuid=2298d744-49cb-4fcb-9487-26f916fecdff" metadata-
↪csw="https://geobretagne.fr/geonetwork/srv/eng/csw?SERVICE=CSW&amp;VERSION=2.0.2&amp;
↪REQUEST=GetRecordById&amp;elementSetName=full&amp;ID=2298d744-49cb-4fcb-9487-
↪26f916fecdff"/>
        </theme>
</themes>
```

Ici, ce code .xml peut être résumé à l'arborescence suivante :

```
<themes>
        <theme name="Population">
                <layer name="Densité de population (hab/km²)"> </layer>
        </theme>
        <theme name="Environnement">
                <layer name="Réserves naturelles régionales"> </layer>
        </theme>
        <theme name="Éducation">
                <layer name="Lycées"> </layer>
        </theme>
        <theme name="Transports">
                <group name="Transport ferroviaire">
                        <layer name="Lignes ferroviaires"> </layer>
                        <layer name="Arrêts ferroviaires régionaux"> </layer>
                </group>
                <group name="Transport maritime">
                        <layer name="Gares maritimes"> </layer>
                        <layer name="Ports"> </layer>
                </group>
        </theme>
        <theme name="Découpages territoriaux">
                <layer name="Commune"> </layer>
                <layer name="Intercommunalité"> </layer>
        </theme>
</themes>
```

Ce qui donne visuellement ceci :


Mviewer
Population
Thème
Densité de population (hab/km²)
Couche
Environnement
Thème
Réserves naturelles régionales
Couche
Éducation
Thème
Lycées
Couche
Transports
Thème
Transport ferroviaire
Groupe
Arrêts ferroviaires régionaux
Couche
Lignes ferroviaires
Couche
Transport maritime
Groupe
Ports
Couche
Gares maritimes
Couche
Découpages territoriaux
Thème
Commune
Couche
Intercommunalité
Couche

3.7.3 Configuration de la liste des thèmes

Syntaxe <themes>

```
<themes mini="" legendmini="" />
```

Paramètres

- `mini` : paramètre optionnel de type booléen (true/false) qui précise si le panneau de gauche est réduit à l'ouverture de l'application. L'attribut `collapsed` des <theme> doit être à true pour que cet attribut soit pris en compte. Défaut = false.
- `legendmini` : paramètre optionnel de type booléen (true/false) qui précise si le panneau de la légende est réduit à l'ouverture de l'application. Défaut = true.

Syntaxe <theme>

Elément enfant de <themes>

```
<theme name=""  collapsed="" id="" icon="" url="" layersvisibility=""/>
```

Paramètres

- `name` *studio* : paramètre obligatoire de type texte qui précise le nom de la thématique.
- `id` *studio* : paramètre obligatoire de type texte qui affecte un identifiant unique interne à la thématique.
- `collapsed` *studio* : paramètre optionnel de type booléen (true/false) qui précise si la thématique est fermée au démarrage de l'application. Pour que la thématique soit ouverte au démarrage, il faut choisir l'option **false**. Attention, il ne peut y avoir qu'une thématique ayant ce paramètre à false. Valeur par défaut **true**.
- `icon` *studio* : paramètre optionnel de type texte qui précise l'icône à utiliser afin d'illustrer la thématique. Les valeurs possibles sont à choisir parmi cette liste sur le site Fontawesome : https://fontawesome.com/v5/search?m=free. Une autre possibilité est d'uliser une classe CSS personnelle mobilisant une image. Il faut alors mettre la classe précédée d'un point comme valeur. exemple « .mycustomicon ».
- `url` *studio* : paramètre optionnel de type url(URL vers fichier xml) qui permet de récupérer une thématique complète depuis un config.xml externe.
- `layersvisibility` *studio* : paramètre optionnel (all/none/default) qui précise la visibilité des couches dans la thématique externe. Valeur par défaut **default**.

Voici un exemple d'icône personnalisée :

```
.mycustomicon::before {
content: '';
display: inline-block;
height: 1em;
width: 1em;
background: url(../img/logo/worldwide.svg) no-repeat scroll;
background-size: 16px 16px;
margin-top: 9px;
}
```

- `url` *studio* : Des thèmes externes (présents dans d'autres configurations peuvent être automatiquement chargés par référence au fichier xml utilisé (url=) et à l'id de la thématique (id=). Attention si la configuration externe est sur un autre domaine, il faut alors que mviewer utilise un proxy Ajax ou alors s'assurer que CORS est activé sur le serveur distant. Les thématiques externes peuvent utiliser des ressources particulières (templates, customLayer, sld…) si les URLs de ces ressources sont absolues et accessibles.

Syntaxe <group>

Elément enfant de <theme>

```
<group name="" id="" />
```

Paramètres

- name : paramètre obligatoire de type texte qui précise le nom du groupe.
- id : paramètre obligatoire de type texte qui précise l'identifiant unique du groupe.

3.8 Configurer - Les couches

3.8.1 Syntaxe <layer>

Élément enfant de <theme> ou <group>

```
<layer   id=""
     name=""
     scalemin=""
     scalemax=""
     visible=""
     owsoptions=""
     tiled=""
     queryable=""
     fields=""
     aliases=""
     fieldsjson=""
     type=""
     filter=""
     filterstyle=""
     searchable=""
     searchid=""
     fusesearchkeys=""
     fusesearchresult=""
     useproxy=""
     secure=""
     authentification=""
     authorization=""
     toplayer=""
     exclusive=""
     infoformat=""
     infohighlight=""
     featurecount=""
     style=""
     styleurl=""
     styletitle=""
     stylesalias=""
     timefilter=""
     timeinterval=""
     timecontrol=""
     timevalues=""
```

(suite sur la page suivante)

(suite de la page précédente)

```
        timemin=""
        timemax=""
        attributefilter=""
        attributefield=""
        attributevalues=""
        attributeoperator=""
        attributestylesync=""
        attributelabel=""
        attributefilterenabled=""
        opacity=""
        legendurl=""
        dynamiclegend=""
        vectorlegend=""
        nohighlight=""
        url=""
        attribution=""
        tooltip=""
        tooltipcontent=""
        tooltipenabled=""
        showintoc=""
        expanded=""
        metadata=""
        metadata-csw=""
        infopanel=""
        index=""
        minzoom=""
        maxzoom=""
        top=""
        defaultSensor=""
        selector=""
        datastreamsfilter=""
        multidatastreamsfilter="">
        <template url=""></template>
</layer>
```

Paramètres obligatoires pour une configuration minimaliste

— `name` *studio* : paramètre de type texte qui précise le nom de la couche.
— `url` *studio* : paramètre de type URL (URL du service web).
— `id` *studio* : paramètre de type texte qui renseigne l'identifiant technique de la couche côté serveur WMS ou WFS.

Paramètres pour gérer l'affichage de la couche

- `visible` *studio* : Booléen stipulant si la couche est visible par défaut.
- `opacity` *studio* : Opacité de la couche (1 par défaut).
- `tiled` *studio* : Booléen stipulant si on désire un affichage tuilé de la couche. Très utile pour affichage de grosses couches.
- `style` *studio* : Style(s) de la couche. Si plusieurs styles , utiliser la virgule comme séparateur. Si la couche est de type wms, il faut faire référence à un style sld. Si la couche est de type geojson, il faut faire référence à un style définit dans lib/featurestyles.js. Si la couche est de type vector-tms, le style correspond à la valeur indiquée en tant que première clé de la propriété « sources » du fichier de style au format JSON. Si la couche est de type customlayer, le style n'est pas défini ici.
- `styleurl` *studio* : pour les couches de type vector-tms uniquement, il indique l'URL vers le fichier de style au format JSON.
- `styletitle` *studio* : Titres à utiliser pour la liste des styles associés.
- `stylesalias` *studio* : Titres à utiliser pour chaques style. utiliser la virgule comme séparateur si plusieurs styles.
- `sld` *studio* : Lien vers un SLD stocké sur le web. Dans ce fichier SLD, la balise sld :Name contenue dans sld :NamedLayer doit être égale au nom de la couche sans mention du namespace. Exemple <sld :Name>aeroports</sld :Name>. Si plusieurs styles , utiliser la virgule comme séparateur. S'applique uniquement aux layers WMS. Il faut indiquer l'URL résolvable par le serveur WMS du ou des sld.
- `index` *studio* : Ordre d'affichage de la couche sur la carte et dans la légende au démarrage. Les couches avec ce paramètre seront visibles sous les toplayers. Les couches sans ce paramètre ni toplayer seront affichées dans l'ordre d'écriture dans le XML.
- `scalemin` *studio* : Échelle minimum de la couche.
- `scalemax` *studio* : Échelle maximum de la couche.
- `dynamiclegend` *studio* : Booléen précisant si la légende est liée à l'échelle de la carte et si elle nécessite d'être actualisée à chaque changement d'échelle de la carte.
- `exclusive` *studio* : Booléen stipulant si la couche est exclusive. Si la valeur est « true », l'affichage de cette couche masquera automatiquement toutes les autres couches ayant ce paramètre activé.
- `legendurl` *studio* : url permettant de récupérer la légende. Si non défini, c'est un GetLegendGraphic qui est effectué.
- `filter` *studio* : Expression CQL permettant de filtrer la couche ex : insee=35000 Ou INTERSECT(the_geom, POINT (-74.817265 40.5296504)) [tutorial] (http://docs.geoserver.org/stable/en/user/tutorials/cql/cql_tutorial.html#cql-tutorial).
- `filterstyle` *studio* : pour les couches de type vector-tms uniquement. Il permet de ne pas conserver, dans le style, la représentation de certaines couches. Cela permet donc de ne pas représenter un type de données présent dans le flux tuilé vectoriel. Il faut indiquer ici le nom d'une ou de plusieurs couches référencées dans la propriété « source-layer » du fichier de style au format JSON. Lorsque plusieurs couches sont à ajouter, le séparateur est la virgule et sans espace.
- `toplayer` *studio* : Précise si la couche demeure figée. Booléen. Défaut = true. Si plusieurs couches sont en toplayer, elles seront affichées dans l'ordre d'écriture du XML.
- `expanded` *studio* : Booléan précisant si le panneau de la couche est agrandi au démarrage. La valeur par défaut est false.
- `showintoc` *studio* : Booléen stipulant si la couche est affichée dans la légende. La valeur par défaut est true.
- `minzoom` : pour les couches de type vector-tms, la valeur correspond au niveau de zoom minimal de visibilité de la couche. Par défaut, la valeur est récupérée à partir du fichier de style au format JSON. Pour plus de détail, voir la documentation Openlayers.
- `maxzoom` : pour les couches de type vector-tms, la valeur correpond au niveau de zoom maximal de visibilité de la couche. Par défaut, la valeur est récupérée à partir du fichier de style au format JSON. Pour plus de détail, voir la documentation Openlayers.

Paramètres pour gérer attributions et métadonnées

— `attribution` *studio* : Copyright de la couche. Le mot-clé « metadata » permet de récupérer cette information depuis des métadonnées compliantes au Dublin Core (champs « source »).
— `metadata` *studio* : Lien vers la fiche de metadonnées complète.
— `metadata-csw` *studio* : Requête CSW pour l'affiche dans la popup du détail de la couche. Mviewer récupère également la date de création ou dernière mise à jour si cela est en Dublin Core.

Paramètres pour gérer l'interrogation et la mise en forme de la fiche d'interrogation de la couche

— `queryable` *studio* : Booléen stipulant est ce que la couche est intérrogeable via un GetFeatureInfo.
— `infoformat` *studio* : Format du GetFeatureInfo. 2 formats sont supportés : text/html et application/vnd.ogc.gml. Le format application/vnd.ogc.gml est demandé pour l'utilisation de templates.
— `infohighlight` : Booléen précisant si les features de la couche sont mises en surbrillance en interrogeant leurs informations, défaut = true. Si false un markeur est affiché. Les styles utilisés pour la mise en surbrillance peuvent être configurés (voir « *Configurer - Les styles utilisés pour la mise en surbrillance des entités sélectionnées* »).
— `featurecount` *studio* : Nombre d'éléments maximun retournés lors de l'interrogation.
— `fields` *studio* : Si les informations retournées par l'interrogation est au format GML, fields représente les attributs à parser pour générer la vignette.
— `aliases` *studio* : Si les informations retournées par l'interrogation est au format GML, aliases représente le renommage des champs parsés.
— `fieldsjson` : Liste des champs de type json. Avec ce paramètre, mviewer parse le contenu des champs spécifiés en JSON, ce qui permet ensuite d'exploiter ces valeurs dans des boucles de templates mustache pour afficher une liste, un tableau…

Paramètres pour gérer la recherche

— `searchable` : Booléen précisant si la couche est interrogeable via la barre de recherche.
— `searchengine` : Moteur de recherche utilisé entre elasticsearch et fuse. Défault=elasticsearch.
— `searchid` : Nom du champ à utiliser côté WMS afin de faire le lien avec l'_id elasticsearch.
— `iconsearch` : Lien vers l'image utilisée pour illustrer le résultat d'une recherche elasticsearch.
— `fusesearchkeys` : Chaîne de caractères contenant la liste des champs de la couche à indexer pour la recherche. Les noms des champs doivent être séparés par des virgules. À n'utiliser que si searchengine = fuse.
— `fusesearchresult` : Chaîne de caractères décrivant l'information à afficher dans les résultats de recherche. Cette chaîne contient soit le nom d'un champ de la couche soit un template Mustache combinant plusieurs noms de champs. Exemple : « {{name}} ({{city}}) ». A n'utiliser que si searchengine = fuse.

Paramètre pour le type de couche sensorthings

— `url` : URI du service sensorthing.
— `style` : Style à utiliser (sensorPolygon|sensorPoint) disponible dans le fichier `js/featurestyles.js`.
— `top` : Nombre d'observations à retourner. La limite est par défaut définie par les capacités du serveur publiant le service.
— `defaultSensor` : Stream par défaut à afficher au clic. Valeur `null` par défaut.
— `selector` : Query param `select` tel que `Locations(33)/Things?$select=id,description` pour filtrer les champs `Things` à retourner.
— `datastreamsfilter` : Query param `select` tel que `Datastreams($select=name,id)` pour filtrer les champs `Datastreams` à retourner.
— `multidatastreamsfilter` : Query param `select` tel que `MultiDatastreams($select=name,id)` pour filtrer les champs `MultiDatastreams` à retourner.

Plus d'informations sur le type de couche OGC SensorThings :

Note : Documentation utile : Spécifications OGC Schéma SensorThings API

Paramètres pour les couches non WMS

- `type` : Type de la couche (wms|geojson|kml|vector-tms|sensorthings|customlayer|import) default=wms. Si customlayer est défini, il faut instancier un Layer OpenLayers dans un fichier javascript ayant pour nom l'id de la couche (voir « *Configurer - Une recherche Fuse* »). Ce fichier js doit être placé dans le répertoire customlayers/. Pour le type import l'extension *fileimport* doit être activée.
- `tooltip` : Pour les couches de type vecteur uniquement. Booléen précisant si les entités de la couche sont affichées sous forme d'infobulle au survol de la souris. (Les infobulles ne fonctionnent qu'avec une seule couche à la fois). Valeur par défaut = false.
- `tooltipenabled` : Précise la couche prioritaire pour l'affichage des infobulles.
- `tooltipcontent` : Chaîne de caractères décrivant l'information à afficher dans les infobulles. Cette chaîne contient soit le nom d'un champ de la couche soit un template Mustache (code html) combinant plusieurs noms de champs. Exemple : `tooltipcontent="{{name}} - ({{city}})"`.

Note : Il est possible d'utiliser du code **HTML** pour mettre en forme la tooltip. Exemple : `{{name}} </br> {{city}}`. En HTML, `</br>` permet d'effectuer un saut de ligne, ce qui nous permet ici d'avoir une tooltip sur 2 lignes. **Attention**, cette expression doit être convertie en une expression compatible XML, c'est à dire avec le code HTML échappé. Il existe des outils en ligne pour cela. L'expression valide pour l'expression précédente est : `tooltipcontent="{name}} </br> {{city}}"`

- `vectorlegend` : Booléen précisant si la légende pour les couches de type vecteur (customlayer ou import) est dynamiquement créée.
- `nohighlight` : Booléen précisant, pour les couches de type vecteur (customlayer, geojson ou import), si la mise en surbrillance du hover est désactivée.

Paramètres pour gérer la dimension temporelle des couches WMS

- `timefilter` : Booléen précisant si la dimension temporelle est activée pour cette couche. Voir (http://docs.geoserver.org/latest/en/user/services/wms/time.html)
- `timeinterval` : Intervalle de temps day|month|year
- `timecontrol` : Type d'affichage de l'intervalle de temps calendar|slider|slider-range
- `timevalues` : Valeurs temporelles séparées par des virgules. À utiliser avec le controle slider pour des valeurs non régulières ex (1950, 1976, 1980, 2004).
- `timemin` : Date mini format : « yyyy-mm-dd ».
- `timemax` : Date maxi format : « yyyy-mm-dd ».

Paramètres pour gérer le filtre attributaire (liste déroulante) des couches WMS

- `attributefilter` *studio* : Booléen précisant si on active la sélection attributaire par menu déroulant.
- `attributefield` *studio* : Nom du champ à utiliser avec le contrôle attributefilter.
- `attributevalues` *studio* : Valeurs de la sélection attributaire séparées par des virgules.
- `attributelabel` : Texte à afficher pour chaque atttribut de la liste déroulante associée.
- `attributestylesync` : Booléen qui précise s'il convient d'appliquer un style (sld) spécifique lors du filtre attributaire. Dans ce cas la convention est la suivante : nom_style@attributevalue ou url_style_externe@attributevalue.sld.
- `attributefilterenabled` : Booléen précisant si le filtre est activé par défaut (avec la première valeur de la liste attributevalues).
- `attributeoperator` : guilabel :*studio* : Opérateur utilisé pour construire le filtre. (= ou like). Defaut = « = ». Attention dans le cas de like, le wildcard est harcodé : %
- `wildcardpattern` : Pattern à utiliser pour les filtre utilisant l'opérateur like. Defaut = « %value%, autres possibilités « %value » et « value% ».

Autres paramètres

- `customlayer` : Texte précisant le nom du fichier JavaScript permettant la création d'une couche ou bien l'url complet du fichier JavaScript.
  - `URL renseignée` : le fichier JavaScript (.js) correspondant à l'URL est chargé
  - `Nom du fichier renseigné` : l'URL est fabriquée automatiquement à partir de l'ID de la couche. Le fichier devra être dans le répertoire customLayers/layerid.js (ou layerid correspond à l'id de la couche)
- `customcontrol` : Booléen précisant si la couche dispose d'un addon html à intégrer. La valeur par défaut est false.
  - `Valeur renseignée` : le fichier JavaScript (.js) correspondant à l'url est chargé
  - `Valeur non renseignée` : l'url est fabriquée à partir de l'ID de la couche (ex : custom :ayers/layerid.js)
- `customcontrolpath` : Texte Précisant le répertoire hébergeant les fichiers nécessaires au contrôle. Dans ce pépertoire, il faut déposer un fichier js et un fichier html ayant pour nom l'id de la couche. La structure du js doit être la suivante : (../controls/epci.js). Valeur par défaut = customcontrols.
- **`secure`** ***studio***

  [Texte précisant le niveau de protection de la couche Les valeurs possibles sont :]
  - `public` : (ou paramètre absent), l'accès à la couche est public
  - `global` : l'accès à la couche est contrainte par le CAS geoserver. Un test est effectué pour savoir si la couche est accessible. Si ce n'est pas le cas, la couche est retirée du panneau et de la carte.
  - `layer` : l'accès à la couche nécessite une authentification sur le service (WMS). Un bouton « cadenas » est ajouté dans la légende pour cette couche. Au clic sur ce bouton, un formulaire est affiché permettant de saisir des identifiants d'accès qui seront envoyés à chaque appel au service.
- `authorization` : Permet d'indiquer des identifiants par défaut si secure est à « layer »
- `useproxy` *studio* : Booléen précisant s'il faut passer par le proxy ajax (nécessaire pour fixer les erreurs de crossOrigin lorsque CORS n'est pas activé sur le serveur distant.
- `owsoptions` : Pour une couche WMS, permet de forcer certains paramètres des requêtes GetMap. Exemple : « VERSION :1.1.1,EXCEPTIONS :application/vnd.ogc.se_inimage ».
- `infopanel` : Permet d'indiquer quel panel d'interrogation utiliser parmis top-panel ou bottom-panel ou modal-panel. Exemple : *infopanel= »bottom-panel »*.

Zoom sur le paramétrage de gestion de l'ordre d'affichage des couches

```
<layer   index="1" showintoc="true" toplayer="true"/>
```

Par défaut, les couches sont affichées sur la carte par ordre d'appararition dans le fichier de configuration XML. L'utilisateur a la possibilité d'utiliser les paramètres suivants pour forcer l'affichage au démarrage de l'application :

— `toplayer` *studio* : Ce paramètre va forcer l'affichage de la couche au dessus des autres couches.

Si plusieurs toplayers sont renseignés dans le fichier de configuration, toutes les toplayers seront au dessus et selon l'ordre d'apparition dans la configuration XML. Si une couche a un toplayer et un index de renseigné, l'index est ignoré.

— `index` *studio* : L'objectif de ce paramètre est donc d'afficher la légende de façon identique à l'affichage sur la carte à l'initialisation de la carte.

Ce paramètre va permettre de forcer l'affichage de la couche à une position pour un index souhaité. Ce paramètre *index* correspond sur la carte au paramètre [zIndex](https://openlayers.org/en/latest/apidoc/module-ol_layer_Layer-Layer.html) d'une couche OpenLayers. Une couche avec le paramètre *index= »2 »* va donc afficher cette couche en seconde position (zIndex 2) et en seconde position dans la légende (sauf cas spécifique).

Par défaut, les couches avec un index seront toujours au-dessus des couches sans index. Si deux couches ont le même index dans un même fichier de configuration XML, parmis ces deux couches, la couche en seconde position dans l'ordre d'apparition du fichier de configuration XML sera considérée sans index (voir explications suivantes).

```
<layer   index="1" />
<layer   index="2" />
```

— `showintoc`

Avec ce paramètre renseigné, les paramètres index et toplayer sont également pris en compte pour l'affichage sur la carte.

```
<layer   index="1" />
<layer   index="2" toplayer="true" showintoc="true"/>
<layer   index="3" />
```

— couches sans index, sans toplayer, sans showintoc

```
<layer   index="1" />
<layer   index="2" />
<layer />
<layer />
```

Pour le cas primaire où aucun paramètre n'est renseigné, c'est l'ordre d'apparition dans le fichier de configuration XML qui permet de définir l'ordre d'affichage des couches au démarrage. Dans le cas où une configuration XML comprend des couches avec le paramètre *index* et / ou *toplayer* et des couches sans aucun de ces paramètres, alors les couches sans paramètre respectent ce principe.

On retrouvera donc en premier les toplayer, ensuite les couches avec index et enfin les couches sans index. Pour rappel, les couches avec un index en doublon et placée en seconde position dans le XML sont considérée sans index et sont concernées par ce mécanisme d'affichage. Elles s'afficheront donc selon les autres couches sans paramètres dans l'ordre d'apparition dans XML.

3.8.2 Syntaxe <template>

Elément enfant de <layer>

Cet élément optionnel, permet d'associer un template type Mustache (https://github.com/janl/mustache.js) à la fiche d'information de la couche.

Pour fonctionner, il faut que le paramètre `infoformat` ait la valeur « application/vnd.ogc.gml ». Le template peut être un fichier statique ex templates/template1.mst ou directement saisi dans le noeud <template> avec les balises < ![CDATA[ ]]>.

```
<template   url="" />
```

Paramètres

— `url` : paramètre optionnel de type url qui indique l'emplacement du template à utiliser.

3.9 Configurer - La recherche

3.9.1 Recherche d'adresse via olscompletion

Liens vers service d'autocomplétion et de géocodage d'adresses.

Syntaxe

```
<olscompletion url="" type="" attribution="" />
```

Attributs

— `url` : URL du service d'autocomplétion d'adresse : BAN https://api-adresse.data.gouv.fr/search/ ou IGN https://data.geopf.fr/geocodage/completion

— `type` : Optional - Type de service utilisé ign ou ban - defaut = ign

— `attribution` : Attribution du service de geocodage.

Exemple

```
<olscompletion url="https://api-adresse.data.gouv.fr/search/"
    type="ban"
    attribution="La recherche d'adresse est un service proposé par l'API adresse.data.
↪gouv.fr"/>
```

3.9.2 Recherche d'entités

2 possibilités sur mviewer :

— « *Configurer - Une recherche basée sur un index Elasticsearch* »

— « *Configurer - Une recherche Fuse* »

3.9.3 Paramétrage des recherches

Options liées à la recherche d'adresse *(olscompletion)* et/ou à la recherche d'entités *(elasticsearch ou fuse)*.

Syntaxe

```
<searchparameters
      banmarker=""
      imgurl=""
      imgwidth=""
      svgcolor=""
      bbox=""
      inputlabel=""
      localities=""
      features=""
      static=""
      querymaponclick=""
      closeafterclick=""
      animate=""
      duration=""/>
```

Attributs

- `localities` *(optionnel)* : Utilisation du service d'adresse olscompletion : true ou false (defaut = true).
- `features` *(optionnel)* : Utilisation du service de recherche d'entités elasticsearch ou fuse : true ou false (defaut = true).
- `bbox` *(optionnel)* : Recherche d'adresse et/ou d'entitées limitée à l'emprise de la carte : true ou false (defaut = false).
- `querymaponclick` *(optionnel)* : Interroge la carte après sélection d'un résultat dans la liste : true ou false - defaut = false.
- `banmarker` *(optionnel)* : Afficher ou non un icône sur le résultat de la recherche d'adresse : true ou false (defaut = true).
- `static` *(optionnel)* : En lien avec le paramètre doctypes. Active ou désactive la recherche associée à des documents requêtés systématiquement, indépendamment des couches affichées : true ou false (defaut = false).
- `inputlabel` *(optionnel)* : Texte à utiliser pour le placeholder de la zone de saisie de la barre de recherche. default = « Rechercher ».
- `closeafterclick` *(optionnel)* : Ferme la liste des résultats de recherche après avoir sélectionné un item : true ou false - defaut = false.
- `animate` *(optionnel)* : Active ou désactive l'animation de la vue lorsqu'un résultat de recherche est sélectionné : true ou false (defaut = false).
- `duration` *(optionnel)* : Durée en ms de l'animation définie dans l'option "animate" (defaut = 1000).
- `imgurl` *(optionnel)* : Url de l'image PNG / JPEG à afficher à l'emplacement du résultat sélectionné en guise de pointeur.
- `imgwidth` *(optionnel)* : Taille de l'image (voir paramètre imgurl) du pointeur représentant le résultat sélectionné.
- `svgcolor` *(optionnel)* : Couleur du pointeur représentant la localisation du résultat sélectionné.

FIG. 1 – Activation de l'option **animate**.

3.10 Configurer - Une recherche basée sur un index Elasticsearch

Permettre d'interroger un index Elasticsearch à partir d'une saisie libre (exemple « Port de Brest »). Le résultat retourné est une collection de documents disposant d'un champ commun avec les entités géographiques servies par l'instance WMS/WFS. Par convention les types **elasticsearch** ont le même nom que les couches WMS/WFS.

3.10.1 Prérequis

Installer une instance Elasticsearch https://www.elastic.co/fr/downloads/elasticsearch.

3.10.2 Création d'un index

créer un index nommé **mviewer** avec un champ de type **geo_shape nommé** geometry et un champ de type **keyword** nommé id.

Pour celà, depuis le serveur hôte hébergeant l'instance Elasticsearch, lancer la commande **CURL** suivante :

```
$ curl -XPUT 'localhost:9200/mviewer?pretty' -d '{
    "template": "mviewer",
    "settings": {
    "number_of_shards": 1
    },
      "mappings": {
        "_default_": {
          "properties": {
            "geometry": {
                "type": "geo_shape",
                "tree": "quadtree",
                "precision": "10m"
            },
            "id": {
                "type":  "keyword"
            }
          }
        }
      }
    }'
```

3.10.3 Alimenter l'index avec des données

Il existe plusieurs mode d'alimentation. Un des plus connus consiste en l'utilisation de logstash https://www.elastic.co/downloads/logstash. et du plugin jdbc https://www.elastic.co/guide/en/logstash/current/plugins-inputs-jdbc.html. permettant d'indexer des données provenant de bases de données.

Une autre méthode consiste à utiliser l'API d'indexation d'Elasticsearch https://www.elastic.co/guide/en/elasticsearch/reference/5.6/docs-bulk.html. via une commande CURL et un fichier contenant les instructions et données d'indexation et les données. Le fichier démo lycee permet de rapidement alimenter un index avec les données des lycées bretons.

Télécharger le fichier lycee http://kartenn.region-bretagne.fr/doc/lycee.bulk.json. et exécuter la commande suivante.

```
$ curl -s -H "Content-Type: application/x-ndjson" -XPOST 'http://localhost:9200/_bulk' --
↪data-binary "@lycee.bulk.json"
```

3.10.4 Tester l'index

Si tout s'est bien déroulé, la commande suivante doit renvoyer deux lycées :

```
$ curl -XGET 'localhost:9200/mviewer?q=zola&pretty'
```

3.10.5 Connecter mviewer à cet index Elasticsearch

Il est conseillé de n'exposer que l'API de recherche (_search) sur le web. Imaginons qu'une configuration serveur expose sur le web "localhost :9200/mviewer/_search" en "http ://monserveur/els/_search"

En partant de la démo Elasticsearch : http://kartenn.region-bretagne.fr/kartoviz/demo/els.xml, modifier le fichier de configuration pour que l'application pointe sur l'index Elasticsearch précédemment créé.

Syntaxe

```
1    <elasticsearch url="http://monserveur/els/_search" geometryfield="geometry" linkid=
↪"search_id" querymode="match"/>
```

Attributs

- `url` : URL de l'API Search
- `geometryfield` : nom du champ utilisé par l'instance elasticsearch pour stocker la géométrie
- `linkid` : nom du champ à utiliser côté serveur wms/wfs pour faire le lien avec la propriété _id des documents elasticsearch
- `querymode` *(optionnel)* : query mode used by elasticsearch to find results : match ou term ou phrase - default = match. Le mode match convient pour la recherche libre et naturelle. Le mode phrase permet de faire des recherches sur une phrase et le mode terme permet de faire une recherche sur un terme exact. Il est à noter que l'utilisateur peut activer le mode terme en préfixant sa recherche de # et activer le mode phrase en encadrant sa recherche de « « .
- `doctypes` *(optionnel)* : types des documents elasticsearch à requêter systématiquement, indépendamment des couches affichées
- `version` *(optionnel)* : version de l'instance elasticsearch (exemple = 5.3)

Tester en lançant http://monserveur/mviewer/?config=demo/els.xml et saisir zola dans la barre de recherche.

3.11 Configurer - Une recherche Fuse

3.11.1 Présentation de Fuse

Fuse permet de rechercher une entité dans la barre de recherche sans installer de solution lourde. Il est adapté à un nombre limité d'entités.

3.11.2 Création du fichier javascript

Il est nécessaire de créer un fichier JavaScript pour utiliser la recherche Fuse. La donnée apparaît sur la carte au format vectoriel.

Dans ce fichier :

- son nom et le nom de la variable déclarée au début devront être identiques au nom de la couche
- la donnée issue de la requête WFS devra être dans la même projection que le mviewer (ici 4326)
- le style de la couche s'appuie sur OpenLayers (https://openlayers.org/workshop/fr/vector/style.html)

3.11.3 Exemples de fichiers javascript

Voici un premier exemple de fichier JavaScript pour une donnée ponctuelle (fichier auto_ecole.js)

```
{
mviewer.customLayers.auto_ecole = {};
var auto_ecole = mviewer.customLayers.auto_ecole;

// Génération de la liste des légendes
auto_ecole.legend = {items: [{
        geometry: "Point",
        label: "Auto Ecole",
        styles: [new ol.style.Style({
            image: new ol.style.Circle({
                fill: new ol.style.Fill({
                    color: "#ff2a00"
                }),
                stroke: new ol.style.Stroke({
                    color: "#ff2a00",
                    width: 4
                }),
                radius: 4
            })
        })]
    }]
};

// Appel de la source de donnée (attention à la projection) et affichage du style sur la␣
↪carte
mviewer.customLayers.auto_ecole.layer = new ol.layer.Vector({
        source: new ol.source.Vector({
            url: "https://geobretagne.fr/geoserver/dreal_b/wfs?SERVICE=WFS&VERSION=1.0.0&
↪REQUEST=GETFEATURE&TYPENAME=auto_ecole&outputFormat=application/json&srsName=EPSG:4326
↪",
            format: new ol.format.GeoJSON()
        }),
        style: function(feature, resolution) {
            return auto_ecole.legend.items[0].styles;
        }
});
mviewer.customLayers.auto_ecole.handle = false;
}
```

Un deuxième sur une donnée linéaire avec analyse sur enjeu biologique (bief.js)

```
{
mviewer.customLayers.bief = {};
var bief = mviewer.customLayers.bief;

bief.legend = { items: [
    {
        label: "Enjeu bio. modéré",
        geometry: "LineString",
        styles: [new ol.style.Style({
```

(suite sur la page suivante)

(suite de la page précédente)

```
            stroke: new ol.style.Stroke({ color: 'rgba(129, 236, 236,1.0)', width: 4 })
        })]
    },
    {
        label: "Enjeu bio. élevé",
        geometry: "LineString",
        styles: [new ol.style.Style({
            stroke: new ol.style.Stroke({ color: 'rgba(0, 206, 201,1.0)', width: 4 })
        })]
    },
    {
        label: "Enjeu bio. très élevé",
        geometry: "LineString",
        styles: [new ol.style.Style({
            stroke: new ol.style.Stroke({ color: 'rgba(250, 177, 160,1.0)', width: 4 })
        })]
    },
    {
        label: "Enjeu bio. majeur",
        geometry: "LineString",
        styles: [new ol.style.Style({
            stroke: new ol.style.Stroke({ color: 'rgba(225, 112, 85,1.0)', width: 4 })
        })]
    },
    {
        label: "Enjeu bio. inconnu",
        geometry: "LineString",
        styles: [new ol.style.Style({
            stroke: new ol.style.Stroke({ color: 'rgba(255, 234, 167,1.0)', width: 4 })
        })]
    }
] };

mviewer.customLayers.bief.layer = new ol.layer.Vector({
        source: new ol.source.Vector({
            url: "https://ows.region-bretagne.fr/geoserver/rb/wfs?SERVICE=WFS&VERSION=1.
↪0.0&REQUEST=GETFEATURE&TYPENAME=bief&outputFormat=application/json&srsName=EPSG:4326",
            format: new ol.format.GeoJSON()
        }),
        style: function(feature, resolution) {
            var stl;
            if (feature.get('enjeu_bio')) {
                switch (feature.get('enjeu_bio')) {
                    case "modéré":
                        stl = bief.legend.items[0].styles;
                        break;
                    case "élevé":
                        stl = bief.legend.items[1].styles;
                        break;
                    case "très élevé":
                        stl = bief.legend.items[2].styles;
                        break;
```

(suite sur la page suivante)

(suite de la page précédente)

```
                    case "majeur":
                        stl = bief.legend.items[3].styles;
                        break;
                    default:
                        stl = bief.legend.items[1].styles;
                }
            }
            return stl;
        }
});
mviewer.customLayers.bief.handle = false;
}
```

Un troisième sur un polygone avec analyse sur l'identifiant de parc (pnr.js)

```
{
mviewer.customLayers.pnr = {};
var pnr = mviewer.customLayers.pnr;

pnr.legend = { items: [
    {
        label: "PNR d'Armorique",
        geometry: "Polygon",
        styles: [new ol.style.Style({
            stroke: new ol.style.Stroke({ color: 'rgba(248, 194, 145,1.0)', width: 3 }),
            fill: new ol.style.Fill({ color: 'rgba(248, 194, 145,.7)'})
        })]
    },
    {
        label: "PNR du golfe du Morbihan",
        geometry: "Polygon",
        styles: [new ol.style.Style({
            stroke: new ol.style.Stroke({ color: 'rgba(246, 185, 59,1.0)', width: 3 }),
            fill: new ol.style.Fill({ color: 'rgba(246, 185, 59,0.7)'})
        })]
    },
    {
        label: "Projet",
        geometry: "Polygon",
        styles: [new ol.style.Style({
            stroke: new ol.style.Stroke({ color: 'rgba(229, 80, 57,1.0)', width: 3 }),
            fill: new ol.style.Fill({ color: 'rgba(229, 80, 57,0.7)'})
        })]
    }
] };

mviewer.customLayers.pnr.layer = new ol.layer.Vector({
        source: new ol.source.Vector({
            url: "https://ows.region-bretagne.fr/geoserver/rb/wfs?SERVICE=WFS&VERSION=1.
↪0.0&REQUEST=GETFEATURE&TYPENAME=parc_naturel_regional&outputFormat=application/json&
↪srsName=EPSG:4326",
            format: new ol.format.GeoJSON()
```

(suite sur la page suivante)

(suite de la page précédente)

```
        }),
        style: function(feature, resolution) {
            var stl;
            if (feature.get('pnr_ident')) {
                switch (feature.get('pnr_ident')) {
                    case "1":
                        stl = pnr.legend.items[0].styles;
                        break;
                    case "2":
                        stl = pnr.legend.items[1].styles;
                        break;
                    default:
                        stl = pnr.legend.items[2].styles;
                }
            }
            return stl;
        }
});
mviewer.customLayers.pnr.handle = false;
}
```

Un dernier exemple sur un polygone avec analyse un champ numérique

```
{
// Définition des variables.
mviewer.customLayers.indice_position_sociale_ecole = {};
var data = mviewer.customLayers.indice_position_sociale_ecole;

data.legend = { items: [
    {
        label: "Moins de 80",
        geometry: "Point",
        styles: [new ol.style.Style({
                        image: new ol.style.Circle({
        fill: new ol.style.Fill({
            color: '#B1252E'
        }),
        stroke: new ol.style.Stroke({
            color: "#ffffff",
            width: 3
        }),
        radius: 7
                })
        })]
    },
    {
        label: "Entre 80 et 100",
        geometry: "Point",
        styles: [new ol.style.Style({
                        image: new ol.style.Circle({
        fill: new ol.style.Fill({
            color: '#C28B7E'
```

(suite sur la page suivante)

(suite de la page précédente)

```
        }),
        stroke: new ol.style.Stroke({
            color: "#ffffff",
            width: 3
        }),
        radius: 7
                })
        })]
    },
    {
        label: "Entre 100 et 120",
        geometry: "Point",
        styles: [new ol.style.Style({
                        image: new ol.style.Circle({
        fill: new ol.style.Fill({
            color: '#A6B4DA'
        }),
        stroke: new ol.style.Stroke({
            color: "#ffffff",
            width: 3
        }),
        radius: 7
                })
        })]
    },
    {
        label: "Plus de 120",
        geometry: "Point",
        styles: [new ol.style.Style({
                        image: new ol.style.Circle({
        fill: new ol.style.Fill({
            color: '#4C75B6'
        }),
        stroke: new ol.style.Stroke({
            color: "#ffffff",
            width: 3
        }),
        radius: 7
                })
        })]
    }
]};


data.layer = new ol.layer.Vector({
        source: new ol.source.Vector({
            url: "https://ows.region-bretagne.fr/geoserver/rb/wfs?SERVICE=WFS&VERSION=1.
↪0.0&REQUEST=GETFEATURE&TYPENAME=indice_position_sociale_ecole&outputFormat=application/
↪json&srsName=EPSG:4326",
            format: new ol.format.GeoJSON()
        }),
        style: function(feature, resolution) {
```

(suite sur la page suivante)

(suite de la page précédente)

```
            var stl;
            if (feature.get('ips') < 80){
                stl = data.legend.items[0].styles;
                        }
            else if (feature.get('ips') >= 80 && feature.get('ips') < 100 ){
                stl = data.legend.items[1].styles;
                        }
            else if (feature.get('ips') >= 100 && feature.get('ips') < 120 ){
                stl = data.legend.items[2].styles;
            }
            else if (feature.get('ips') >= 120 ){
                stl = data.legend.items[3].styles;
            }
            return stl;
        }
});
data.handle = false;
}
```

3.11.4 Configuration dans le XML

Au niveau du fichier de configuration mviewer, il est nécessaire de faire les adaptations suivantes au niveau de la couche :

```
type="customlayer" vectorlegend="true" url="https://geobretagne.fr/pub/mviewer-formation/
↪exemples/customlayers/auto_ecole.js"
searchable="true" searchengine="fuse" fusesearchkeys="NOM" fusesearchresult="{{NOM}} - {
↪{TYPE}}" fusesearchthreshold="0.5"
```

- `type` : mettre customlayer
- `vectorlegend` : activer l'affichage de la légende saisie dans le fichier javascript
- `url` : url du fichier javascript
- `searchable` : activer la recherche
- `searchengine` : activer le mode de recherche fuse
- `fusesearchkeys` : champ dans lequel on va effectuer la recherche. Possible sur plusieurs champs (exemple : « NOM,TYPE »)
- `fusesearchresult` : expression d'affichage du résultat de la recherche
- `fusesearchthreshold` : optionnel, cette valeur permet de préciser si la recherche doit retourner des résultats très proches de la saisie (0) ou tout mot ou partie de mot qui correspond (1)

3.12 Configurer - Le proxy

CORS

Cross-Origin Resource Sharing
Il n'y a pas besoin d'utiliser de proxy pour les données servies par GeoBretagne par exemple car CORS est activé. En savoir plus.

Lien vers votre proxy permmettant l'interrogation CROSS DOMAIN des couches. Mviewer n'est pas fourni avec un proxy Ajax. L'application peut fonctionner avec le proxy de **GeorChestra**. Un proxy cgi peut être utilisé. Plus de détail ici

Syntaxe

```
<proxy url="" />
```

Attributs

— `url` : Url vers votre proxy

3.13 Configurer - Extensions

3.13.1 Configuration globale

Chargement de librairies JavaScript externes ou de composants personnalisés. Ce module d'extension permet de répondre à deux cas d'usage :

— J'ai besoin d'une librairie **JavaScript** (chart.js) pour faire mes templates de couche.

— J'ai besoin de créer un nouveau **composant** (mini carte de localisation) sans modifier le cœur de mviewer. Plus de précisions ici : « *Configurer - Custom Component* »

Syntaxe

```
<extensions>
<extension type="javascript" src=""/>
<extension type="component" id="" path=""/>
</extensions>
```

Paramètres pour les extensions de type JavaScript

— `src` : paramètre obligatoire qui correspond à l'URL vers le fichier.

Paramètres pour les extensions de type component

— `id` : paramètre obligatoire qui correspond au nom du dossier du composant.

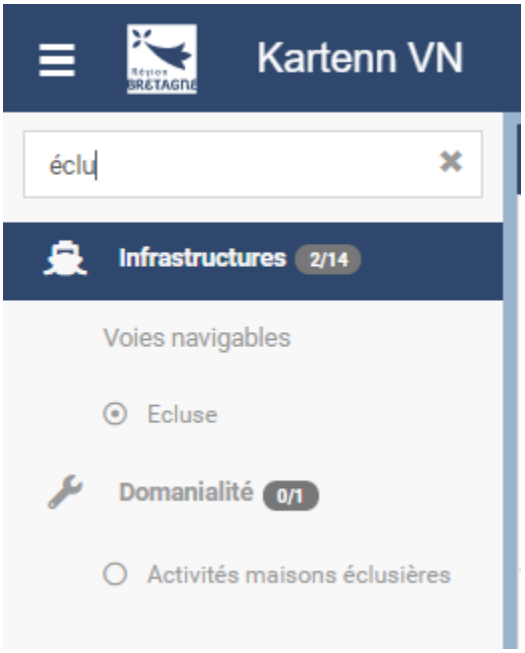
— `path` : paramètre obligatoire qui correspond à l'URL vers le dossier contenant la structure su composant.

Exemple

```
<extensions>
<extension type="javascript" src="chart.js"/>
<extension type="component" id="graph3d" path="demo/addons"/>
</extensions>
```

3.13.2 Extension filtre sur nom de la couche

Cett extension permet de filtrer ses couches selon leur titre comme ceci :



Il faut pour cela ajouter ceci dans votre XML :

```
<extensions>
    <extension type="component" id="layerfilter" path="demo/addons"/>
</extensions>
```

3.13.3 Extension plein écran

Cette extension permet d'afficher la carte en plein écran comme ceci :

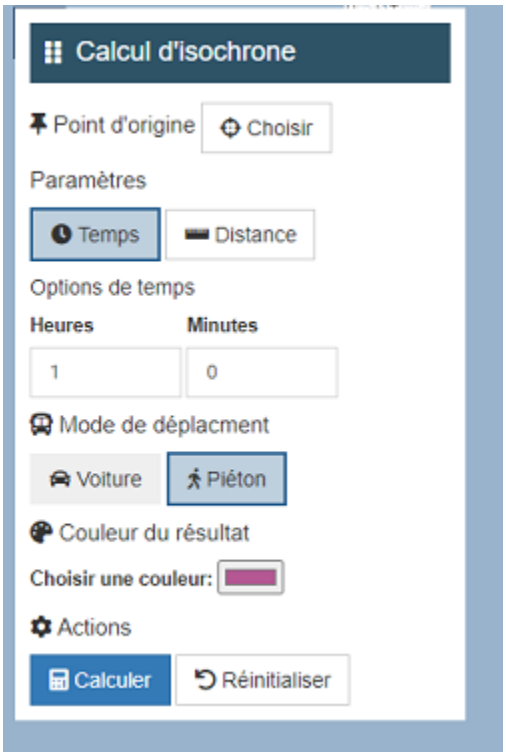
Cette extension est très utile si vous intégrez votre carte via une iframe.

Il faut pour cela ajouter ceci dans votre XML :

```
<extensions>
    <extension type="component" id="fullscreen" path="demo/addons"/>
</extensions>
```

3.13.4 Extension isochrone

Cett extension permet d'ajouter la possibilité de calculer des isochrones dans votre mviewer comme ceci :

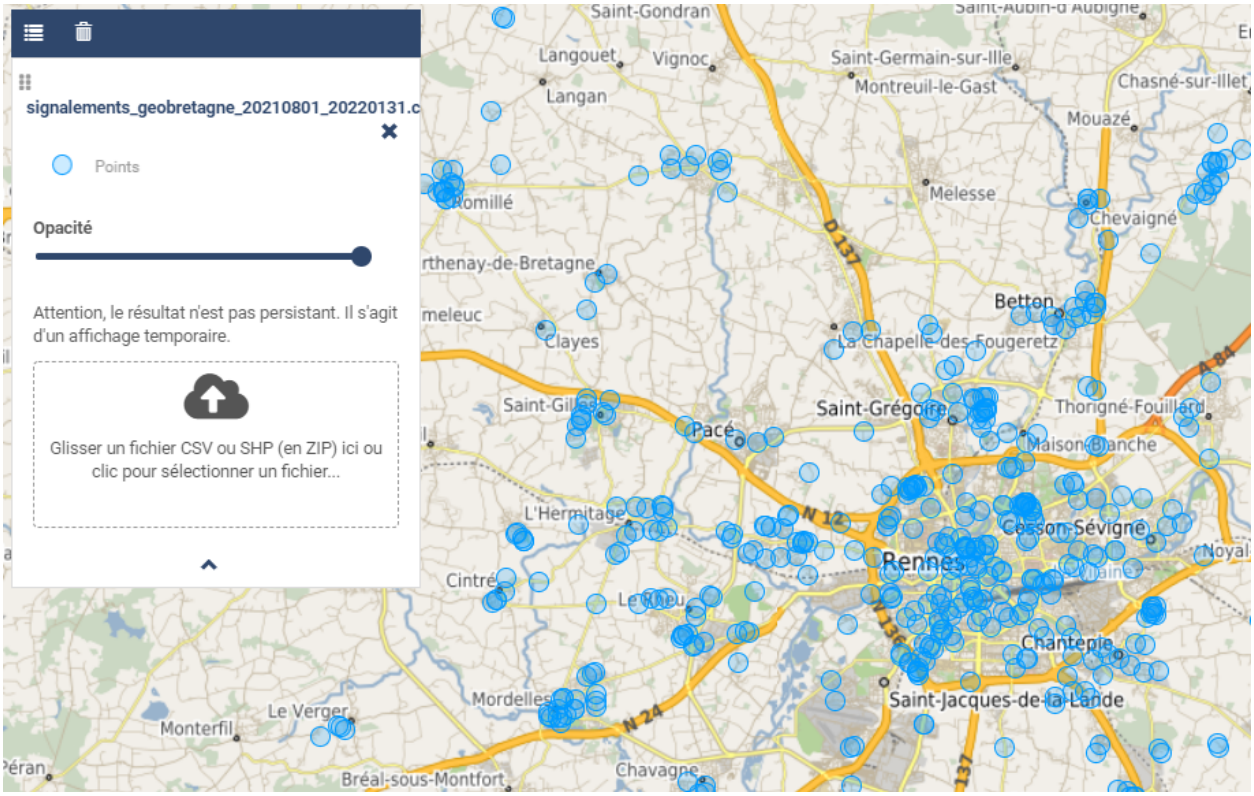


Cette extension utilise le geoservice de l'IGN. Il permet de faire des isochrones sur les parcours piétons et voiture.
Il faut pour cela ajouter ceci dans votre XML :

```
<extensions>
    <extension type="component" id="isochroneAddon" path="demo/addons"/>
</extensions>
```

3.13.5 Extension ajout couche temporaire

Cette extension permet d'ajouter une couche dans votre mviewer. Attention, la couche ne sera pas persistente :



Elle fonctionne avec les formats CSV et Shapefile (via un ZIP).
Il faut pour cela ajouter l'appel à l'extension dans votre XML :

```
<extensions>
    <extension type="component" id="fileimport" path="demo/addons"/>
</extensions>
```

Puis une couche au niveau de l'import donc voici un exemple (plus d'info sur la conf https://github.com/mviewer/mviewer/tree/master/demo/addons/fileimport) :

```
<theme name="Données externes" collapsed="true" id="import" icon="caret-right">
    <layer type="import" id="import_file" name="Import de donnée locales"  visible="false
↪"
        legendurl="img/blank.gif"
        queryable="true"
        vectorlegend="true"
        geocoder="ban"
        xfield="longitude"
```

(suite sur la page suivante)

(suite de la page précédente)

```
        yfield="latitude"
        attribution="Attention, le résultat n'est pas persistant. Il s'agit d'un␣
↪affichage temporaire."
        expanded="true">
        <projections>
            <projection proj4js="'EPSG:3857','+proj=merc +a=6378137 +b=6378137 +lat_ts=0.
↪0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +wktext  +no_defs'"/>
            <projection proj4js="'EPSG:2154','+proj=lcc +lat_1=49 +lat_2=44 +lat_0=46.5␣
↪+lon_0=3 +x_0=700000 +y_0=6600000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
↪'"/>
        </projections>
    </layer>
</theme>
```

3.13.6 Extension filtre sur données

Cette extension permet de filtrer les entités d'une donnée :

Elle nécessite plusieurs prérequis :

— Elle s'applique sur les couches de type customlayer (couche vecteur avec création d'un fichier JavaScript pointant sur un flux WFS par exemple). Cela ne fonctionne pas sur WMS. Pour plus d'information sur le fichier JavaScript à créer, se référer à cette page « *Configurer - Une recherche Fuse* ».

```
type="customlayer"
```

— il faut définir un id au niveau de l'application dans le XML :

```
<application
        id="livre_lecture"
```

— il faut compléter le fichier demo/addons/filter/config.json en y ajoutant vos options de recherche et en mettant en début de liste le même id que dans votre XML. Exemple ici avec au début la configuration du positionnement de la fenêtre et ensuite les recherches par couches

```
"livre_lecture":{
  "tooltipPosition": "bottom-left",
  "title": "Filtrer",
  "open": true,
  "zoomOnFeatures": true,
  "legendTitle": "Sélectionner une donnée :",
  "style": {
    "border": "1px #2e5367 solid",
    "background": "#2e5367",
    "text": "white",
    "colorButton": "#2e5367"
  },
  "layers": [{
    "layerId": "reseau",
    "filter": [{
        "attribut": "code_departement",
        "type": "button",
        "label": "Départements"
      },
      {
        "attribut": "diagnostic_terr",
        "type": "button",
        "label": "Diagnostic Territorial",
        "updateOnChange": true
      }
    ]
  }]
}
```

Il faut pour cela ajouter ceci dans votre XML :

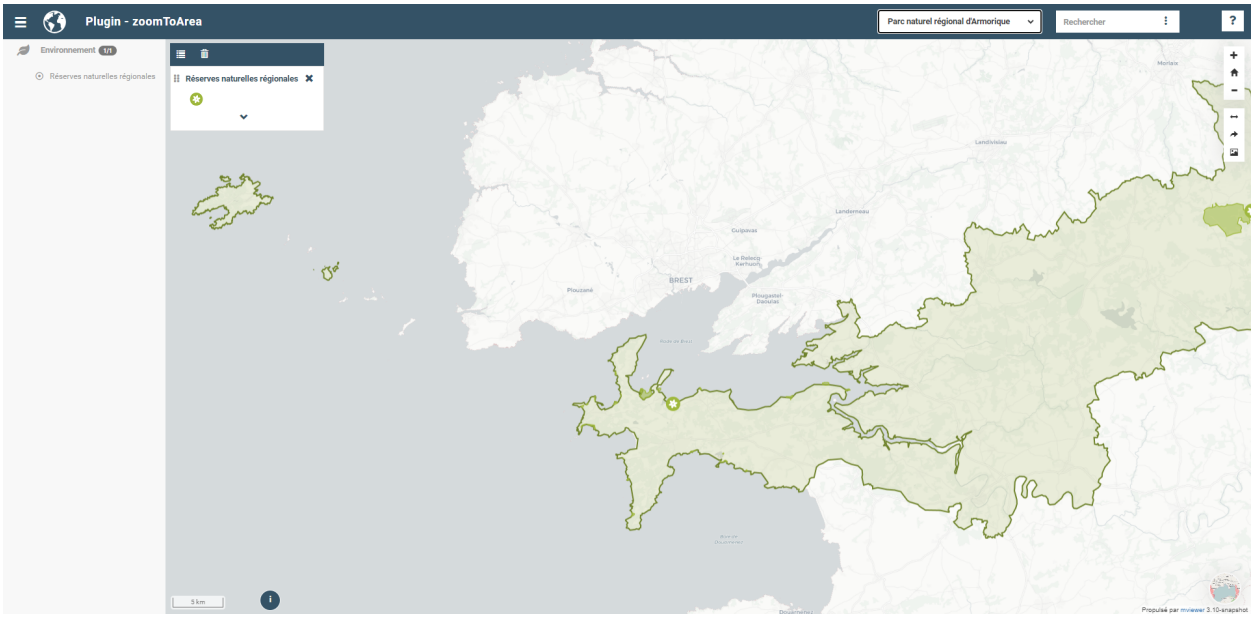
```
<extensions>
                <extension type="component" id="filter" path="demo/addons"/>
</extensions>
```

— Vous pouvez aussi rajouter des boutons pour permettre à l'utilisateur de télécharger les données filtrées en utilisant la propriété *downloadFormats*, ceci n'est possible que pour un layer WFS et créé un filtre CQL pour télécharger les données via une requête WFS.

```
"layers": [{
  "layerId": "occurence_public_filter",
  "downloadFormats":[
    { "label": "CSV", "format": "CSV" },
    { "label": "Shapefiles", "format": "SHAPE-ZIP" },
    { "label": "Geojson", "format": "application/json" }
  ],
  "filter": [{
      "attribut": "man_made",
      "type": "combobox",
      "label": "Man_made",
      "updateOnChange": true
    },...
```

3.13.7 Extension zoomToArea

Ce plugin permet à l'utilisateur de zoomer sur une zone d'intérêt sélectionnée préalablement dans une liste déroulante située en haut à droite du header mviewer.



Les entités disponibles dans la liste peuvent provenir :

— soit d'un flux wfs issue d'un catalogue de données

— soit d'un fichier au format .geojson disponible dans le dossier de l'application `apps/monapp/data`

1. Utiliser un ID par Mviewer

Nous vous conseillons d'utiliser un identifiant respectif à votre Mviewer :

```
<application id="zoomtoarea"></application>
```

À l'image de certains plugins existants (filter), cet ID permettra de lier une configuration (fichier `config.json`) à un seul mviewer. Ainsi, un seul dossier et un seul fichier `config.json` pourra être utilisé pour configurer le plugin. Ce qui évite de dupliquer le dossier principal du plugin par mviewer.

2. Importer le plugin

Comme tous les plugins, vous devez ajouter dans le fichier de configuration de votre mviewer une balise permettant de charger le plugin :

```
<extensions>
                <extension type="component" id="zoomToArea" path="demo/addons"/>
</extensions>
```

La configuration du plugin est accessible dans le fichier `config.json` du répertoire `addon/zoomToArea`. Ce répertoire peut être localisé différemment selon votre organisation.

3. Déclarer les paramètres du plugin pour votre carte

Pour commencer, vous devez ajouter votre ID de Mviewer sous la propriété `mviewer` indiquant ainsi que le plugin est paramétré pour la carte associée à l'ID :

```
{
"js": ["zoomToArea.js"],
"css": "style.css",
"html":"zoomToArea.html",
"target": "page-content-wrapper",
"options":
        {
                "mviewer":
                {
                        "idApp1":
                                {
                                ...
                                }
                },
                {
                        "idApp2":
                                {
                                ...
                                }
                }
        }
```

}

4. Configurer les paramètres du plugin

Pour fonctionner, le plugin a besoin des paramètres suivants :

```
       "zoomtoarea":
       {
               "dataUrl": "apps/monapp/data/featuresZoom.geojson",
               "dataEPSG": "EPSG:4326",
               "fieldNameAreas": "name_feature",
               "fieldIdAreas": "id_feature",
               "fieldSortBy":"name_feature",
               "bufferSize": 5000,
               "selectLabel":"Sélectionner un territoire"
       }


- ``dataUrl`` : Lien vers la couche de données (flux wfs ou couche geojson)
```

(suite sur la page suivante)

(suite de la page précédente)

```
- ``dataEPSG`` : Projection des données sources
- ``fieldNameAreas`` : Nom du champs où se trouve le nom des entités
- ``fieldIdAreas`` : Nom du champs où se trouve l'id des entités
- ``fieldSortBy`` : Nom du champs pour ordonner les entités dans la liste déroulante␣
↪(ordre croissant)
- ``bufferSize``:  Valeur numérique définissant la taille du buffer réalisé autour des␣
↪entités (permet de régler le niveau de zoom),
- ``selectLabel`` : Label de la liste déroulante
```

Exemple

Vous pouvez retrouver un exemple complet dans les dossiers suivants : - Fichier de configuration du plugin : `demo/addons/zoomToArea/config.json` - Fichier de configuration de la carte : `demo/zoomtoarea.xml`

Visible également sur la page des démonstrations mviewer.

Astuces

Afficher les contours sur la carte

Ce plugin ne permet pas d'afficher la couche de données sur la carte.

Si vous souhaitez visualiser les contours des polygones, vous pouvez intégrer la couche de données en tant que `layer` comme une couche classique.

Si vous ne voulez pas afficher cette couche dans le menu thématique et la légende (aucune action possible pour l'utilisateur), vous pouvez activer le paramètre suivant à votre `layer` :

```
showintoc="false"
```

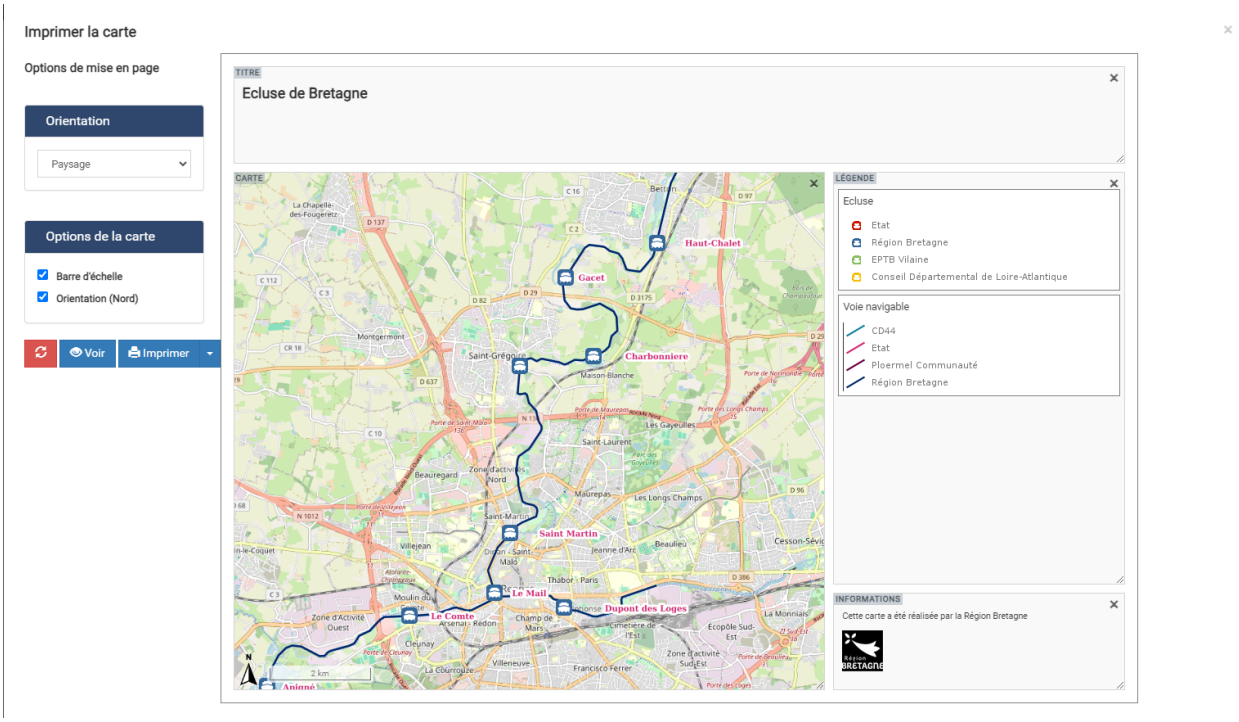
Filtrer les entités d'une couche

A l'heure actuelle, il n'est pas possible de filtrer les entités sur lesquelles zoomer depuis le plugin. Vous devez préalablement créer une couche avec vos entités filtrées et les importer dans votre application *apps/monapp/data* en privilégiant la projection *EPSG :4326* dans la mesure du possible.

Il est ensuite nécessaire de pointer le plugin vers cette couche comme présenté auparavant.

3.13.8 Extension print

Ce plugin permet d'ajouter un module d'impression dans mviewer pour générer des cartes aux formats PDF ou PNG. Seul le format A4 est disponible.



Installation

Par défaut, print extension est disponible dans le dossier `/demo/addons`. Il est conseillé de déplacer les addons que vous utiliser dans un dossier séparé.

1. Ajout de l'extension dans votre conf XML :

```
<extensions>
  <extension type="component" id="print" path="demo/addons"/>
</extensions>
```

2. Ajout d'un id mviewer dans le bloc application de votre conf XML :

```
<application id="mon_app" title="A super title" />
```

3. Ajout de votre application dans le fichier de conf `demo/addons/print/config.json` :

```
"options": {
  "mviewer": {
        "mon_app": {
        }
  }
}
```

Configuration

La configuration du modèle n'est pas requise, vous pouvez utiliser le modèle par défaut.

3 paramètres sont possibles dans le config.json :
- `printLayouts` : lien vers le modèle
- `ownerlogo` : logo à afficher dans le modèle
- `ownerInfos` : texte par défaut de la zone d'information

Voici un exemple :

```
"options": {
  "mviewer": {
        "mon_app": {
                "printLayouts": "demo/addons/print/layouts/standard.json",
                "ownerLogo": "https://avatars.githubusercontent.com/u/114171481?s=400&
↪u=7fcf63ac01887ece3f5f2d5527e92c10527c7a91&v=4",
                "ownerInfos": "Voici la carte"
        }
  }
}
```

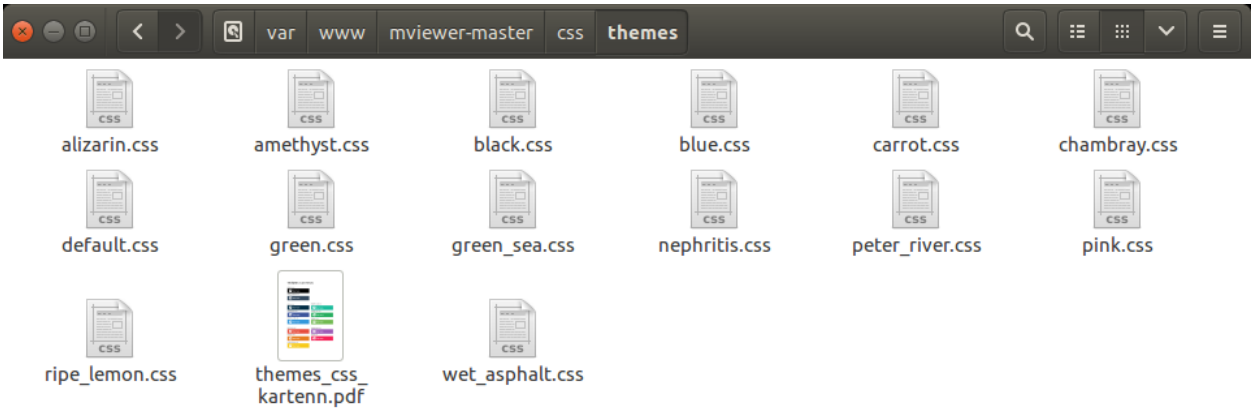
Personnalisation du modèle

Pour la personnalisation du modèle, se référer à la documentation ici https://github.com/mviewer/mviewer/tree/master/demo/addons/print#layout-with-a-json-template

3.14 Configurer - Apparence

Pour configurer l'apparence de mviewer, vous avez la possibilité de changer le style présent dans un fichier .css.

Par défaut, une liste de .css est fournie dans le dossier **/css/themes/**



3.14.1 Changer l'apparence

Pour changer l'apparence *(et donc le .css associé)* de mviewer, il vous suffit d'éditer le fichier **index.html** et de modifier l'adresse du fichier .css à la ligne suivante (n°49).

```
var style = "css/themes/default.css";
```

Par défaut, le fichier associé est **css/themes/default.css**.

```
$.ajax({
    url: configFile + '?_dc=' + (new Date()).getTime(),
    dataType: "xml",
    success: function (xml) {
        var style = "css/themes/default.css";
        if ($(xml).find("application").attr("style") && $(xml).find("application").attr("style").match("css")) {
            style = $(xml).find("application").attr("style");
        }
```

Remarque

À noter qu'un fichier themes_css_kartenn.pdf permet de visualiser à l'avance le rendu de chacun des thèmes proposés.

3.15 Configurer - Templates

Personnalisation de la fiche d'information

Pour les couches de type vecteur et WMS, il est possible de définir un template afin de formater côté client, la fiche d'information des entités sélectionnées. Le moteur de template (logic less) utilisé est Mustache : https://github.com/janl/mustache.js

3.15.1 Ce qu'il faut savoir de Mustache

— On fait référence à la valeur d'un champ de cette façon : `{{champ}}`.
— Il est possible de gérer une absence de valeur ou une valeur false de cette façon :

```
{{#champ2}}
    Ce texte s'affiche si champ2 contient une valeur ou est différent de false.
{{/champ2}}
```

La finalité du template est ici de fabriquer un contenu formaté HTML. L'ajout des balises <style> permet de personnaliser l'affichage du champ via du CSS. Exemple ci-dessous sur le formatage du texte et d'un bouton pour clic.

Pour aller plus loin sur la personnalisation, consulter les différentes documentation sur HTML et CSS.

Nous avons la possibilité d'injecter du code via la balise <script>.

3.15.2 Exemple de template structuré

```
{{#features}}
    <li id="{{feature_ol_uid}}" class="item">
        Exemple de formatage
        <h3 class="title-feature">{{nom}}</h3>
        <img src="{{image}}" class="img-responsive" style="margin-top:5%;" /><br/>
        <p class="text-feature">
            <b> Surface : </b> {{surface}} ha <br/>
        </p>
        <a href="{{url}}" target="_blank" title="Lien site internet" class="but-link">
            <span class="fa fa-globe" aria-hidden="true"></span> <b>Site Web</b>
        </a>
    </li>
{{/features}}

```

(suite sur la page suivante)

(suite de la page précédente)

```
<style>
    .title-feature {
        color: #BA88A4;
        font-family:"Trebuchet MS";
        font-size:20px;
        margin-bottom:3%;
        line-height:1;
    }
    .text-feature{
        font-family:"Trebuchet MS";
        color:#555;
        font-size:13px;
        margin-top:2%;
        margin-bottom:2%;
    }
    .but-link, .but-link:focus, .but-link:hover{
        display:inline-block;
        padding:0.5em 1em;
        margin:0 0.3em 0.3em 0;
        border-radius:0.3em;
        box-sizing: border-box;
        text-decoration:none;
        font-family:'Trebuchet MS';
        font-weight:300;
        color:#FFFFFF;
        background-color:#BA88A4;
        text-align:center;
        transition: all 0.2s;
    }
    .but-link:hover{
        background-color:#b0006b;
    }
</style>
```

Les éléments suivants en rouge sont obligatoires.

Explications du template mustache :

- `{{#features}}{{/features}}` est une boucle effectuée sur chaque entité présente dans la couche sélectionnée.
- {{surface}} affiche le contenu du champ surface.
- `<li id="{{feature_ol_uid}}" class="item"></li>` est une entrée de liste html utilisée par le mviewer. S'il y a plusieurs entrées de liste car plusieurs entités sélectionnées, le mviewer présentera les réponses sous la forme d'un carousel.
- Pour synchroniser le carousel et la sous-sélection sur la carte lors d'un clic, l'injection de la `feature_ol_uid` est requise dans l'" `id` de la balise.
- Puisque une `feature id` n'est pas obligatoire comme attribut pour une feature l'" `ol_uid` interne d'OpenLayers est utilisée à ce propos.
- <style> permet d'injecter du code CSS / HTML de personnalisation des styles utilisés dans le template.
- <script> permet d'injecter du code JavaScript.

Résultat de l'exemple ci-dessus

3.15.3 Itérer sur un champ de type json

Prérequis : disposer d'un champ - exemple `monchampjson` - dont le contenu est une liste de valeurs ou d'objets sous la forme `["item1","item2"]` ou de la forme `[{"nom": "item1", "code": 1}, {"nom": "item2", "code": 2}]`, configurer le layer dans le config.xml avec le paramètre `jsonfields="monchampjson"`.

Exemple 1 pour *monchampjson* = `["item1","item2"]`

```
{{#monchampjson}}
    Cette ligne s'affiche autant de fois qu'il y a d'éléments dans la liste.
    <li>{{.}}</li>
{{/monchampjson}}
```

Exemple 2 pour *monchampjson* = `[{"nom": "item1", "code": 1}, {"nom": "item2", "code": 2}]`

```
{{#monchampjson}}
    Cette ligne s'affiche autant de fois qu'il y a d'éléments dans la liste.
    <li>{{nom}} - {{code}}</li>
{{/monchampjson}}
```

Exemple 3 pour afficher un tableau comme dans ici https://kartenn.region-bretagne.fr/kartoviz/?config=demo/jsonfields.xml

```
<table>
    <thead>
        <tr><th>NOM</th><th>CODE</th></tr>
    </thead>
```

(suite sur la page suivante)

(suite de la page précédente)

```
    <tbody>
        {{#communes}}
            <tr><td>{{nom}}</td><td>{{code_insee}}</td></tr>
        {{/communes}}
    </tbody>
</table>
```

Résultat du template ci dessus

Liffré-Cormier Communauté

code_siren: 243500774

Liste des communes :

NOM	CODE
Chasné-sur-Illet	35067
Dourdain	35101
Ercé-près-Liffré	35107
Gosné	35121
La Bouëxière	35031
Liffré	35152
Livré-sur-Changeon	35154
Mézières-sur-Couesnon	35178
Saint-Aubin-du-Cormier	35253

Itérer sur les champs disponibles

En plus d'afficher la valeur d'un champ comme expliqué précédemment, il est aussi possible de lire et parcourir l'ensemble des champs disponibles avec `{{#fields_kv}}...{{/fields_kv}}`.

Pour chaque champ listé, on peut accéder :
- au nom du champ via `{{key}}`
- à la valeur via `{{value}}`

Par exemple, ce code :

```
{{#features}}
  <li id="{{feature_ol_uid}}" class="item" style="width:238px;">
      <ul>
        {{#fields_kv}}
          <li>{{key}} : {{value}}</li>
        {{/fields_kv}}
      </ul>
  </li>
{{/features}}
```

affichera la liste des champs avec leur nom suivi de leur valeur sans avoir à connaître les noms des champs à l'avance.

Dans le même ordre d'idée, un autre « champ virtuel », `{{serialized}}`, permet de récupérer l'ensemble des champs sous forme sérialisée, prête à être passée en paramètre dans une URL. Par exemple, dans une iframe :

```
<iframe class="iframe_bottom"src="apps/myapp/presentation/en/pages/mylayer_charts.html?
↪data={{serialized}}"></iframe>
```

Vous pourrez ensuite le désérialiser de façon standard. Par exemple, en javascript dans le fichier mylayer_charts.html (cf. exemple ci-dessus) :

```
<script>
  var url = new URL(location.href);
  var data = url.searchParams.get("data");

  if(data) {
    var obj = JSON.parse(data)
    keys = Object.keys(obj);
    for ( i=0 ; i<keys.length ; i++ ) {
      key=keys[i];
      console.log(key + ': '+obj[key]);
    }
    ...
  }
</script>
```

Les champs `{{#fields_kv}}` et `{{serialized}}` sont tous les deux virtuels : ils sont créés grâce à une fonctionnalité de Mustache permettant de définir des champs comme des fonctions. S'ils ne sont pas utilisés, ils ne consomment pas de ressource. Ils ont été ajoutés aux champs simples afin de faciliter certains flux de traitement des données.

3.15.4 Exemples de scripts de reformatage de champs

Reformatage d'un champ date

```
{{#features}}
<li id="{{feature_ol_uid}}" class="inventaire item" style="width:100%;">

        <!-- ici la div qui contiendra la date à afficher -->
        <!-- elle doit avoir un ID unique lié à la feature (champ id ou id openLayers␣
↪peu importe tant que c'est unique)-->
        <div id="date-area-{{feature_ol_uid}}"></div>
        <!-- le script de formatage -->
        <script>
                let date = "{{date_field}}";
                let id = "{{feature_ol_uid}}";
                let divWithDate = document.getElementById("date-area-" + id);
                // format de la date de la donnée d'entrée

                // voir ici pour les correspondances du code que vous devez modifier /␣
↪adapter selon le format d'entrée :
                // https://momentjs.com/

                let inDate = "YYYY-MM-DDT";
                let outDate = "DD/MM/YYYY";

                // on transforme
                let dateFormated = moment(date, inDate).format(outDate);
                // on met le nouveau format à afficher dans la div
                divWithDate.innerHTML = dateFormated;
        </script>
</li>
{{/features}}
```

Arrondi d'un champ nombre

```
<li id="{{feature_ol_uid}}" class="inventaire item" style="width:100%;">
        <div id="number"></div>
        <script>
                var maDiv = document.getElementById("number");

                // Récupérer la valeur textuelle de la div en utilisant innerText ou␣
↪textContent
                var valeurTextuelle = maDiv.innerText; // ou maDiv.textContent

                var pourcentageArrondi = parseInt(valeurTextuelle, 10); // 10 indique la␣
↪base décimale

                maDiv.innerText = pourcentageArrondi +"%"
        </script>
</li>
```

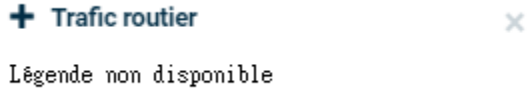
3.15.5 Appel depuis le XML

Le template sera enregistré avec l'extension .mst. Pour l'appeler dans la configuration mviewer au niveau de la layer, il faut le bon format `infoformat="application/vnd.ogc.gml"` et ajouter un appel au mst via une balise template au sein du layer `<template url=""/>`.

3.16 Débug erreurs courantes

Dans cette page nous mettrons des exemples de bug récurrents dans le mviewer et la façon de les résoudre.

— **Légende non disponible**



A venir…

A venir…

— **Je n'arrive pas à interroger ma couche**

Ce problème peut venir de plusieurs facteurs.

Bien vérifier le infoformat. Pour rappel, 2 possibilités :

— text/html : fichiers FTL GeoServer

— application/vnd.ogc.gml : fichiers mustache ou utilisations de fields et aliasfields

Vérifier sur votre serveur carto (GeoServer, MapServer…) que votre couche est interrogeable.

— **Je ne vois pas mon thème**

L'identifiant unique de mon thème n'est pas unique. Pour corriger, changer l'identifiant du thème invisible. Même principe pour un groupe de couche.

3.17 Configurer - Traduction

Cette page vous servira à comprendre et mettre en place les éléments pour traduire le mviewer.

Démonstration kartenn.

3.17.1 Fonctionnement technique

Librairies Le mviewer utilise la libraire i18n.js pour traduire l'interface vers le langage de votre choix.

Principe

La traduction s'appuie sur un dictionnaire de traduction (fichier mviewer i18n json). Tous les éléments html disposant d'un attribut i18n seront traduits si une correspondance est trouvée dans le dictionnaire :

```
<span i18n="une.cle.a.traduire"> une valeur par défaut </span>
```

Le fichier mviewer.i18n.json est structuré pour utiliser plusieurs langues de traduction :

```
{
    "fr": {
        "une.cle.a.traduire": "une valeur en français",
        "une.autre.cle": "une autre valeur en français"
```

(suite sur la page suivante)

(suite de la page précédente)

```
    },
    "en": {
        "une.cle.a.traduire": "a value",
        "une autre cle": "another value"
    }
}
```

Pour trouver la valeur à traduire, le système recherchera donc la clé contenue dans l'attribut « i18n » de notre élément de la page dans la langue sélectionnée :

```
<span i18n="search.adresse"> une valeur par défaut </span>

// donnera pour le français
"une.cle.a.traduire": "une valeur en français"
```

La valeur traduite sera ensuite appliquée sur l'élément de la page à la place de la valeur par défaut.

Le texte affiché pour notre élément de la page () sera alors :

```
"une valeur en français"
```

3.17.2 Quels sont les paramètres de configuration utilisés ?

Il vous faudra ajouter dans votre fichier de configuration les propriétés suivantes comme décrit dans la page mviewer :

— lang

Langue à utiliser pour l'interface. Si plusieurs langues sont spécifiées, un menu additionnel sera créé afin de permettre à l'utilisateur de choisir sa langue a première langue de la liste saisie sera alors utilisée par défaut. Par défaut, lang n'est pas activé.

— langfile

URL du fichier de traduction supplémentaire à utiliser en complément de mviewer.i18n.json. Ce fichier peut être un fichier distant (URL classique type https ://) ou bien un fichier localisé n'importe où dans votre serveur web.

Voici un exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<application title="Une belle carte" langfile="./custommviewer.i18n.js" lang="en,fr"></>
```

Vous pouvez donc changer de langue via l'URL directement. Essayez en premier avec cette url :

http://kartenn.region-bretagne.fr/kartoviz/?config=demo/lang.xml

Maintenant rajoutez

```
lang=fr
```

http://kartenn.region-bretagne.fr/kartoviz/?config=demo/lang.xml&lang=fr

3.17.3 Interface de changement de langue

Sélecteur de langue

Selon votre paramètre *lang* le mviewer vous permettra de sélectionner parmis les valeurs de traduction disponibles.

Avec « lang=fr » ou aucun paramètre « lang » vous n'ajouterez pas cette interface. Avec *lang=en,fr* vous pourrez modifier la langue dans l'interface du mviewer.

Affichage du sélecteur

Deux types d'affichage existent :

— Une liste de sélection :

```
<application lang="en,fr" langfile="demo/demo.i18n.json"/>


.. image:: ../_images/dev/config_translate/list_lang.png
          :alt: lang selector
          :align: center
```

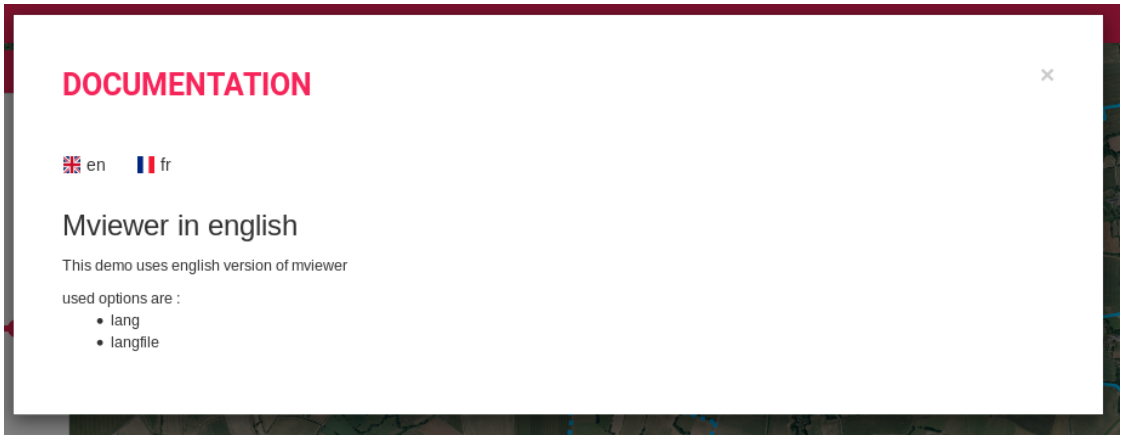
Dans cette configuration, la liste de sélection sera remplacée par une popup pour les petits écrans (mobiles et tablettes).

Cette fenêtre sera accessible via la barre de navigation en haut.

— Dans la fenêtre d'aide en ajoutant le paramètre *showhelp= »true »* dans la configuration de l'application :

```
<application lang="en,fr" langfile="demo/demo.i18n.json" help="demo/demo_lang_help.
↪html" showhelp="true"/>
```

Les langues seront visibles ainsi :



3.17.4 Comment insérer vos traductions ?

1 - Créez un fichier de traduction au format JSON ou compléter le fichier mviewer.i18n.json déjà disponible

Nous recommandons de créer un nouveau fichier. Le fichier mviewer.i18n.js sera ainsi surchargé (= complété) par votre fichier de traduction.

2 - Insérez les traductions dans le fichier en respectant ce formalisme :

```
{
    "fr": {
        "popup.help.title": "Bienvenu"
    },
    "en": {
        "popup.help.title": "Welcome"
    },
    "bzh": {
        "popup.help.title": "Degemer mat"
    }
}
```

3 - Pour du nouveau contenu HTML, rajoutez l'attribut *i18n="popup.help.title"* pour que le contenu soit traduit et rajoutez les nouvelles clés dans votre fichier de traduction :

```
<span i18n="search.adresse"> une valeur par défaut </span>
```

4 - Ajoutez les paramètres *lang="en,fr,bzh"* et *langfile="./chemin/fichier/traduction"*

5 - Eventuellement, choissisez d'afficher le sélecteur de langue dans la popup d'aide avec le paramètre *showhelp="true"*

6 - Testez

3.18 Configurer - Custom layer

Un custom layer est une couche personnalisée s'appuyant sur la librairie `OpenLayers`. Exemples

- J'ai besoin d'afficher une couche de type KML.
- J'ai besoin de créer une couche de type cluster avec une analyse personnalisée.
- J'ai besoin d'une couche de tuiles vectorielles OSM…

Syntaxe

```
<layer id="moncustomlayer"
       type="customlayer"
       url="chemin_vers_customlayer.js">
</layer>
```

Paramètres custom layer

- `type="customlayer"` : paramètre précisant qu'il s'agit d'une couche de type customlayer.
- `url` : paramètre qui indique où mviewer doit charger le fichier customlayer.js.

Exemple

Code source 3 – config.xml

```
    <layer id="heatmap"
    name="Earthquakes Heatmap"
    visible="true"
           queryable="true"
    url="demo/heatmap/customlayer.js"
    type="customlayer"
    legendurl="demo/heatmap/legend.png"
    opacity="1"
    expanded="true"
    attribution=""
    metadata=""
    metadata-csw="">
</layer>
```

Note : Apprendre par l'exemple :

- *Développer un customLayer*

3.19 Configurer - Custom control

Un custom control est une interface permettant à l'utilisateur d'interagir avec une couche. Le customcontrol est affiché dans le bloc de légende associée à la couche concernée. Un custom control est associé à une seule couche et est constitué de deux fichiers **html** + **javascript**. Ces deux fichiers doivent obligatoirement être nommé comme l'" `id` de la couche associée.

Exemples

- J'ai besoin d'une liste déroulante pour filtrer ma couche.
- J'ai besoin d'une zone de saisie pour filtrer ma couche

Syntaxe

```
<layer id="moncustom"
        customcontrol="true"
        customcontrolpath="chemin_vers_fichiers">
</layer>
```

Paramètres custom control

- `customcontrol` : paramètre qui indique si mviewer doit charger un customcontrol. valeurs : true || false. Défaut = false.
- `customcontrolpath` : paramètre qui indique où mviewer doit charger le customcontrol. Si ce paramètre n'est pas renseigné, le customcontrol est recherché dans le répertoire racine **customcontrols**.

Exemple

Code source 4 – config.xml

```
    <layer id="heatmap"
    name="Earthquakes Heatmap"
    visible="true"
    url="demo/heatmap/customlayer.js"
    queryable="true"
    type="customlayer"
    customcontrol="true"
    customcontrolpath="demo/heatmap/control"
    legendurl="demo/heatmap/legend.png"
    opacity="1"
    expanded="true"
    attribution=""
    metadata=""
    metadata-csw="">
</layer>
```

Note : Apprendre par l'exemple :

- *Développer un customControl*

3.20 Configurer - Custom Component

3.20.1 Pour quoi faire ?

On utilise les composants personnalisés pour modifier l'apparence de mviewer, ajouter des fonctionnalités, rajouter des librairies sans modifier le coeur de Mviewer. De cette façon, le Mviewer est ainsi plus facile à maintenir.

Il est ainsi possible de créer un composant très simple pour :

- afficher un logo dans une zone particulière
- ajouter un nouveau bouton dans la barre de navigation mviewer
- Créer un sélecteur temporel qui répercutera la sélection à l'ensemble des couches…

Les composants apportent donc plus de souplesse pour personnaliser vos Mviewer selon vos besoins.

3.20.2 Qui peut réaliser un composant ?

Tout le monde peut réaliser un composant à condition d'être un petit peu familier avec la configuration mviewer et d'avoir des notions en JavaScript. Une fois votre premier composant réaliser, il vous sera très facile d'en réaliser de nouveaux.

3.20.3 Comment faire ?

Un composant personnalisé - **customComponent** - est un dossier physique disposant d'un fichier obligatoire `config.json` qui identifie toutes les ressources à charger. Les ressources sont de 3 types :

- `html` –> html à charger
- `css` –> feuille de style à charger
- `js` –> liste de fichiers javascripts à charger

Exemple d'arborescence :

```
/demo
        ├── components
        │   ├── 3d
        │   │   ├── js
        │   │   │   ├── lib.js
        │   │   │   └── main.js
        │   │   │
        │   │   ├── config.json
        │   │   ├── component.html
        │   │   ├── style.css
        │   │   │
        │   ├── autre_customComponent
```

Exemple de fichier config.json :

```
{
  "js": [
      "js/lib.js",
      "js/main.js"
  ],
  "css": "style.css",
  "html": "component.html",
  "target": "map",
  "options": {
      "title": "3D",
      "position": 0,
  }
}
```

Le composant sera alors créé dans une **div** mviewer existante ciblée par le paramètre **target**. Par défaut le composant est ajouté comme dernier noeud enfant du **target** dans le DOM. Le paramètre optionnel **position** permet de changer cette position et ajouter le composant comme enfant **n** du **target**.

Voici un extrait de la configuration xml :

```
<config>
    <extensions>
        <extension type="component" id="3d" path="demo/components"/>
```

(suite sur la page suivante)

(suite de la page précédente)

```
    </extensions>
</config>
```

3.20.4 Exemples

Afin de mieux comprendre ce que fait un composant et pour vous en inspirer, voici des exemples de niveau simple et avancé.

— Ajouter un logo.

Ce composant permet d'ajouter un logo dans votre Mviewer. Pour vous approprier facilement les composant, vous pouvez commencer par observer et reproduire ce premier composant avec votre propre composant.

— Réaliser un graphique en 3D

Ce composant permet de créer un graphique en 3D avec des boutons et iterfaces de contrôle. Ce composant montre que l'on peut vraiment importer des librairies externes et afficher ce que l'on souhaite dans l'interface Mviewer.

— Filtrer les données dans une fenêtre flottante

Ce composant permet de créer une fenêtre qui contiendra des filtres sur la couche de votre choix. Ici, nous utilisons une librairie externes pour ajouter une fenêtre flottante qui n'existe pas nativement dans le Mviewer.

Ce composant permet ainsi de rajouter une fonctionnalité de filtre sur les données qui n'existe pas nativement dans le Mviewer.

Note : Apprendre par l'exemple :

— *Développer un Custom component*

3.21 Configurer - Les styles utilisés pour la mise en surbrillance des entités sélectionnées

3.21.1 Principe

Les styles sont définis par défaut dans l'application. Il existe un style pour la **sélection** et un style pour la **sous-sélection**. Chaque style définit la répresentation sur la carte des entités de type **point**, **ligne** et **polygone**

Afin de personnaliser ces styles, il faut ajouter un noeud `styles` au même niveau que les noeuds `application`, `baselayers`...

Exemple de configuration complète

```
<styles>
    <selectionstyle>
        <point radius="7" fillcolor="26, 188, 156" opacity="0.5" strokecolor="26, 188, 156" strokewidth="4" />
        <line opacity="0.5" strokecolor="26, 188, 156" strokewidth="4" />
        <polygon fillcolor="26, 188, 156" opacity="0" strokecolor="26, 188, 156" strokewidth="4" />
    </selectionstyle>
    <subselectionstyle>
        <point radius="7" fillcolor="175, 122, 197" opacity="0.5" strokecolor="175, 122, 197" strokewidth="2" />
        <line opacity="0.5" strokecolor="175, 122, 197"  strokewidth="2" />
```

(suite sur la page suivante)

(suite de la page précédente)

```
        <polygon fillcolor="175, 122, 197" opacity="0" strokecolor="175, 122, 197" strokewidth="8" />
    </subselectionstyle>
</styles>
```

Exemple de configuration plus légère

```
<styles>
    <selectionstyle>
        <polygon opacity="0" strokewidth="4" />
    </selectionstyle>
    <subselectionstyle>
        <polygon opacity="0" strokewidth="8" />
    </subselectionstyle>
</styles>
```

3.22 Définir des variables d'environnement

3.22.1 Pour quoi faire ?

Lorsque l'on réalise des cartes dans certains environnements, et que l'on utilise le processus suivant :

1. Développement de la carte en local ==> `localhost:xxxx/mviewer`
2. Passage sur un environnement de test type recette / pré-pod ==> `https://map.recette.fr/mviewer`
3. Déploiement final vers l'environnement de production ===> `https://map.fr/mviewer`

Pour chacun de ces environnements, l'URL change. On doit donc modifier à chaque étape les chemins et URLs des couches dans les XML, customlayers, extensions, customcontrol, etc…

La modification est manuelle et il y a souvent des erreurs. C'est aussi clairement fastidieux pour une carte contenant beaucoup de ressources ou alors lorsque le nombre d'applications à migrer est important.

3.22.2 Concept

Pour faciliter cette transition, un système basé sur un/des fichier(s) `.json` permet de définir des **variables d'environnement**.

A noter que par défaut, un fichier `apps/settings.json` est fourni.

Les variables d'environnement qui peuvent être contenues dans ce type de fichier sont principalement des liens pour accéder aux ressources des couches. Ils sont donc propres à chaque environnement.

Exemple de variable définie dans `apps/settings.json` sur un environnement de recette :

```
{
    "fqdn": "map.recette.fr"
}
```

La même variable définie dans `apps/settings.json` sur un environnement de production :

```
{
    "fqdn": "map.fr"
}
```

Ainsi en définissant des variables communes aux différents environnements mais dont seule la valeur change d'un environnement à l'autre, **il n'est plus nécessaire de changer l'URL** des ressources à la main lors du passage d'un fichier de la préproduction à la production.

3.22.3 Comment faire ?

1. Créer un fichier définissant les variables

Variables globales

Dans le répertoire `/apps`, modifiez (ou créer si inexistant) le fichier `settings.json` et définissez une ou plusieurs variable(s) comme ci-dessous :

```
{
    "fqdn": https://map.recette.fr
}
```

Note : Les variables définies dans le fichier `.json` pourront être appelées dans l'ensemble des applications disponibles dans le répertoire `/apps`.

Variables spécifiques à une application

Si l'on souhaite définir des variables pour une **application spécifique** nommée `MaCarte.xml`, il est possible de créer un fichier d'environnement dédié à cette application. Ainsi, dans le même répertoire que l'application, créez un nouveau fichier nommé `MaCarte.json` (le fichier d'environnement doit avoir le même nom que le fichier de configuration .XML).

Dans ce fichier, définissez une ou plusieurs variable(s) comme ci-dessous comme on le ferez avec le fichier `apps/settings.json` :

```
{
    "fqdnMaCarte": https://MaCarte.recette.fr
}
```

Avertissement : Les variables définies dans ce fichier ne pourront être utilisées **que dans l'application associée**, ici `MaCarte.xml`

Comportement des fichiers d'environnement

- On utilise les variables par défaut du fichier `apps/setings.json`.
- Si aucun fichier `.json` n'existe, alors le système actuel et classique fonctionnera afin de lire simplement le XML sans chercher à remplacer des URLs.
- Si le code détecte un fichier `.json` du même nom que la config (et au même endroit) alors le fichier `apps/settings.json` est surchargé par le fichier `.json` qui accompagne la config (donc `maConfig.json` surchage et remplace les valeurs communes de `apps/settings.json`).
- On peut utiliser l'URL pour préciser le fichier `.json` à utiliser (`e.g &env=apps/test/test.json`) afin de remplacer le fichier `apps/settings.json` par le fichier passé dans l'URL.
- Les `.json` sont ignorés par git via le `.gitignore` pour ne pas écraser les fichiers selon les environnements et ne pas avoir à le réécrire à chaque mise à jour de son projet Git.

2. Mobiliser les variables d'environnement dans le fichier de configuration

Pour appeler les variables dans le fichier de configuration .XML, on utilise la syntaxe Mustache en configurant une couche comme ci-dessous :

```
<layer  url="{{fqdn}}/geoserver/rb/wms"/>
```

Lors de l'appel du XML au chargement de la carte, le code remplacera l'expression `{{fqdn}}` par `https://ma.recette.fr` pour obtenir un XML avec les URLs correctes. On aura alors pour notre <layer> dans le XML final :

```
<layer url="https://ma.recette.fr/geoserver/rb/wms" />
```

Cette syntaxe peut être utilisée pour l'ensemble des ressources mobilisées dans le fichier de configuration .XML telles que des styles, des images, des templates, etc …

3. Mobiliser les variables d'environnement dans un customlayer

Il est également possible d'appeler les variables d'environnement dans un fichier du type customlayer en configurant votre fichier `moncustomlayer.js` comme ci-dessous :

Code source 5 – moncustomlayer.js

```
const url = `https://${mviewer.env?.fqdn}/geoserver/rb/wfs...`;

let layer = new ol.layer.Vector({
        source: new ol.source.Vector({
            url: url,
            format: new ol.format.GeoJSON()
        })
});
```

Avertissement : Pour que cette syntaxe fonctionne, vous devez utiliser des **littéraux de gabarits** ` ` (accent grave au lieu des apostrophes doubles ou simples) pour délimiter l'url.

3.22.4 Cas d'usage

Exemple 1

J'ai une application sur une préprod qui a le domaine : `map.recette.fr` Ma carte mviewer utilise un flux WMS de préprod et un customLayer avec un flux WFS de préprod. Je souhaite pouvoir changer facilement vers une plateforme de production qui a le domaine : `map.fr`

Alors je saisis sur ma recette `apps/settings.json` une variable aléatoire que j'appelle `fqdn` :

```
{
    "fqdn": "map.recette.fr"
}
```

et je saisis sur mon serveur de production dans `apps/settings.json` la même variable avec la valeur qui convient :

```
{
    "fqdn": "map.fr"
}
```

Dans mon config XML, j'utilise alors pour mon WMS la syntaxe Mustache `{{fqdn}}` :

```
 <layer  url="{{fqdn}}/geoserver/rb/wms"/>
```

Dans mon customLayer, je peux par ailleurs appeler directement l'URL WFS via :

```
const url = `https://${mviewer.env?.fqdn}/geoserver/rb/wfs...`;
```

Exemple 2

Prenons ces 3 fichiers et le contenu associé :

— apps/settings.json

```
{
    "version": "3.8.1-snapshot"
}
```

— demo/test/test.json

```
{
    "fqdn": "https://fqdn.com"
}
```

— demo/test/test.xml

En chargeant la carte mviewer pour la configuration demo/test/test.xml, je peux déjà ouvrir la console et voir mes variables et valeurs associées en saisissant en JavaScript mviewer.env afin de les consulter :

```
{
    "fqdn": "https://fqdn.com",
    "version": "3.8.1-snapshot"
}
```

On comprend là que mviewer.env permet d'accéder aux variables depuis un customlayer, le coeur mviewer ou une extension.

Si je souhaite à présent remplacer la valeur de la variable version fournie dans mon fichier apps/settings.json par défaut, je n'aurais qu'à l'ajouter avec la nouvelle valeur (ex. : inconnu) dans le fichier demo/test/test.json :

```
{
    "fqdn": "https://fqdn.com",
    "version": "inconnu"
}
```

On pourra ici encore obtenir via la console du navigateur la liste des variables et les valeurs avec le code JavaScript mviewer.env :

```
{
    "fqdn": "https://fqdn.com",
    "version": "inconnu"
}
```

CHAPITRE 4

Documentation développeur

Cette partie est dédiée aux personnes qui souhaite développer des fonctionnalités pour mviewer.

4.1 Développer avec mviewer

Note : Mviewer est développé en `javascript` et utilise les librairies suivantes :
- Openlayers version 6.3.1
- Fuse version 5.1.0
- I18njs no version
- Bootstrap version 3.3.6
- jQuery version 1.10.2
- Sortable version 1.4.2
- Mustache version 2.3.0

4.1.1 Chapitres abordés

- Intro : *Les grands principes de mviewer*
- Partie 1 : « *Développer ses propres composants* ».
- Partie 2 : Découvrir « *Les fonctions publiques de mviewer* ».

Les grands principes de mviewer

MVIEWER s'appuie sur les 2 principes suivants :

Chargements à la demande

mviewer s'initialise avec un fichier de configuration - **config.xml**. Ce fichier contient les paramétrages nécessaires à l'application ainsi que les ressources utiles à charger :

Astuce

Pour visualiser dans votre navigateur tous les flux mviewer, activez la console - touche `F12` avant le démarrage de l'application.

- résumé des fiches de métadonnées - .xml
- templates de couche personnalisés - .mst
- fichier d'aide personnalisé - .html
- feuille de style personnalisée - .css
- données - (formats multiples)
- extensions - .js
- composants personnalisés (.js .css .html)

Tout est personnalisable

Si mviewer est avant tout une application aux nombreux paramétrages possibles, il peut être nécessaire de personnaliser l'interface voire de créer des couches personnalisées avec leurs propres contrôles (liste déroulante, slider, calendrier…). Les couches personnalisées sont par exemple nécessaires pour appliquer une analyse thématique sur une source de données de type vecteur ou développer un nouveau type d'intéraction entre l'utilisateur et une source de données (mise à jour par exemple).

Sans développement

- La mise en forme de la fiche d'information des données « *Configurer - Templates* »
- La représentation des données servies par WMS - SLD

Avec développement

- couche personnalisée - JavaScript
- contrôle personnalisé - JavaScript + HTML
- composant personnalisé - JavaScript + HTML + CSS

4.2 Développer ses propres composants

Il est possible de développer 3 types de composants dans mviewer en s'appuyant sur la le socle natif de mviewer et sans modification de son coeur.

Tableau 1 – Title

Type	Portée	Affichage	Lien
customLayer	couche identifiée par son id	Carte	*détail*
customControl	couche identifiée par son id	Légende de la couche	*détail*
customComponent	—	—	*détail*

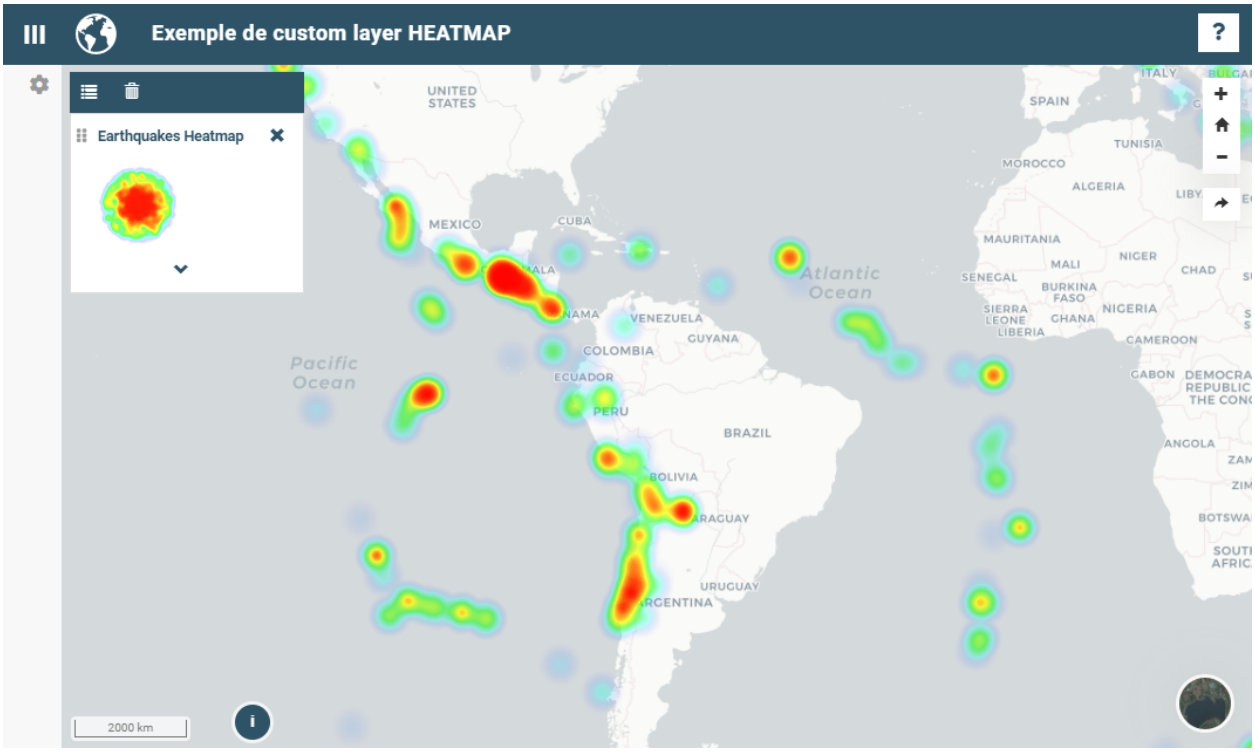


Fig. 1 – Exemple de customlayer heatmap

Note : Pour aller plus loin :
- *Développer un customLayer*
- *Développer un customControl*
- *Développer un Custom component*

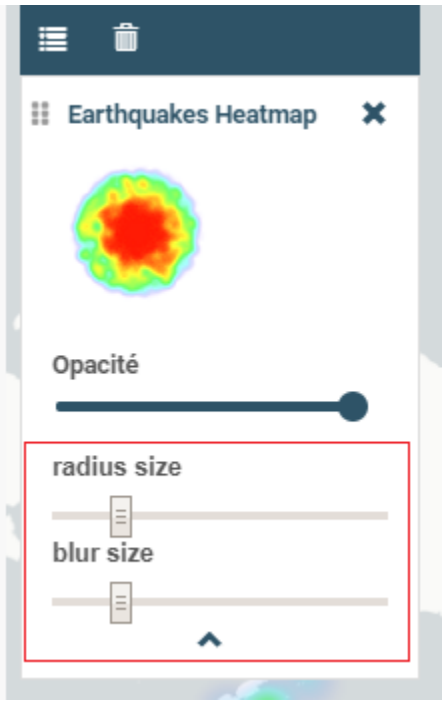


FIG. 2 – Exemple de customcontrol heatmap

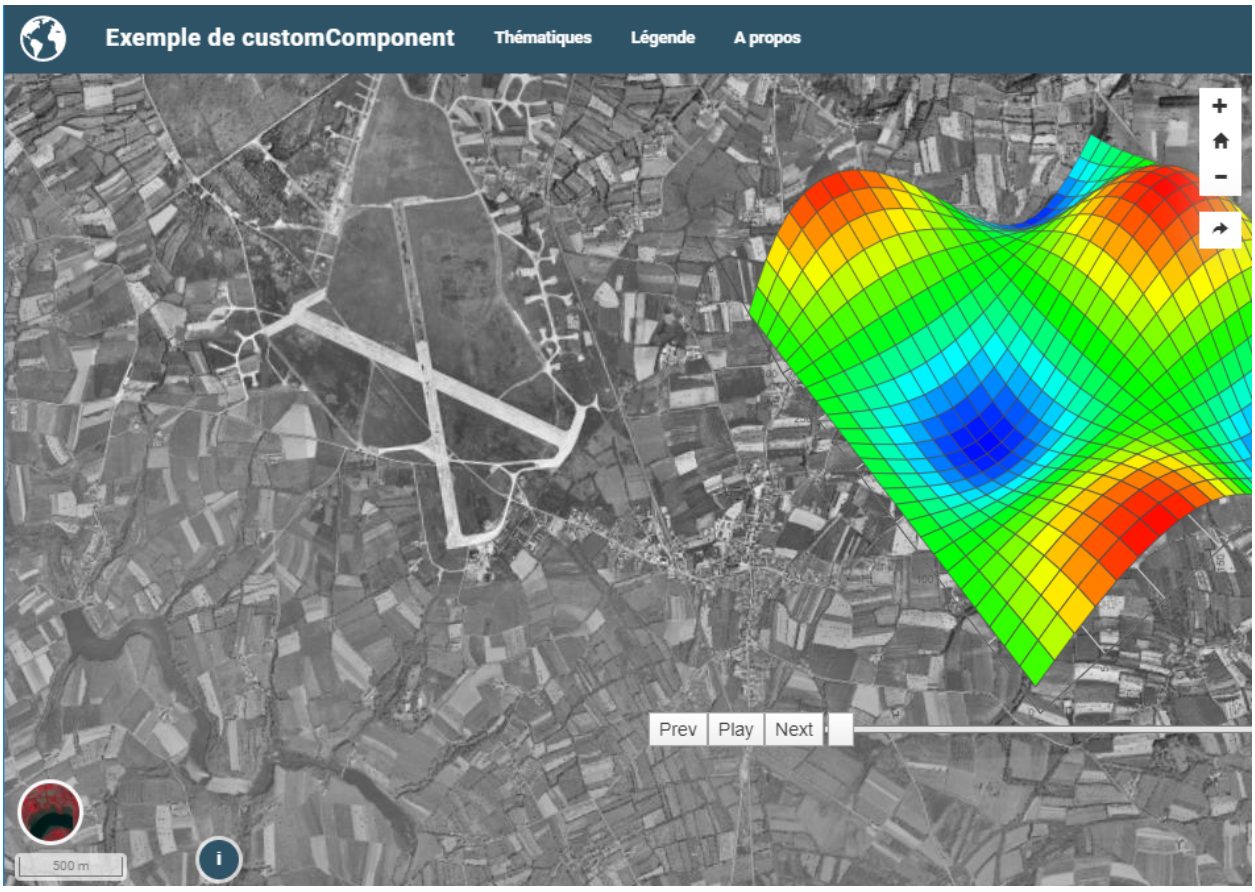
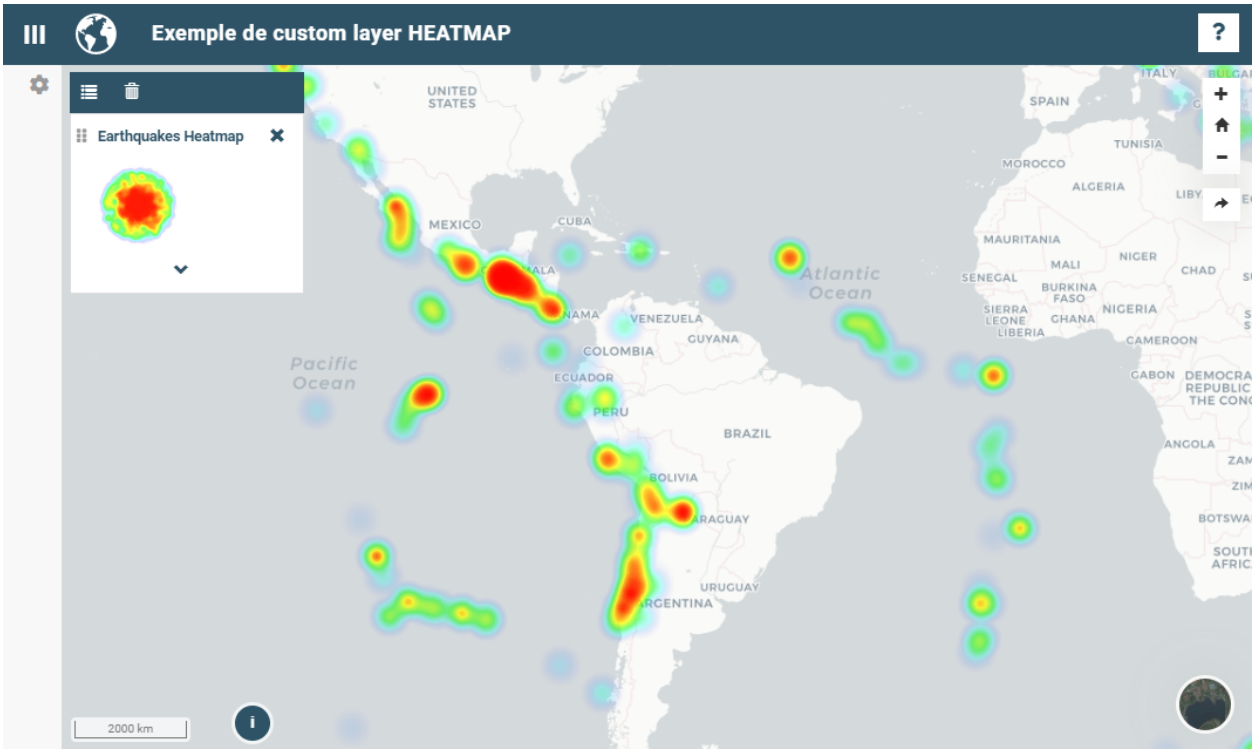


FIG. 3 – Exemple de customcomponent 3d

4.3 Développer un customLayer



L'ojectif est ici de créer une couche personnalisée de type `heatmap` à partir d'un fichier **KML** en utilisant la librairie `Openlayers` et en s'inspirant de cet exemple.. Avant tout, il faut préparer la structure de fichiers qui convient.

Créez un répertoire avec :

- fichier image pour la légende
- fichier de configuration
- fichier customlayer
- les données

```
/ demo
|
|── heatmap
|       |
|       |── config.xml
|       |── customlayer.js
|       |── legend.png
|       |── data
|           |── 2012_Earthquakes_Mag5.kml
|
```

L'exemple complet est disponible sur github.

4.3.1 Ecrire le customLayer

Astuce : L'ancienne méthode fonctionne toujours mais est plus verbeuse que la nouvelle. Les 3 lignes mises en surbrillance sont remplacées par une seule ligne, également mise en surbrillance, dans la nouvelle version.

Ancienne méthode

Code source 1 – customlayer.js

```
//Layerid is the same than the layerid in config.xml
const layerid = "heatmap";
//Create customlayer object
mviewer.customLayers[layerid] = {};
//Create Openlayers heatmap layer
const layer = new ol.layer.Heatmap({
    source: new ol.source.Vector({
        url: 'demo/heatmap/data/2012_Earthquakes_Mag5.kml',
        format: new ol.format.KML({
            extractStyles: false
        })
    }),
    blur: 10,
    radius: 10,
    weight: function(feature) {
        var name = feature.get('name');
        var magnitude = parseFloat(name.substr(2));
        return magnitude - 5;
    }
});
// Set the openlayers layer to the customLayer object layer
mviewer.customLayers[layerid].layer = layer;
```

Nouvelle méthode

Depuis la version **3.2** de mviewer, une classe `CustomLayer` a été développée afin de faciliter la saisie de nouveaux customLayers :

Code source 2 – customlayer.js

```
const layer = new ol.layer.Heatmap({
    source: new ol.source.Vector({
        url: 'demo/heatmap/data/2012_Earthquakes_Mag5.kml',
        format: new ol.format.KML({
            extractStyles: false
        })
    }),
    blur: 10,
    radius: 10,
    weight: function(feature) {
```

(suite sur la page suivante)

(suite de la page précédente)

```
        var name = feature.get('name');
        var magnitude = parseFloat(name.substr(2));
        return magnitude - 5;
    }
});
new CustomLayer('heatmap', layer);
```

4.3.2 Ecrire le config.xml

Dans le fichier de configuration, il faut reprendre l'id du customlayer `id="heatmap"`, préciser `type="customlayer"` ainsi que l'URL du fichier `url="demo/heatmap/customlayer.js"` :

Code source 3 – config.xml

```
<layer id="heatmap"
    name="Earthquakes Heatmap"
    visible="true"
    type="customlayer"
    legendurl="demo/heatmap/legend.png"
    opacity="1"
    url="demo/heatmap/customlayer.js"
    attribution=""
    metadata=""
    metadata-csw="">
</layer>
```

4.3.3 Exemples de customLayers

En respectant les consignes détaillées plus haut, voici quelques exemples prêts à l'emploi.

customlayer ArcGis REST Feature

Code source 4 – test_esri_college_80.js - ancienne version

```
{
    mviewer.customLayers.test_esri_college_80 = {};

    var esrijsonFormat = new ol.format.EsriJSON();

    mviewer.customLayers.test_esri_college_80.layer = new ol.layer.Vector({
        source: new ol.source.Vector({
            url: 'https://services2.arcgis.com/aYGUaoapooTe49i1/arcgis/rest/services/
↪COLLEGES_PUBLICS_avec_Liens/FeatureServer/0/query?where=&objectIds=&time=&
↪geometry=154415%2C6384802%2C345565%2C6494181&geometryType=esriGeometryEnvelope&
↪inSR=3857&spatialRel=esriSpatialRelIntersects&resultType=none&distance=0.0&
↪units=esriSRUnit_Meter&returnGeodetic=false&outFields=*&returnGeometry=true&
↪featureEncoding=esriDefault&multipatchOption=xyFootprint&maxAllowableOffset=&
↪geometryPrecision=&outSR=2154&datumTransformation=&applyVCSProjection=false&
↪returnIdsOnly=false&returnUniqueIdsOnly=false&returnCountOnly=false&
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪returnExtentOnly=false&returnQueryGeometry=false&returnDistinctValues=false&
↪cacheHint=false&orderByFields=&groupByFieldsForStatistics=&outStatistics=&having=&
↪resultOffset=&resultRecordCount=&returnZ=false&returnM=false&
↪returnExceededLimitFeatures=true&quantizationParameters=&sqlFormat=none&f=pjson&token=
↪',
            format: esrijsonFormat
        }),
        style: new ol.style.Style({
            image: new ol.style.Circle({
                fill: new ol.style.Fill({
                    color: 'rgba(255, 118, 117,1.0)'
                }),
                stroke: new ol.style.Stroke({
                    color: "#ffffff",
                    width: 4
                }),
                radius: 9
            })
        })
    });
    mviewer.customLayers.test_esri_college_80.handle = false;

}
```

Code source 5 – test_esri_college_80.js - nouvelle version

```
let esrijsonFormat = new ol.format.EsriJSON();

const layer =  new ol.layer.Vector({
    source: new ol.source.Vector({
        url: 'https://services2.arcgis.com/aYGUaoapooTe49i1/arcgis/rest/services/
↪COLLEGES_PUBLICS_avec_Liens/FeatureServer/0/query?where=&objectIds=&time=&
↪geometry=154415%2C6384802%2C345565%2C6494181&geometryType=esriGeometryEnvelope&
↪inSR=3857&spatialRel=esriSpatialRelIntersects&resultType=none&distance=0.0&
↪units=esriSRUnit_Meter&returnGeodetic=false&outFields=*&returnGeometry=true&
↪featureEncoding=esriDefault&multipatchOption=xyFootprint&maxAllowableOffset=&
↪geometryPrecision=&outSR=2154&datumTransformation=&applyVCSProjection=false&
↪returnIdsOnly=false&returnUniqueIdsOnly=false&returnCountOnly=false&
↪returnExtentOnly=false&returnQueryGeometry=false&returnDistinctValues=false&
↪cacheHint=false&orderByFields=&groupByFieldsForStatistics=&outStatistics=&having=&
↪resultOffset=&resultRecordCount=&returnZ=false&returnM=false&
↪returnExceededLimitFeatures=true&quantizationParameters=&sqlFormat=none&f=pjson&token=
↪',
        format: esrijsonFormat
    }),
    style: new ol.style.Style({
        image: new ol.style.Circle({
            fill: new ol.style.Fill({
                color: 'rgba(255, 118, 117,1.0)'
            }),
            stroke: new ol.style.Stroke({
                color: "#ffffff",
```

(suite sur la page suivante)

(suite de la page précédente)

```
                width: 4
            }),
            radius: 9
        })
    })
});

new CustomLayer('test_esri_college_80', layer);
```

Note : Pour aller plus loin :
- *Appronfondir - Custom Layer*
- *Les fonctions publiques de mviewer*

4.4 Développer un customControl

Les **Customs Controls** permettent de créer des interactions entre l'utilisateur et un layer. Si on souhaite créer une nouvelle fonctionnalité en lien avec une couche particulière, c'est la solution à privilégier. Le mécanisme de customControl permet le développement à façon de solution nouvelle sans modifier le coeur de mviewer.

Convention

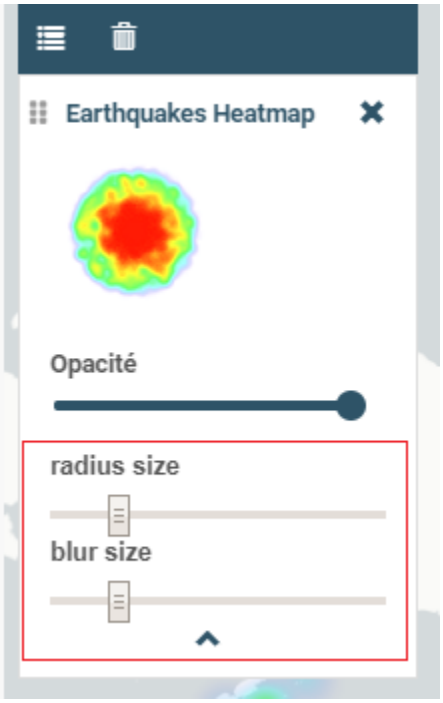
Un custom control consiste en deux fichiers - **JavaScript** et **HTML**. Ces deux fichiers doivent impérativement respecter la règle de nommage suivante :

- **layerid**.html
- **layerid**.js
- *où layerid correspond à l'id du layer tel que définit dans le config.xml.*

L'emplacement de ces fichiers doit être précisé dans le config.xml avec le paramètre `customcontrolpath=""`

Exemple :

Le custom control - encadré rouge - s'affiche par défaut dans la légende en dessous des attributions :



4.4.1 Ecrire le code - html

Le bloc écrit en HTML s'affichera dans le panneau légende associé à la couche. Il est possible de créer des éléments HTML qui permettront à l'utilisateur d'interagir avec la couche associée. Dans l'exemple suivant, on affiche deux sliders qui vont nous permettre de modifier dynamiquement l'affichage de notre couche.

Code source 6 – customcontrol.html

```
<form>
    <label>radius size</label>
    <input id="heatmap-radius" class="" type="range" min="1" max="50" step="1" value="10
↪"/>
    <label>blur size</label>
    <input id="heatmap-blur" type="range" min="1" max="50" step="1" value="10"/>
</form>
```

Tooltip

Si vous souhaitez utiliser une infobulle (tooltip), vous pouvez utiliser les paramètres *title* et *tooltip* :

Code source 7 – customcontrol.html

```
<button tooltip="top,hover,true,body,mviewer.templates.tooltip" title="Déplacement en␣
↪voiture">
```

Tableau 2 – Paramètres à utiliser :

Paramètre	Description	Exemple
tooltip	Permet de définir les options de la tooltip	<placement - string>, <trigger - string>, <html - boolean>, <container - string>, <template, string>
title	Texte affiché	title="Déplacement en voiture"

4.4.2 Ecrire le code - JavaScript

Ancienne méthode

La ligne 2 surlignée indique la notation propre à l'ancienne méthode.

Code source 8 – customcontrol.js

```
1  const layerid = "heatmap";
2  mviewer.customControls[layerid] = (function() {
3
4      /*
5      * Private
6      */
7
8      var _initialized = false;
9
10     var _layer;
11
12     var _blurElement = false;
13
14     var _radiusElement = false;
15
16     var _blurHandler = function(e) {
17         _layer.setBlur(parseInt(e.target.value, 10));
18     };
19
20     var _radiusHandler = function(e) {
21         _layer.setRadius(parseInt(e.target.value, 10));
22     };
23
24     return {
25         /*
26         * Public
27         */
28
29         init: function () {
30             // mandatory - code executed when layer is added to legend panel
31             if (!_initialized) {
32                 _layer = mviewer.getLayer(layerid).layer;
33                 _blurElement = document.getElementById('heatmap-blur');
34                 _radiusElement = document.getElementById('heatmap-radius');
35                 if (_blurElement && _radiusElement) {
36                     _blurElement.addEventListener('change', _blurHandler);
37                     _radiusElement.addEventListener('change', _radiusHandler);
38                     _initialized = true;
39                 }
40
41             }
42         },
43
44         destroy: function () {
45             // mandatory - code executed when layer panel is closed
```

(suite sur la page suivante)

(suite de la page précédente)

```
            _initialized = false;
        }
    };

}());
```

Nouvelle méthode

Depuis la version **3.2** de mviewer, une classe `CustomControl` a été développée afin de faciliter la saisie de nouveaux customControls. Les 2 lignes surlignées (2, 51) indiquent les lignes modifiées par rapport à l'ancienne méthode.

Code source 9 – customcontrol.js

```
const layerid = "heatmap";
const cc = (function() {

    /*
    * Private
    */

    var _initialized = false;

    var _layer;

    var _blurElement = false;

    var _radiusElement = false;

    var _blurHandler = function(e) {
        _layer.setBlur(parseInt(e.target.value, 10));
    };

    var _radiusHandler = function(e) {
        _layer.setRadius(parseInt(e.target.value, 10));
    };

    return {
        /*
        * Public
        */

        init: function () {
            // mandatory - code executed when layer is added to legend panel
            if (!_initialized) {
                _layer = mviewer.getLayer(layerid).layer;
                _blurElement = document.getElementById('heatmap-blur');
                _radiusElement = document.getElementById('heatmap-radius');
                if (_blurElement && _radiusElement) {
                    _blurElement.addEventListener('change', _blurHandler);
                    _radiusElement.addEventListener('change', _radiusHandler);
                    _initialized = true;
```

(suite sur la page suivante)

(suite de la page précédente)

```
                }

            }
        },

        destroy: function () {
            // mandatory - code executed when layer panel is closed
            _initialized = false;
        }
    };

}());
new CustomControl(layerid, cc.init, cc.destroy);
```

Avertissement : Si on souhaite disposer d'un bloc de code public, il faut remplacer la ligne `const cc = (function() {` par `var cc = (function() {`

4.4.3 Ecrire le config.xml

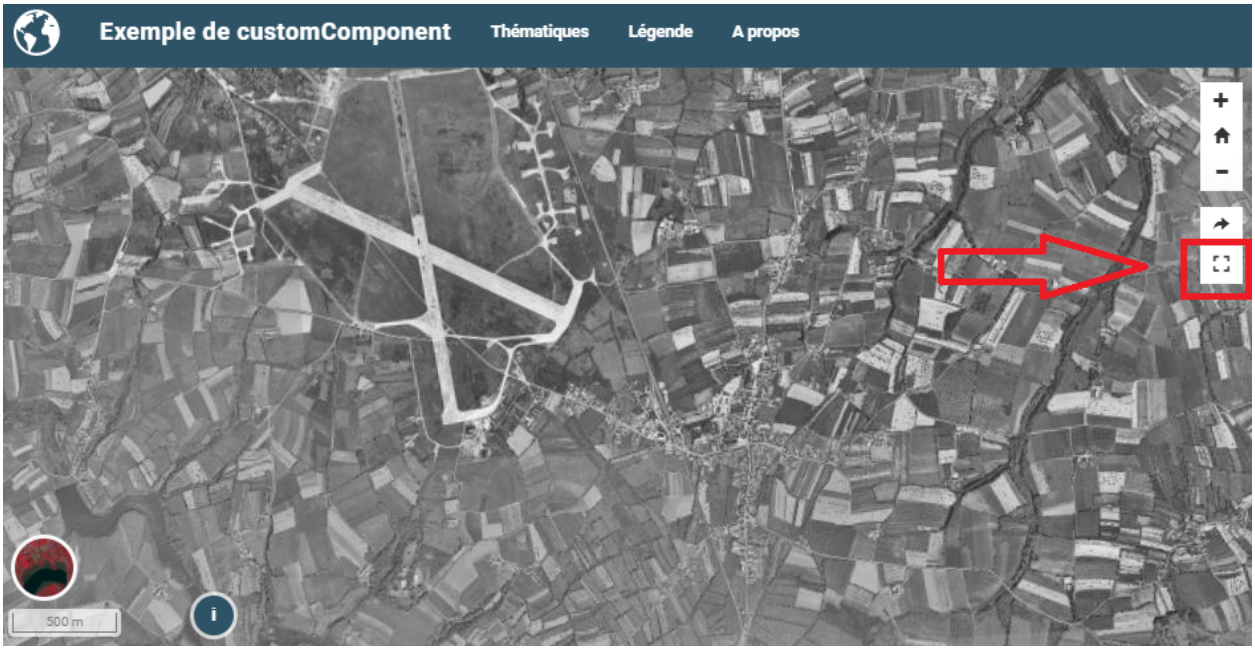
Dans le fichier de configuration, à partir de l'exemple customLayer **heatmap**, il faut ajouter les 3 lignes mises en surbrillance.

Code source 10 – config.xml

```
<layer id="heatmap"
    name="Earthquakes Heatmap"
    visible="true"
    url="demo/heatmap/customlayer.js"
    queryable="true"
    type="customlayer"
    customcontrol="true"
    customcontrolpath="demo/heatmap/control"
    legendurl="demo/heatmap/legend.png"
    opacity="1"
    expanded="true"
    attribution=""
    metadata=""
    metadata-csw="">
</layer>
```

Note : Pour aller plus loin :
— *Approfondir - Custom Control*
— *Les fonctions publiques de mviewer*

4.5 Développer un Custom component



L'ojectif est ici de créer un bouton dont la fonction est d'afficher la carte en mode « plein écran » en utilisant l'API HTML 5 `requestFullscreen`.

Créez un répertoire avec :

- fichier HTML
- fichier JavaScript
- fichier de configuration
- fichier de style

```
/ demo/addons
        |
        |-- fullscreen
        |       |
        |       |-- config.json
        |       |-- fullscreen.js
        |       |-- fullscreen.html
        |       |-- style.css
```

L'exemple complet est disponible sur github.

4.5.1 Ecrire le code html

Le code html est la partie visible du composant. Le code HTML sera intégré par mviewer et « dessiné » dans la `div` ciblée via le fichier config.json. Dans le cas présent on créé un simple bouton avec une icône **fontawesome** :

Code source 11 – fullscreen.html

```
<a class="btn btn-default btn-raised" type="button" id="fullscreen-btn">
    <span class="fas fa-expand"></span>
</a>
```

4.5.2 Ecrire le code javascript

Le code JavaScript est la partie logique de notre composant. Dans l'exemple ci-dessous, on associe une fonction à l'évènement `click` du bouton créé précédemment.

Code source 12 – fullscreen.js

```
const fullscreen = (function() {

    var _btn;
    var _fullscreen = function (e) {
        document.getElementById("map").requestFullscreen();
    };

    return {

        init : function () {
            _btn = document.getElementById("fullscreen-btn");
            _btn.addEventListener('click', _fullscreen);
        }
    };

})();

new CustomComponent("fullscreen", fullscreen.init);
```

Avertissement : Si on souhaite disposer d'un bloc de code publique, il faut remplacer la ligne `const fullscreen = (function() {` par `var fullscreen = (function() {`

4.5.3 Ecrire le code CSS

Le code CSS permet d'affiner le style de notre bouton.

Code source 13 – style.css

```
#fullscreen-btn {
    border-radius: 0px;
    padding: 5px 10px 5px 10px;
}
```

4.5.4 Ecrire le config.json

Dans le fichier de configuration - **config.json** - , il faut référencer toutes les ressources utiles. le paramètre `target` permet de cibler la `div` dans laquelle le composant sera affiché.

Code source 14 – config.json

```
{
    "js": ["fullscreen.js"],
    "css": "style.css",
    "html": "fullscreen.html",
    "target": "toolstoolbar"
}
```

4.5.5 Ecrire le config.xml

Dans le fichier de configuration, il faut ajouter la ligne en surbrillance.

Code source 15 – config.xml

```
<extensions>
    <extension type="component" id="fullscreen" path="demo/addons"/>
</extensions>
```

Note : Pour aller plus loin :
— *Les fonctions publiques de mviewer*

4.6 Les fonctions publiques de mviewer

Pour accéder à ces fonctions publiques, il faut simplement utiliser l'objet `mviewer` et accéder à une fonction (`mviewer.nomDeLaFonction()`).

Il existe déjà les fonctions suivantes :

mviewer

getActiveBaseLayer()

Renvoie
L'id du baselayer (couche de fond visible).

setBaseLayer(*id*)

Paramètres
id (*string*) – Id du layer

Renvoie
Affiche le baselayer correspondant à l'id.

getLayer(*layerid*)

Paramètres
layerid (*string*) – Id du layer

Renvoie
La configuration du layer.

`getLayer(layerid).layer`

Renvoie
Le layer (ol.Layer) du layerid.

`getMap()`

Renvoie
La map (ol.Map).

`toggleLayer(layerid)`

Paramètres
layerid (*string*) – Id du layer

Renvoie
Affiche/masque le layer correspondant au layerid.

`removeAllLayers()`

Renvoie
Masque toutes les couches.

`showLocation(projection, x, y)`

Paramètres
— **projection** (*string*) – Projection de la carte
— **x** (*float*) – Coordonnée y
— **y** (*float*) – Coordonnée y

Renvoie
Affiche une punaise sur les coordonnées entrées.

`tr()`

Renvoie
Traduit dans la langue courante de mviewer une valeur de type `machaine.a.traduire` (cf *Configurer - Traduction*) .

`zoomToLocation(x, y, zoom, querymap)`

Paramètres
— **x** (*float*) – Coordonnée y
— **y** (*float*) – Coordonnée y
— **zoom** (*int*) – Zoom de la carte
— **querymap** (*boolean*) – Interrogation de la carte

Renvoie
Zoom aux coordonnées indiquées et en option interroge la carte à ces coordonnées.

`getLayersAttribute(attribute)`

Paramètres
attribute (*string*) – Attribut recherché dans les couches Mviewer

Renvoie
Un objet contenant les couches et la valeur de l'attribut recherchée du type {id1 : value, id2 : null}.

`orderLegendByMap()`

Renvoie
Réordonne la légende selon l'affichage des couches sur la carte.

`orderLegendByMap()`

Renvoie
Réordonne la légende en respectant l'affichage des couches sur la carte.

`reorderLayer`(*layer*, *index*)

Paramètres
- **layer** (*Object*) – Couche OpenLayers depuis la carte (e.g depuis mviewer.getLayer(id)).
- **index** (*int*) – Nouvel index de la couche sur la carte.

Renvoie
Permet de changer l'affichage d'une couche sur la carte (voir openLayers zindex).

`showLayersByAttrOrder`(*layers*, *reverse*)

Paramètres
- **layer** (*Object*) – Dictionnaire des couches avec le zindex pour chacune (ex. {id1 :1, id2 :2,,,})
- **reverse** (*boolean*) – reverse == true pour lire les couches dans le sens inverse de l'ordre de l'objet.

Renvoie
Permet de modifier l'ordre complet des couches de la carte, réordonnera les topLayer et la légende ensuite.

`setLegendLayerPos`(*id*, *position*)

Paramètres
- **id** (*string*) – Identifiant de la couche. Correspond à l'attribut data-layerid d'un élement de la légende.
- **position** (*int*) – Nouvelle position de l'élément de légende ciblé dans la légende.

Renvoie
Permet de modifier l'ordre d'affichage d'un élément de la légende sans impacter l'ordre d'affichage sur la carte.

`orderLayerByIndex`()

Renvoie
Ordonne les éléments en respectant le paramètre index. Les couches sans index seront listées dans l'ordre d'écriture dans le XML. Cela n'impacte pas l'affichage dans la légende.

`orderTopLayer`()

Renvoie
Ordonne les/la couches avec le paramètre toplayer. Cela n'impacte pas l'affichage dans la légende.

`clickedCoordinates`()

Renvoie
Les coordonnées latitude / longitude du point cliqué sur la carte.

4.7 Appronfondir - Custom Layer

Les **Customs Layers** permettent de personnaliser la représentation et les interactions que l'on a avec les layers de façon plus avancée que ce que l'on peut faire avec le fichier de configuration XML.

Dans chacune des deux méthodes présentées dans la suite de cette page les customs layers ont pour base commune la classe `CustomLayer` présente dans le fichier `custom.js`.

Les méthodes suivantes sont à utiliser dans le fichier `maLayer.js` qui est dans l'arborescence suivante :

```
/apps
        ├── ma_carte1
        │   ├── addons
        │   │   ├── couche1
        │   │   │   ├── control
        │   │   │   └── layer
        │   │   │       └── malayer.js
        │   │   └── couche2
        │   │       ├── control
        │   │       └── layer
        │   ├── data
        │   ├── css
        │   ├── sld
        │   ├── img
        │   ├── templates
        │   └── ma_carte1.xml
        └── ma_carte2
```

4.7.1 Première Méthode : Définition Simple

Cette solution permet de rapidement mettre en place un custom layer mais ne permet pas de travailler avec des variables ou des méthodes privées.

```
1  // Définition de la couche représentant le custom layer
2  const aerial = new ol.layer.Tile({
3      source: new ol.source.XYZ({
4          url: 'https://api.maptiler.com/tiles/satellite/{z}/{x}/{y}.jpg?key=' + key,
5          maxZoom: 20
6      })
7  });
8
9  // Votre code ici
10 ...
11
12 /* Créer le custom layer à partir du code ci-dessus */
13 // Le custom layer aura pour id "monCustomLayer" et pour couche aerial définit dans l
   ↪'exemple
14 new CustomLayer("monCustomLayer", aerial);
```

Tous les paramètres de la classe `CustomLayer` ne sont pas nécessaires seul **l'ID** et **la couche (layer)** sont indispensables.

Ajouter des fonctions et des variables

Une fonction privée ne sera pas accessible en dehors du code de la classe alors qu'une fonction publique sera accessible depuis n'importe où ce qui peut entrainer des conflits avec d'autres fonctions de l'application si l'on ne fait pas attention.

Pour les variables et fonctions de classe publique

Il faut définir un nouvel attribut pour la classe `CustomControl` de la manière suivante :

```
  ...
  // Initialiser l'objet avec les fonctions init() et destroy() et l'id de couche
↪"monLayer".
  var monLayer = new CustomLayer("monLayer",aerial);

  // Une fois créé on peut ajouter des propriétés (une propriété peut être une fonction␣
↪ou une variable)

  // Ajouter une fonction
  monLayer.maNouvelleFonction = function(){
      // Votre Code ici
      ...
  }

  // Ajouter une variable
  monLayer.maNouvelleVariable = "je suis un exemple";
```

Ces attributs seront alors publics et accessibles depuis l'extérieur.

4.7.2 Deuxième Méthode : Création d'une sous-classe

La création d'une sous-classe permet de mieux structurer le code et de faciliter l'ajout de fonctions et variables privées sans avoir à modifier la classe parent `CustomLayer`. Ell se présente comme suit :

```
  const aerial = new ol.layer.Tile({
      source: new ol.source.XYZ({
          url: 'https://api.maptiler.com/tiles/satellite/{z}/{x}/{y}.jpg?key=' + key,
          maxZoom: 20
      })
  });
  // Classe qui étend la classe 'CustomLayer' et décrit le custom Layer
  class MonCustomLayer extends CustomLayer {

      // Initialiser le custom layer
      constructor(id, layer, legend, handle = false) {

          // Initialiser les attributs de la classe parent
          super(id, layer, legend, handle);

      }
  }
  // Créer le Custom Layer
  new MonCustomLayer("monCustomLayer",aerial);
```

La classe `MonCustomLayer` hérite de la classe `CustomLayer` et doit donc utiliser le `constructor()` parent pour créer un objet en spécifiant au minimum **l'ID** et **la layer**.

Ajouter des fonctions et des variables

Pour empêcher de potentiels bugs on peut ajouter à la classe `MonCustomLayer` (vue dans la partie précédente) des fonctions privées ou publiques.

Une fonction privée ne sera pas accessible en dehors du code de la classe alors qu'une fonction publique sera accessible depuis n'importe où ce qui peut entraîner des conflits avec d'autres fonctions de l'application si l'on ne fait pas attention.

Pour les variables et fonctions de classe publique

Directement en ajoutant dans le code de la classe `MonCustomLayer` :

```
// Classe qui étend la classe et décrit le custom Layer
class MonCustomLayer extends CustomLayer {
     constructor(id, layer, legend, handle = false, nouvelleVariable) {

         // Initialiser les attributs de la classe parent
         super(id, layer, legend, handle);

         // Ajout d'une variable publique qui prend en valeur le paramètre de constructor "nouvelleVariable"
         this.nouvelleVariable = nouvelleVariable;

     }
     maFonctionPublique(){
         // Votre code ici
         ...
     }
}
// Créer l'objet layer avec l'id 'monLayer' qui est le nom de la couche
new MonCustomLayer("monLayer",aerial);
```

Cette fonction sera appelable grâce à `monobjet.maFonctionPublique()` et l'on peut bien sûr y passer des paramètres.

Concernant la variable elle est de la même façon accessible grâce à `monobjet.nouvelleVariable`.

Pour les variables et fonctions de classe privée

Une fonction privée est définie en dehors du code de la classe `MonCustomLayer` et déclarée comme une `constante` :

```
// Fonction privée non utilisable en dehors de ce code
const maFonctionPrivée = function(){
    // Votre code ici
    ...
}
// Classe qui étend la classe et décrit le custom Layer
class MonCustomLayer extends CustomLayer {
    ...
```

(suite sur la page suivante)

(suite de la page précédente)

```
    maFonctionPublique(){
        maFonctionPrivée();
        // Votre code ici
        ...
    }
}
// Créer l'objet layer avec l'id 'monLayer' qui est le nom de la couche
new MonCustomLayer("monLayer",aerial);
```

Cette fonction sera appelable grâce à `maFonctionPrivée()` seulement dans ce bout de code et donc on peut par exemple l'utiliser dans une fonction publique (ici `maFonctionPublique()`).

Pour ajouter une variable de classe privée il faut ajouter le « # » avant le nom de la variable et la déclarer avant la fonction `constructor()` :

```
...
// Classe qui étend la classe et décrit le custom Layer
class MonCustomLayer extends CustomLayer {

    // Déclaration de la variable Privée
    #maVariablePrivee;
    constructor(id, layer, legend, handle = false,maVariablePrivee = "valeurParDefaut") {

        // Initialiser les attributs de la classe parent
        super(id, layer, legend, handle);

        // Initialiser #maVariablePrivee
        this.#maVariablePrivee = maVariablePrivee

        ...
    }
    ...
}
// Initialiser un objet avec la chaine de caractères "maVariablePrivee" dans la variable␣
↪de classe privée #maVariablePrivee et l'id de couche "monLayer".
new MonCustomLayer("monLayer",aerial);
```

Si vous voulez quand pouvoir accéder et modifier la valeur de cette variable en dehors de ce code mais de manière plus sécuriser il faut déclarer une fonction `get()` pour récupérer la valeur et une fonction `set(valeur)` pour la modifier :

```
...
// Classe qui étend la classe et décrit le custom Layer
class MonCustomLayer extends CustomLayer {

    // Déclaration de la variable Privée
    #maVariablePrivee;

    constructor(id, layer, legend, handle = false,maVariablePrivee = "valeurParDefaut") {

        // Initialiser les attributs de la classe parent
        super(id, layer, legend, handle);
```

(suite sur la page suivante)

(suite de la page précédente)

```
        // Initialiser #maVariablePrivee
        this.#maVariablePrivee = maVariablePrivee

        ...
    }

    // Fonction pour récupérer la valeur de #maVariablePrivee
    getMaVariablePrivee(){
        return this.#maVariablePrivee;
    }

    // Fonction pour modifier la valeur de #maVariablePrivee
    setMaVariablePrivee(valeur){
        this.#maVariablePrivee = valeur;
    }
}
// Initialiser un objet avec la chaîne de caractères "maVariablePrivee" dans la variable␣
↪de classe privée #maVariablePrivee et l'id de couche "monLayer".
new MonCustomLayer("monLayer",aerial);
```

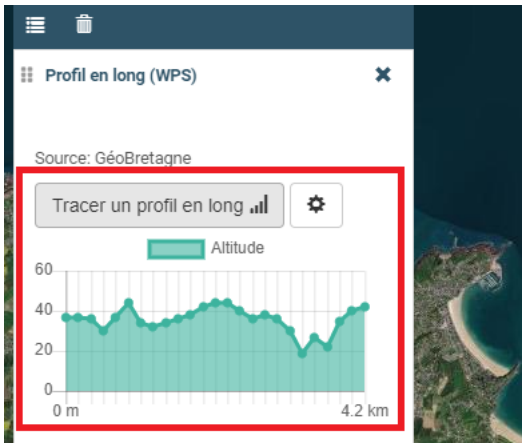
4.7.3 Interactions customLayer et mviewer

Depuis le customLayer il est possible de communiquer et d'interagir avec la carte et d'une façon plus générale avec mviewer. Vous pouvez ainsi mobiliser toutes les méthodes publiques dans votre développement. Pour en savoir plus, consultez, dans la documentation développeur, la partie « *Les fonctions publiques de mviewer* ».

4.8 Appronfondir - Custom Control

Les **Customs Controls** permettent de personnaliser la représentation et les interactions que l'on a avec les layers de façon plus avancée que ce que l'on peut faire avec le fichier de configuration XML.

Voici un exemple, le custom control s'affiche dans la légende en dessous des attributions :



4.8.1 Première méthode : Définition Simple (CustomControl)

Cette méthode permet de créer des customs controls plus simples qu'avec la deuxième méthode mais permet moins de personnalisation.

Dans ce cas précis on utilise la classe de base `CustomControl` dans le fichier `custom.js`.

La classe possède une méthode `constructor()` qui prend en paramètre les méthodes `init()` et `destroy()` que l'on peut définir dans le fichier JavaScript (dans cet exemple `moncontrol.js`) présent dans l'arborescence suivante :

```
/apps
    ├── ma_carte1
    │   ├── addons
    │   │   ├── couche1
    │   │   │   ├── control
    │   │   │   │   ├── moncontrol.html
    │   │   │   │   └── moncontrol.js
    │   │   │   └── layer
    │   │   └── couche2
    │   │       ├── control
    │   │       └── layer
    │   ├── data
    │   ├── css
    │   ├── sld
    │   ├── img
    │   ├── templates
    │   └── ma_carte1.xml
    └── ma_carte2
```

Pour cela il faut donc définir les deux fonctions en les déclarant avec le mot-clé **const** pour les rendre inaccessibles depuis le reste de l'application indépendamment du custom control dans lequel elles sont définies, puis il faudra les donner en paramètre à la classe `CustomControl` vue plutôt.

```
1  const init = function() {
2  // Obligatoire - code exécuté quand le panel est ouvert
3  // Votre code ici
4  ...
5
6  };
7
8  const destroy =  function() {
9      // Obligatoire - code exécuté quand le panel se ferme
10     // Votre code ici
11     ...
12 }
13 // Initialiser l'objet avec les fonctions init() et destroy() et l'id de couche
   ↪"monControl".
14 new CustomControl("monControl", init, destroy);
```

Ajouter des fonctions et des variables

Une **fonction/variable privée** ne sera pas accessible en dehors du code de la classe alors qu'une **fonction/variable publique** sera accessible depuis n'importe où ce qui peut entrainer des conflits avec d'autres fonctions/variables de l'application si l'on ne fait pas attention.

Pour les variables et fonctions de classe publique

Il faut définir un nouvel attribut pour la classe `CustomControl` de la manière suivante :

```
  ...
  // Initialiser l'objet avec les fonctions init() et destroy() et l'id de couche
↪"monControl".
  var monControl = new CustomControl("monControl", init, destroy);

  // Une fois créer on peut ajouter des propriétés (une propriété peut être une fonction␣
↪ou une variable)

  // Ajouter une fonction
  monControl.maNouvelleFonction = function(){
      // Votre Code ici
      ...
  }

  // Ajouter une variable
  monControl.maNouvelleVariable = "je suis un exemple";
```

Ces attributs seront alors publics et accessibles depuis l'éxterieur.

4.8.2 Deuxième méthode : Création d'une sous-classe (AdvancedCustomControl)

Cette méthode est la plus complète des deux et permet de créer des customs controls plus poussés.

Tous les Custom Control ont une base commune dans le fichier `custom.js` où est définie la classe `AdvancedCustomControl`.

Pour utiliser cette classe il faut modifier le fichier Javascript (dans cet exemple `moncontrol.js`) présent dans l'arborescence suivante :

```
/apps
    ├── ma_carte1
    │   ├── addons
    │   │   ├── couche1
    │   │   │   ├── control
    │   │   │   │   ├── moncontrol.html
    │   │   │   │   └── moncontrol.js
    │   │   │   └── layer
    │   │   └── couche2
    │   │       ├── control
    │   │       └── layer
    │   ├── data
    │   ├── css
    │   ├── sld
```

(suite sur la page suivante)

(suite de la page précédente)

```
    │   ├── img
    │   ├── templates
    │   └── ma_carte1.xml
    └── ma_carte2
```

Ce fichier définit une classe qui étend (est un héritier de la classe) la classe `AdvancedCustomControl` :

```
// Classe qui étend la classe abstraite et décrit le custom Control
class MonControl extends AdvancedCustomControl {
    constructor(id) {
        // Initialise l'id de l'objet avec le constructeur parent
        super(id);
    }
    // Obligatoire - ce code est exécuté lors de l'ouverture du panel
    init() {
        // Votre code ici
        ...
    }
    // Obligatoire - ce code est exécuté lors de la fermeture du panel
    destroy() {
        // Votre code ici
        ...
    }
}
```

La classe `AdvancedCustomControl` étant **abstraite** cela signifie qu'elle nous oblige à redéfinir les fonctions `init()` et `destroy()` qui sont obligatoires sinon elle nous renvoie une erreur.

De plus la fonction `constructor(id)` permet à l'objet d'être initialisé avec la valeur **id (obligatoire)** lors de la création d'un **objet MonControl**.

Pour créer cet objet et le rendre disponible au reste de l'application il faut rajouter le code suivant :

```
// Créer l'objet MonControl avec l'id 'monControl' qui est le nom de la couche
new MonControl("monControl");
```

Ajouter des fonctions

Pour empêcher de potentiels téléscopage de variables ou de méthodes on peut ajouter à la classe `MonControl` (vue dans les parties précendentes) des fonctions privées ou publiques.

Une fonction privée ne sera pas accessible en dehors du code de la classe alors qu'une fonction publique sera accessible depuis n'importe où ce qui peut entrainer des conflits avec d'autres fonctions de l'application si l'on ne fait pas attention.

Pour une fonction publique

Directement en ajoutant dans le code de la classe `MonControl` :

```
// Classe qui étend la classe abstraite et décrit le custom Control
class MonControl extends AdvancedCustomControl {
    ...
    maFonctionPublique(){
        // Votre code ici
        ...
    }
}
// Créer l'objet control avec l'id 'monControl' qui est le nom de la couche
new MonControl("monControl");
```

Cette fonction sera appelable grâce à `monobjet.maFonctionPublique()` et l'on peut bien sûr y passer des paramètres.

Pour une fonction privée

En dehors du code de la classe `MonControl` et en la déclarant comme une `constante` :

```
// Fonction privée non utilisable en dehors de ce code
const maFonctionPrivée = function(){
    // Votre code ici
    ...
}
// Classe qui étend la classe abstraite et décrit le custom Control
class MonControl extends AdvancedCustomControl {
    ...
    maFonctionPublique(){
        maFonctionPrivée();
        // Votre code ici
        ...
    }
}
// Créer l'objet control avec l'id 'monControl' qui est le nom de la couche
new MonControl("monControl");
```

Cette fonction sera appelable grâce à `maFonctionPrivée()` seulement dans ce bout de code et donc on peut par exemple l'utiliser dans une fonction publique (ici `maFonctionPublique()`).

Ajouter des variables

Pour empêcher de potentiels bugs on peut ajouter à la classe `MonControl` (vue dans les parties précendentes) des variables de classe privée ou publique.

Une variable de classe privée ne sera pas accessible en dehors du code de la classe alors qu'une variable de classe publique sera accessible depuis n'importe où ce qui peut entrainer des bugs (modification involontaire de celle-ci) si l'on ne fait pas attention.

Pour une variable de classe publique

Pour ajouter une variable de classe publique il faut juste ajouter une propriété à l'objet :

```
// Classe qui étend la classe abstraite et décrit le custom Control
class MonControl extends AdvancedCustomControl {
    constructor(id,maVariablePublique){
        // Initialise l'id de l'objet avec le constructeur parent
        super(id);
        // Initialiser maVariablePublique
        this.maVariablePublique = maVariablePublique
        ...
    }
    ...
}
// Initialiser l'objet avec la chaine de caractères "maVariablePublique" dans la variable de classe publique maVariablePublique et l'id de couche "monControl".
new MonControl("monControl","maVariablePublique");
```

Cette variable est accessible à partir du moment où l'on accède à l'objet (dans le navigateur par exemple).

Si on ne souhaite pas forcément donner une valeur à `maVariablePublique` on peut déclarer une valeur par défaut en spécifiant une valeur dans les paramètres de la fonction `constructor()` :

```
// Classe qui étend la classe abstraite et décrit le custom Control
class MonControl extends AdvancedCustomControl {
    // Fonction avec un paramètre ayant une valeur par défaut
    constructor(id,maVariablePublique = "valeurParDefaut"){
        // Initialise l'id de l'objet avec le constructeur parent
        super(id);
        // Initialiser maVariablePublique
        this.maVariablePublique = maVariablePublique
        ...
    }
    ...
}
// Initialiser l'objet avec la chaine de caractères par défaut "valeurParDefaut" dans la variable de classe publique maVariablePublique et l'id de couche "monControl".
new MonControl("monControl");
```

La valeur de `maVariablePublique` sera toujours **« valeurParDefaut »** tant que vous ne spécifiez pas d'autres valeurs.

Pour une variable de classe privée

> **Attention :** La syntaxe suivante ne fonctionne que sur Chrome pour les autres navigateurs remplacez le « # » par un « _ » et vous n'aurez plus besoin de déclarer la variable.

Pour ajouter une variable de classe privée il faut ajouter le **« # »** avant le nom de la variable et la déclarer avant la fonction `constructor()` :

```
// Classe qui étend la classe abstraite et décrit le custom Control
class MonControl extends AdvancedCustomControl {
    // Déclaration de la variable Privée
    #maVariablePrivee;
    constructor(id,maVariablePrivee = "valeurParDefaut"){
        // Initialise l'id de l'objet avec le constructeur parent
        super(id);
        // Initialiser #maVariablePrivee
        this.#maVariablePrivee = maVariablePrivee
        ...
    }
    ...
}
// Initialiser un objet avec la chaine de caractères "maVariablePrivee" dans la variable␣
↪de classe privée #maVariablePrivee et l'id de couche "monControl".
new MonControl("monControl","maVariablePrivee");
```

Si vous voulez quand même pouvoir accéder et modifier la valeur de cette variable en dehors de ce code mais de manière plus sécurisée, il faut déclarer une fonction `get()` pour récupérer la valeur et une fonction `set(valeur)` pour la modifier :

```
// Classe qui étend la classe abstraite et décrit le custom Control
class MonControl extends AdvancedCustomControl {
    // Déclaration de la variable Privée
    #maVariablePrivee;
    constructor(id,maVariablePrivee = "valeurParDefaut"){
        // Initialise l'id de l'objet avec le constructeur parent
        super(id);
        // Initialiser #maVariablePrivee
        this.#maVaribalePrivée = maVariablePrivee
        ...
    }
    // Fonction pour récupérer la valeur de #maVariablePrivee
    getMaVariablePrivee(){
        return this.#maVariablePrivee;
    }
    // Fonction pour modifier la valeur de #maVariablePrivee
    setMaVariablePrivee(valeur){
        this.#maVariablePrivee = valeur;
    }
}
// Initialiser un objet avec la chaine de caractères "maVariablePrivee" dans la variable␣
↪de classe privée #maVariablePrivee et l'id de couche "monControl".
new MonControl("monControl","maVariablePrivee");
```

4.8.3 Interactions customLayer et mviewer

Depuis le customControl il est possible de communiquer et d'interagir avec la carte et d'une façon plus générale avec mviewer. Vous pouvez ainsi mobiliser toutes les méthodes publiques dans votre développement. Pour en savoir plus, consultez, dans la documentation développeur, la partie « *Les fonctions publiques de mviewer* ».

4.9 Custom layer et manipulation des templates Mustache

Avertissement : Niveau nécessaire : Avancé

Cette section vous permettra de :
- comprendre le fonctionnement technique des templates dans mviewer
- être en mesure d'adapter un custom layer pour manipuler et personnaliser le template
- être autonome sur la correction des erreurs entre le custom layer et le template Mustache

4.9.1 Prérequis

- Connaissances en JavaScript
- Maîtriser le fonctionnement des `custom layer` et des templates
- Connaître le fonctionnement de `Mustache.js`

Des sections de la documentation sont dédiées à ces thématiques si vous avez besoin d'en savoir plus. Voici un rappel sur Mustache.js.

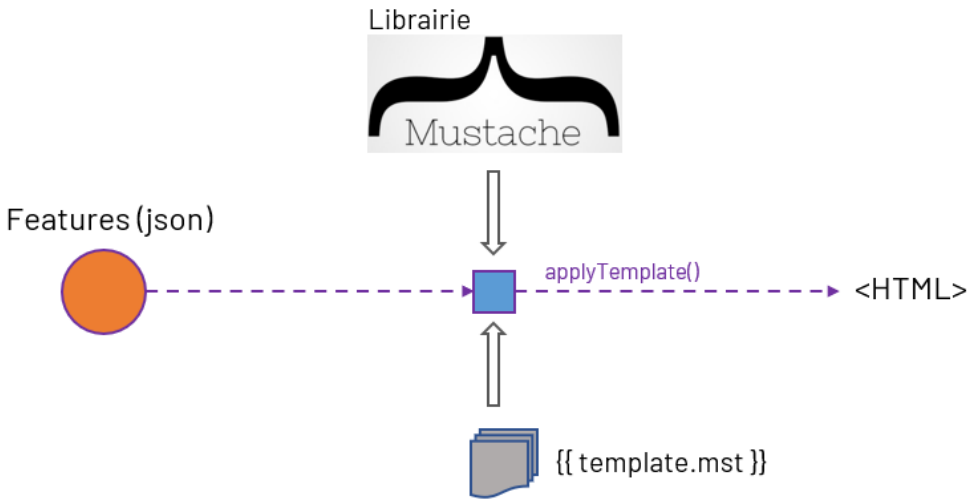
4.9.2 Rappels sur le fonctionnement des templates

Les templates Mustache (format .mst) permettent d'afficher de façon standard ou personnalisée les informations d'une entité géographique visible sur la carte. L'utilisateur cliquera alors sur la carte pour avoir des informations sur l'entité ciblée.

Au clic, le code mviewer réalise une requête (getFeatureInfo) vers le serveur cartographique utilisé (ex. geoserver) afin d'interroger la couche et obtenir les informations sur les entités localisées sous le clic.

Note : Les templates utilisés sont localisés dans le fichier mviewer/js/templates.js.

Si le résultat de la requête contient bien des informations à afficher, alors les informations seront intégrées par ce code mviewer dans le template à droite (right-panel) ou en bas (bottom-panel) au format HTML. C'est via la librairie Mustache.js que nous obtenons du HTML à partir d'un template (.mst) et d'information JSON.



4.9.3 Limite avec un custom layer

Il est possible que le template à afficher se base sur des informations (`JSON`) qui ne sont pas fournies par les informations fournies par le serveur cartographique. C'est le cas par exemple si vous récupérez des données statistiques et que vous souhaitez afficher des données agrégées ou calculées.

Vous rencontrerez ce problème si vous utilisez une représentation type cluster comme ici.

Vous devrez donc organiser avec le JavaScript les informations `JSON` à fournir au template .mst afin de l'afficher correctement.

Pour réaliser cette manipulation de données et préparer le `JSON` à fournir au template, vous pouvez ajouter du JavaScript dans le template via la balise HTML <script>. Cette solution rajoute de la complexité dans le code et alourdir considérablement le fichier .mst.

Une autre solution éxiste via l'utilisons d'une fonction personnalisée que nous allons expliquer dans la section suivante.

4.9.4 Utilisation d'une fonction personnalisée

Ce type de fonction permet de manipuler librement les données que l'on va fournir au template Mustache et qui seront affichées par la suite lors d'un clic sur un objet de la carte.

Cette méthode est très utile pour :

- Afficher les informations d'un cluster (car on cherche à afficher les features qui sont dans la feature Cluster).
- Créer un mustache simple à partir d'informations complexes qui doivent être agrégées par le custom layer.
- Conserver le code JavaScript dans le custom layer pour alléger le fichier .mst.

Vous trouverez ici un (exemple avec un cluster) native dans mviewer.

4.9.5 Exemple d'utilisation

Voici un exemple de méthode (`handle`) pour manipuler les informations au sein d'une ou plusieurs features afin de les afficher dans le template mustache dédié :

```
/**
 * Obligatoire pour que le template mustache fonctionne avec les clusters
 * @param {Array} clusters
 * @param {Object} views
 */
const _handle = function(clusters, views) {
    if (clusters.length > 0) {
    var l = mviewer.getLayer("myclusterlayerid");
    var elements = [];
    var html;
    var panel = "";
    // ici on parcours toutes les features pour accéder au contenu et manipuler les propriétés
    // on pourra par exemple rajouter pour une entité de nouveaux attributs ou d'autres données supplémentaires
    clusters.forEach(c => {
        // ICI ON CREER MANUELLEMENT LES FEATURES POUR MUSTACHE ET LE TEMPLATE POUR V3.5 ET ANTERIEUR
        if (c?.properties) {
            // v<3.5
            elements = elements.concat(
                c.properties.features.map(d =>
                    ({
                        properties: d.getProperties()
                    })
                )
            )
        } else {
            // v>=3.5
            elements = elements.concat(
                c?.getProperties()?.features || c.properties.features
            );
        }
    })
    // Création du HTML à partir du template et des features issues de la manipulation des données
    if (l.template) {
        html = info.templateHTMLContent(elements, l);
    } else {
        html = info.formatHTMLContent(elements, l);
    }
    // force l'affichage selon le mode mobile ou desktop
    if (configuration.getConfiguration().mobile) {
        panel = "modal-panel";
    } else {
        panel = "right-panel"
    }
    var view = views[panel];
```

(suite sur la page suivante)

(suite de la page précédente)

```
    view.layers.push({
        "id": view.layers.length + 1,
        "firstlayer": true,
        "manyfeatures": (elements.length > 1),
        "nbfeatures": elements.length,
        "name": l.name,
        "layerid": "myclusterlayerid",
        "theme_icon": l.icon,
        "html": html
    });
    }
};
// déclarer la fonction _handle pour ce custom layer à sa création
new CustomLayer(ID_LAYER, layer, null, _handle);
```

Notez donc que vous pouvez modifier les propriétés d'une feature avec cette solution ou que vous pouvez totalement créer un nouveau JSON à donner en paramètre d'entrée du template.

Pour rajouter un attribut `randomValue` supplémentaire nous aurions donc pu utiliser quelque chose comme :

```
elements = elements.concat(
    c.properties.features.map(d =>
        ({
            properties: {
                ...d.getProperties(),
                randomValue: math.random()
            }
        })
    )
)
```

L'attribut `randomValue` serait alors utilisable dans le template.mst via `{{randomValue}}`.

CHAPITRE 5

Documentation contribution

Cette partie est dédiée aux personnes qui ont vocation à contribuer au code mviewer.

5.1 Contribuer

Cette partie permet de donner des clés pour contribuer à mviewer.

5.1.1 Présentation générale

Après un déploiement, mviewer permet nativement d'obtenir une carte fonctionnelle.

Cependant vous pouvez ressentir le besoin de modifier mviewer pour créer vos propres cartes, vos propres fonctionnalités et apporter vos propres styles. Pour y arriver vous vous sentez probablement un peu perdu dans cet ensemble de dossiers et fichiers dont vous ne savez pas lesquels il est pertinent de modifier.

Vous trouverez ici une présentation et des recommandations pour créer vos cartes et vos fonctionnalités sans modifier le cœur (ou presque). Vous obtiendrez un mviewer maintenable et vous n'aurez plus l'appréhension de toucher aux mauvais fichiers.

5.1.2 Code style

Avertissement : Ne jamais modifier les fichiers présentés ci-dessous, sauf si la modification des règles de style est demandées par la communauté !

Cette section vous permettra de connaître les règles à utiliser lors de l'écriture du code dans mviewer. Ces règles permettent d'assurer que les contributions sont uniformes (e.g indentations, longueur de lignes, etc.) et le code de meilleur qualité.

Les règles de style permettent également d'adopter les bonnes règles pour tous afin de faciliter la comparaison des fichiers et les contributions.

1. Formattage dans VS Code

Dans VS Code, vous trouverez un fichier *.vscode/settings.json*.

Ce fichier contient des règles de paramétrage de VS Code qui peuvent être réutilisées par défaut par tous les développeur qui utilisent VS Code avec mviewer.

Ce fichier *settings.json* permet de définir le formatter par défaut par type de language.

Il permet aussi d'utiliser le formattage automatique à la sauvegarde :

```
"editor.formatOnSave": false
```

Dans VS Code, si vous n'utilisez pas ce fichier, vous pouvez formatter à la sauvegarde automatiquement via les préférences VS Code décrites plus bas.

2. Prettier

Mviewer utilise Prettier.

Vous pouvez utiliser Prettier en ligne si vous ne vous voulez pas utiliser node.js en suivant les actions définies ici

Vous pouvez également utiliser l'éditeur Visual Studio Code et son plugin Prettier - Code formatter (npm obligatoire).

— Installation

Via *npm* (voir ici pour installer *npm*) :

```
npm install
```

A l'ouverture du projet mviewer avec VS Code, le plugin détectera automatiquement les fichiers *.prettierrc* et *.prettierignore*. Les règles seront donc automatiquement détectées et vous pourrez alors utiliser *Prettier* pour appliquer les règles de style au code mviewer.

— Fichier .prettierrc

C'est le fichier de configuration de Prettier (format JSON). Il contient toutes les règles à utiliser par Prettier.

Ce fichier situé à la racine contient les règles que toute la communauté utilise.

Pour plus d'informations sur la configuration :

https://prettier.io/docs/en/configuration.html

— Fichier .prettierignore

Ce fichier permet d'ignorer certains fichiers ou dossier du code mviewer.

```
# Ignore artifacts:
build
coverage

# Ignore all HTML files:
*.html

# Ignore folders :
**/lib
```

Pour plus d'informations, sur ce fichier :

https://prettier.io/docs/en/ignore.html

3. Formatter vos fichiers avec Prettier

Avec VS Code, le formattage peut être réalisé à la sauvegarde ou à la demande.

— Formattage à la sauvegarde

Vous devez paramétrer votre éditeur pour formatter à la sauvegarde ou utiliser le fichier *.vscode* décrit plus haut.

Avec VS Code, aller dans Fichier > Préférences > Paramètres et rechercher *format on save*. Cocher alors *Editor :Format on save* pour activer le formattage à la sauvegarde.

— Formattage à la demande

Avec VS Code, réaliser un clic droit dans le fichier et cliquer sur *mettre en forme le document avec… `. Sélectionner alors `Prettier* dans la liste .

— Fromattage via *npm*

Le fichier */mviewer/package.json* contient une commande *pretty*. Cette commande va effectuer un formattage sur les fichiers *.js* et *.json* des répertoires */js*, */demo*, */customcontrols*, */customlayers*.

```
"pretty": "prettier --write \"./mviewer.i18n.json\" \"./js/**/*.{js,json}\" \"./demo/**/
↪*.{js,json}\" \"./customcontrols/*.{js,json}\" \"./customlayers/*.{js,json}\""
```

Elle s'exécute dans le répertoire */mviewer* comme ceci :

```
npm install
npm run pretty
```

5.1.3 Le cœur de mviewer

Qu'est-ce que c'est ?

C'est l'ensemble des fichiers et dossiers présents nativement sur la page GitHub mviewer.

Quand puis-je le modifier ?

Vous devez éviter de modifier les fichiers natifs du mviewer. En effet, modifier ces fichiers vous empêchera de mettre à jour facilement votre déploiement de mviewer pour prendre en compte une nouvelle version officielle.

Néanmoins, vous pouvez être amené à modifier ces fichiers principalement pour contribuer au développement de l'outil :

— Vous détectez un bogue ou un comportement suspect et vous le corrigez.
— Vous créez une évolution sur le cœur (une nouvelle fonctionnalité).
— Vous créez une amélioration du code existant.

Dans chacune de ces situations l'intervention sur le cœur de mviewer doit être justifiée par une issue sur GitHub.

5.1.4 Les autres fichiers

Pour vos modifications et l'organisation de vos fichiers, nous recommandons de suivre la page « *Organisation des fichiers de carte* ».

5.1.5 Proposer une modification

Pour proposer une correction d'anomalie ou une évolution, vous devez suivre ces étapes :

— Créer une issue sur Github en suivant la page *Créer une issue*.
— Faire un fork du code (si ce n'est pas encore fait) en suivant la page *Récupérer les sources (fork)*.
— Créer une branche portant le numéro de l'issue (ex : issue-2287).
— Apporter vos modifications sur cette branche.
— Partager cette branche via l'issue pour que les autres puissent tester et obtenir des conseils ou des avis.
— Réaliser une pull request via GitHub en suivant la page *Pull Request*.

La pull request permettra d'importer votre modification dans le code natif. Vous diposerez alors de votre modification de manière native sans vous en préoccuper ultérieurement.

5.1.6 Documentation

Pour mieux contribuer :

1. Première contribution
2. Comment contribuer

5.2 Contribuer à la documentation - Readthedoc

Avertissement : La documentation est désormais dans le répertoire docs du dépôt principal **mviewer** et non plus dans le dépôt **mviewer.doc**.

Participez à l'amélioration de la documentation en ligne de mviewer.

Respectez le processus de contribution décrit dans la section « *Proposer une modification* ».

Sources

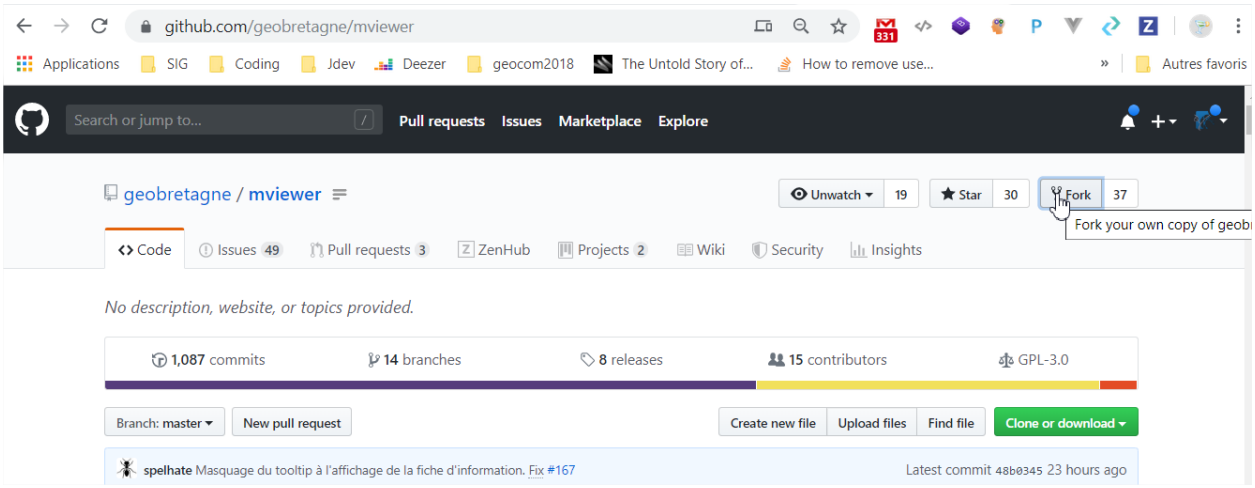
Les sources de la documentation sont disponibles sur GitHub mviewer/mviewer, dans le répertoire docs.

Créez une issue sur GitHub

Voir la page « *Créer une issue* » pour proposer une modification.

Fork

Pour apporter des modifications sur la documentation, vous devrez réaliser un fork vers votre compte organisation de Github. Pour réaliser un fork, dirigez-vous vers l'exemple dans la section « *Récupérer les sources (fork)* ».



5.2.1 Déployer la documentation en local

Prérequis

— Vous devez disposez d'un serveur web type Apache2

Si vous ne connaissez pas Apache, vous pouvez installer facilement XAMPP *en suivant l'explicatif « Installer XAMPP (windows) » plus bas.*

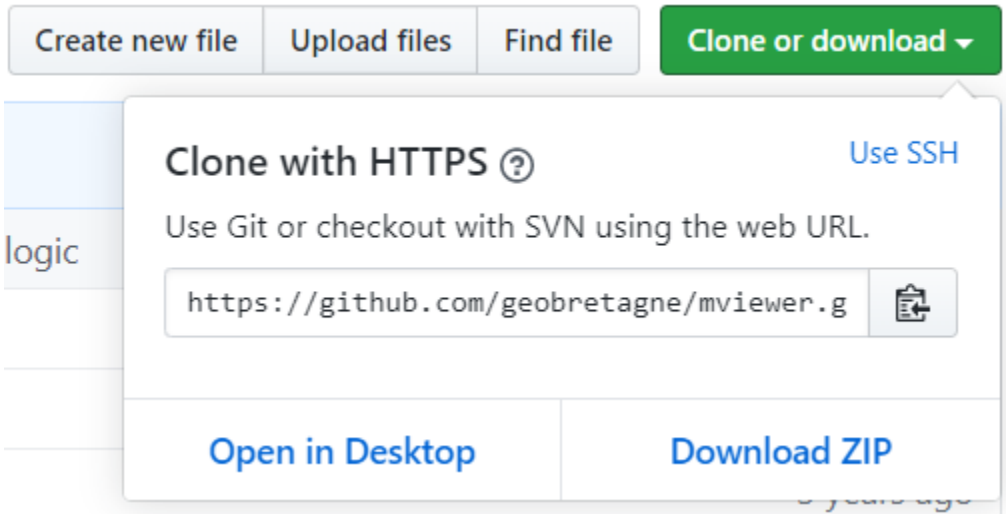
— Vous devez disposer des droits sur votre ordinateur pour installer python et disposer de pip.

— Avoir réalisé un fork du repository mviewer/mviewer.

— Avoir réalisé un clone de votre fork vers un répertoire local de votre ordinateur :

```
cd /home/user/pierre/git/
git clone https://github.com/mon_compte_github/mviewer.git
```

L'URL de votre fork est disponible en cliquant sur « Clone or download »



— Avoir installé python sphinx en suivant la page sphinx :

```
// debian
apt-get install python3-sphinx
```

— Disposer de pip (debian) :

```
// debian
sudo apt install python-pip
pip --version
```

— Installer le package stemmer (si manquant au build) :

```
// Debian
sudo apt-get install python-stemmer

// pip
pip install PyStemmer
```

Actions

— Positionnez-vous dans votre dossier mviewer issu du clone :

```
cd /home/user/pierre/mviewer
```

— Vous pouvez apporter des modifications dans le dossier « docs » :

```
cd /home/user/pierre/mviewer/docs
```

— Pour rajouter des parties et sous-parties dans le menu de gauche, il vous faudra modifier le fichier index.rst :

```
/home/user/pierre/mviewer/docs/index.rst
```

— Rajouter par exemple une partie « Nouvelle partie » en respectant cette syntaxe :

```
Nouvelle partie
-----------------------------------------------

Nouvelle partie pour tester.

.. toctree::
:hidden:
:maxdepth: 1
:caption: Ici le Titre

doc_test/introduction
doc_test/sous_partie1
doc_test/sous_partie2
```

— Comme décrit dans la syntaxe précédente, vous devez rajouter un dossier « doc_test » (où « doc_test » est le nom que vous avez choisis) :

```
/home/user/pierre/mviewer/docs/doc_test
```

— Dans ce dossier, rajoutez les fichiers comme décrits dans l'arborescence :

```
../mviewer/docs/doc_test/introduction.rst
../mviewer/docs/doc_test/sous_partie1.rst
../mviewer/docs/doc_test/sous_partie2.rst
```

— Inspirez-vous de l'existant pour comprendre l'organisation des fichiers avec index.rst.
— Vous devrez écrire selon une syntaxe particulière. Recherchez dans les fichiers et dans les exemples de cette page pour vous aider.

5.2.2 Ajouter des blocs de code

Utilisez la syntaxe suivante (respectez les sauts de ligne) :

```
mon text::

mon bloc de code

Suite du texte.
```

5.2.3 Ajouter des puces

Utilisez la syntaxe suivante (respectez les sauts de ligne) :

```
Voici une liste :

- premier tiret
- deuxième tiret

Suite du texte.
```

5.2.4 Rajouter des images

— Ajoutez un dossier dans ../docs/_images tel que :

```
/home/user/pierre/mviewer/docs/_images/doc_test/
```

— Ajoutez vos images dans ce dossier et renseignez le chemin de l'image à afficher dans le code tel que :

```
Voici une image :

.. image:: ../_images/doc_test/image1.png
           :alt: description de l'image
           :align: center

Suite du texte.
```

5.2.5 Liste numérotée

Utilisez la syntaxe suivante (respectez les sauts de ligne) :

```
Une liste avec des numéros :

#. Mon premier
#. Mon second
#. ...

Suite du texte.
```

5.2.6 Référencer une page

— Pour créer un point de référence .._reference : que l'ont peut citer comme lien depuis n'importe quelle page (lien interne).
— Utilisez les titres pour afficher le texte à afficher comme référence :

```
.. _reference:

Page de référence
-----------------
```

— Appelez la référence affichera « Voir la Page de référence » :

```
Voir la ":ref:`reference`"
```

— « Page de référence » sera cliquable pour s'y rendre.

5.2.7 Lien, hyperlien

Utilisez la syntaxe suivante (respectez les sauts de lignes) :

```
Ceci est un `lien cliquable <https://github.com/mviewer/mviewer>`_
```

5.2.8 Construire et déployer la documentation

— Les sources de la documentation sont localisées dans votre dossier git/mviewer/docs créé par le clone (voir plus haut) :

```
/home/user/pierre/mviewer/docs
```

— Nous voulons que notre documentation soit construite (build) dans le dossier :

```
/var/www/mviewer-doc/
```

— Si vous avez utilisé XAMPP (voir « xampp » : :), le dossier cible où sera construite la documentation sera (sous windows) :

```
C:\xampp\mviewer-doc\
```

— Nous avons ensuite à passer la commande :

```
sphinx-build -b html home/user/pierre/mviewer/docs /var/www/mviewer-doc/
```

— La documentation est maintenant dans le dossier de notre choix :

```
/var/www/mviewer-doc/

ou pour XAMPP:

C:\xampp\mviewer-doc\
```

— Déployez la documentation créée via la commande avec Apache2 si vous avez d'autres chemins d'accès.
— Avec XAMPP Accédez à la documentation via localhost/mviewer-doc (mviewer-doc étant le nom de dossier que vous avez utilisé).

Faites votre pull request

Retrouvez la procédure décrite dans la partie « *Pull Request* ».

5.2.9 Installer XAMPP (windows)

— Téléchargez XAMPP
— Lancez XAMPP pour afficher l'interface d'administration (GUI)
— Sur la ligne du module « Apache », à droite cliquez sur « Start » au sein des actions
— « Apache » doit passer en vert dans la colonne « Module »
— Cliquez sur « Explorer » dans la colonne tout à droite
— Une fenêtre d'exploration s'affiche (par défaut vers C :xampp)
— Rechercher « htdocs » dans la fenêtre d'exploration
— Créez un dossier « mviewer-doc »

C'est dans le dossier « mviewer-doc » que sera déployée la documentation après la phase de build (voir plus haut).
— Accédez au dossier avec votre navigateur via l'URL :

```
localhost/mviewer-doc
```

5.2.10 Documentation

Pour obtenir plus d'information sur la syntaxe et sphinx :

1. Sphinx
2. Sphinx syntaxe tutoriel
3. Sphinx syntaxe infos
4. Sphinx example syntaxe

5.3 Créer une issue

Quelques règles pour proposer à la communauté vos modifications, vos nouvelles fonctionnalités et vos corrections sur mviewer.

5.3.1 Règle générale

Toute modification du cœur de mviewer **doit être proposée** à l'aide d'une nouvelle issue sur le Github mviewer/mviewer.

5.3.2 Pourquoi créer une issue ?

D'abord, pour pouvoir intégrer votre modification dans le cœur et maîtriser la maintenance des sources. Si vous modifiez le cœur mviewer de votre instance et que vous souhaitez réaliser une mise à jour, vous aller devoir gérer des conflits. Ce qui ne sera pas le cas si vos modifications sont déjà dans le code de la nouvelle version.

Ensuite, pour vous éviter de réaliser une demande de contribution qui ne sera pas acceptée lors de la revue :
— Le code proposé ne respecte pas le formatage initial.
— Vous avez réalisé une contribution qui existe déjà.
— Vous proposez un code avec un niveau de complexité trop élevé.
— Le fonctionnement ou le code doivent être optimisés.
— Une partie de ce que vous proposez fonctionne mais ne peut pas être intégrée (dépendances externes non maintenues, mauvaise utilisation du langage, etc…).
— Votre contribution ne peut pas être intégrée dans le cœur, car vous seul en avez l'utilisation.

Il y a également d'autres bonnes raisons de créer une issue pour contribuer :

— Partager votre idée et vous aurez probablement des personnes pour la réaliser (intéressant !).
— Avertir d'autres contributeurs que vous travaillez sur un sujet et mutualiser le travail.
— Eviter que plusieurs contributeurs travaillent en parallèle sur la même chose.
— Permettre d'obtenir des informations pour réaliser ce que vous souhaitez.
— Obtenir un avis ou de l'aide sur une idée de contribution et la manière de la réaliser.

5.3.3 Où créer une issue ?

Il faudra vous rendre sur la page GitHub mviewer. Vous devez disposer d'un compte et vous connecter.

Cliquer ensuite sur « Issue » puis, « New issue ».

5.3.4 Comment créer une issue ?

1 - Donner un titre

Le titre doit être court mais explicite. Le titre doit permettre de retrouver facilement votre issue à la lecture.
— Bon : « Ajouter un boutton "Nouvelle issue" »
— Mauvais : « Ajouter un nouveau boutton pour créer une issue aujourd'hui manquant »

2 - Ajouter la description

Il sera obligatoire de formater votre texte grâce au Markdown afin de faciliter la lecture. Notamment pour citer ou mettre en évidence des morceaux de code. Autrement, votre texte risque d'être vite illisible. Voici un exemple de syntaxe Makrdown.

La description peut contenir des images, des liens, des citations ou des morceaux de code.

3 - Surveiller les réponses

Soyez animateur de votre discussion et participez dans les réponses. Sinon, elle sera vite oubliée… Pour citer une personne, utilisez @identifiant_Github. Elle sera avertie par email de votre message.

5.3.5 Qui peux créer une issue ?

Tout le monde peut créer une issue à condition de disposer d'un compte GitHub (gratuit).

5.4 Travailler avec Git et GitHub

Entrenez votre mviewer et facilitez vos contributions en travaillant avec les outils Git et Github.

5.4.1 Présentation générale

GitHub est incontournable pour contribuer et découvrir la communauté mviewer. C'est là où vous proposerez vos idées, obtiendrez des informations et les dernières versions mviewer.

Grâce à Git vous pourrez réaliser vos contributions ou obtenir le code source dans votre environnement de travail.

Git propose une multitude d'actions pour s'interfacer avec GitHub. Ces actions vous seront plus qu'utiles pour contribuer et maintenir votre mviewer à jour. Nous expliquerons ici l'utilité et les manipulations utiles telles qu'un fork, rebase, commit, pull.

5.4.2 Présentation GitHub

GitHub est une plate-forme en ligne de contrôle de version et de collaboration pour le développement. GitHub facilite le travail en équipe via une interface web qui permet d'accéder au dépôt de code Git. GitHub fournit des outils de gestion pour la collaboration.

5.4.3 Présentation Git

Git est un outil libre de gestion de version décentralisée pour tout type de projet. Il permet de réaliser des développements sur votre propre dépôt (dépôt = répertoire). Git facilite ensuite la mise en commun du code entre les différents dépôts.

Git historise toutes les modifications afin d'identifier les nouveautés et permettre de revenir dans n'importe quelle version précédente.

5.4.4 Présentation Git Flow

Qu'est-ce que c'est ?

Git Flow est un workflow (organisation) permettant d'établir une stratégie de base sur la création des branches. Git Flow permet de gérer le travail collaboratif pour gérer les issues (bugs), les features (nouvelles fonctionnalités) et les releases (versions).

C'est le workflow qui a été adopté par les contributeurs Mviewer.

Principes de base

Pour simplifier, le fonctionnement est basé sur 2 branches : master et develop.

Si plusieurs personnes travaillent sur ces deux branches, l'historique des réalisations (commits) peut vite devenir illisible et les modifications risquent de se court-circuiter (conflits).

L'idée est donc de créer des sous-niveaux de branches :

— Les branches **features-xxxx** permettent de travailler sur des nouvelles fonctionnalités. Elles sont crées directement à partir de la branche develop et une fois le travail fini, fusionnées vers la branche develop.
— Les branches **release-xxxx** permettent de faire une mise à jour de la branche master à partir de la branche develop.
— Les branches **hotfix-xxxx** permettent de publier rapidement (hot) une correction (fix) depuis la branche master. Ces branches seront ensuite fusionnées vers la branche master et develop.

Voici quelques références externes pour vous aider à mieux comprendre Git-Flow :

1. Avoir une bonne stratégie avec Git-flow
2. Git-Flow est-il le workflow dont j'ai besoin ?
3. Tutoriel vidéo Grafikart
4. Git-Flow workflow tutorial by Atlassian

5.4.5 Fork et Pull

Nous avons déjà montré l'utilité et l'utilisation d'un **fork** et d'un **pull** sur la page « *Bien commencer avec mviewer* ».
— Le fork est décrit à la section « *Récupérer les sources (fork)* »
— Le pull est utilisé à la section « *Récupérer les nouveautés de la branche* »

5.4.6 Rebase

Lorsque vous faites des modifications via des commits sur votre branche de travail d'autres contributeurs modifient le code mviewer avec de nouveaux commits.

Le code de mviewer dans le GitHub GéoBretagne n'a pas connaissance de vos modifications. Tant que vous n'avez pas mis à jour votre branche, votre code ne contient pas les modifications faites sur le GitHub mviewer GéoBretagne (upstream).

Les deux codes ont donc des nouveautés. On peut dire que les branches ont divergé.

Vous devez alors intégrer les nouveautés de mviewer dans une branche de travail contenant vos modifications. Pour rappel, cette branche ne doit pas jamais être la branche master.

Pour cela, nous allons en premier reprendre tous les nouveaux commits du mviewer natif (GéoBretagne) en mettant à jour votre fork (branche master). Ensuite, nous mettrons à jour votre branche de travail depuis la branche master de votre fork.

Dans la pratique, nous placerons les nouveaux commits de la branche master du fork dans l'arbre de commits de votre branche de travail via un **rebase**.

Mise à jour du fork

Reprenez dans l'ordre les étapes « *Définir un upstream* » et « *Mettre à jour votre fork - master* ».

Réalisez ensuite la procédure suivante.

Que fait un Rebase ?

— Git va reprendre le dernier commit commun entre votre branche de travail à mettre à jour et la branche qui contient les nouveautés (master)
— Git replacera ensuite vos commits et les nouveaux commits dans l'ordre chronologique

Vous disposerez donc des nouveaux commits et de vos propres commits.

Comment faire ?

— Faites une copie de votre branche (optionnel mais conseillé) en créant une nouvelle branche à partir de votre branche de travail.
— Si votre branche s'appelle par exemple « RM-work », lancez la commande de rebase de la branche master (fork à jour) vers votre branche à mettre à jour (RM-work) :

```
git rebase origin/master RM-work
```

```
gaetan@DESKTOP-1HU3HJF MINGW64 /c/xampp/htdocs/mviewer_rm_fork (RM-work)
$ git rebase origin/master RM-work
```

— Vous verrez la liste des commits dérouler les messages des commits un à un.
— Vous aurez probablement un conflit. Le processus sera donc stoppé mais pas abandonné.
— Si vous souhaitez abandonner lancer la commande(*) :

```
git rebase --abort
```

```
To abort and get back to the state before "git rebase", run "git rebase --abort".
```

— Si vous souhaitez ignorer le conflit (déconseillé !) :

```
git rebase --skip
```

— Nous conseillons de résoudre le conflit. Git vous indique un nom de fichier en conflit (ici indiqué index.html). C'est qu'il n'a pas réussi tout seul à intégrer les modifications sans perdre votre code actuel comme indiqué :

```
.. image:: ../_images/contrib/filetoresolverebase.png
```

alt
git abort

align
center

— Ouvrez ce fichier avec un éditeur classique. Vous observerez que Git a inséré des caractères spéciaux pour nous permettre d'identifier les lignes en conflit :

```
// je suis une pomme
var type = "Pomme"
<<< HEAD
// nouveau code
var test = "je suis rouge";
==========
// code actuel
var test = "je suis verte";
var taille = 12;
>>>>>
var region = "Normandie";
```

— Vous pouvez garder le nouveau code entrant entre <<< HEAD et === ou bien garder le code actuel entre ==== et >>> ou bien garder les deux.
— Pour cela, vous allez modifier à la main le fichier en supprimant les caractères <<< HEAD et ==== et >>>> ainsi que les lignes indésirables.
— Nous avons maintenant ce contenu :

```
// voici ma couleur
var type = "Pomme"
var test = "je suis rouge";
var taille = 12;
var region = "Normandie";
```

— Sauvegardez votre fichier
— Indiquez à Git que vous avez géré le conflit :

```
git add /chemin/vers/le/fichier/index.html
```

```
gaetan@DESKTOP-1HU3HJF MINGW64 ~/Documents/git/mviewer_rm_audit/mviewer (RM-work|REBASE 12/41)
$ git add index.html
```

— On contrôle que le fichier est marqué comme « modified » avec la commande :

```
git status
```

```
gaetan@DESKTOP-1HU3HJF MINGW64 ~/Documents/git/mviewer_rm_audit/mviewer (RM-work|REBASE 12/41)
$ git status
rebase in progress; onto d7e59df
You are currently rebasing branch 'RM-work' on 'd7e59df'.
  (all conflicts fixed: run "git rebase --continue")

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.html
```

— Indiquez à git de poursuivre le rebase comme décrit dans le message :

```
git rebase --continue
```

```
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
```

— Vous verrez d'autres commits listés et vous aurez probablement d'autres conflits. Répétez les opération précédentes pour bien tous les gérer.
— Lorsque le rebase est terminé vous n'aurez pas de message spécifique qui vous l'indiquera. Vous pourrez cependant voir que les derniers commits ont bien été appliqués.

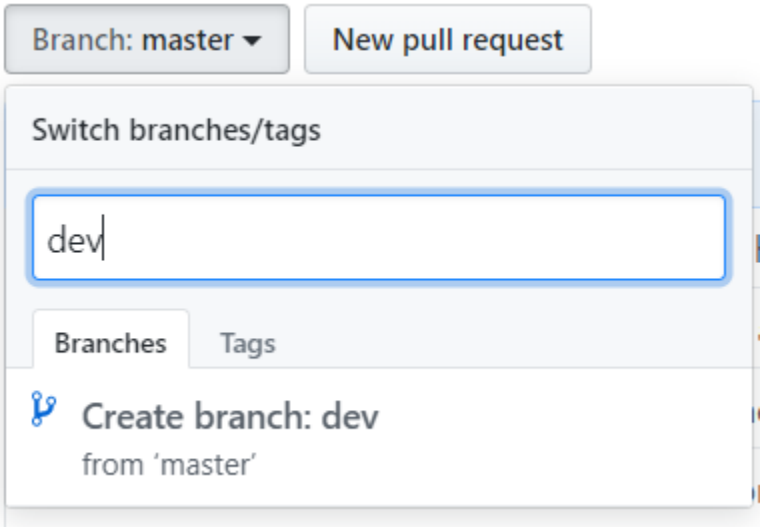
```
gaetan@DESKTOP-1HU3HJF MINGW64 ~/Documents/git/mviewer_rm_audit/mviewer (RM-work|REBASE 39/41)
$ git rebase --continue
Applying: Fix seach js file and some path
Applying: Fix path for templates
Applying: Move lib, JS, CSS files to apps/public

gaetan@DESKTOP-1HU3HJF MINGW64 ~/Documents/git/mviewer_rm_audit/mviewer (RM-work)
$
```

Vérifier le résultat du rebase

Nous devons absolument vérifier que le rebase a pris en compte les commits natifs issus de GéoBretagne et vos commits de travail.

— Aller sur la page GitHub mviewer/mviewer
— Ouvrez la page des commits

— Vérifiez dans la liste déroulante que vous êtes bien sur la branche master



— Observez les derniers commits, la date et le titre
— Nous allons maintenant vérifier que ces commits sont bien dans notre historique de commits après le rebase.
— Affichez l'historique des commits dans le terminal Git :

```
git logs
```

```
gaetan@DESKTOP-1HU3HJF MINGW64 ~/Documents/git/mviewer_rm_audit/mviewer (RM-work)
$ git log
commit c71a896b154863f02cfa50a511d4986ffdbe1edb (HEAD -> RM-work)
Author: Gaetanbrl <gaetan.bruel@jdev.fr>
Date:   Tue Nov 12 10:44:01 2019 +0100

    Move lib, JS, CSS files to apps/public
```

— Affichez la liste les commits présente sur la page des commits.
— Vous devez les retrouver dans la liste des commits de la branche dans laquelle vous venez de réaliser votre rebase.
— En cas de doute sur la gestion de certains conflits, vérifiez les fichiers visuellement et réalisez des tests dans vos applications.
— Si tout vous semble correct, vous avez bien récupéré les modifications et votre arbre de commits est à jour (ainsi que votre code).

Transmettre du local vers la branche

Actuellement, le rebase a apporté des modifications sur votre ordinateur. Mais le code en ligne (GitHub) n'a pas changé. Vous devez pousser les modifications vers la branche distante.

— Lancez la commande suivante pour transmettre le travail du rebase à la branche distante (en ligne et visible sur GitHub) (**) :

```
git push -f
```

```
gaetan@DESKTOP-1HU3HJF MINGW64 ~/Documents/git/mviewer_rm_audit/mviewer (RM-work)
$ git push -f
Enumerating objects: 542, done.
Counting objects: 100% (542/542), done.
Delta compression using up to 8 threads
Compressing objects: 100% (130/130), done.
Writing objects: 100% (531/531), 264.08 KiB | 13.90 MiB/s, done.
Total 531 (delta 351), reused 455 (delta 321)
remote: Resolving deltas: 100% (351/351), completed with 9 local objects.
To https://github.com/sigrennesmetropole/mviewer.git
 + eb62455...c71a896 RM-work -> RM-work (forced update)
```

— Ouvrez la page des commits de votre branche de travail (ex pour la branche dev :) et vérifiez le succès de l'opération.
— Supprimer ensuite la branche de sauvegarde si tout vous semble bon.

(*) Avec –abort Il faudra tout reprendre tout le rebase depuis le début si vous arrêter et décidez de recommencer.

(**) *Avec -f, cela indique un push forcé afin de réécrire en force l'historique des commits sur la branche distante. Il vaut mieux maîtriser ce que l'on pousse et contrôler votre code en local avant.*

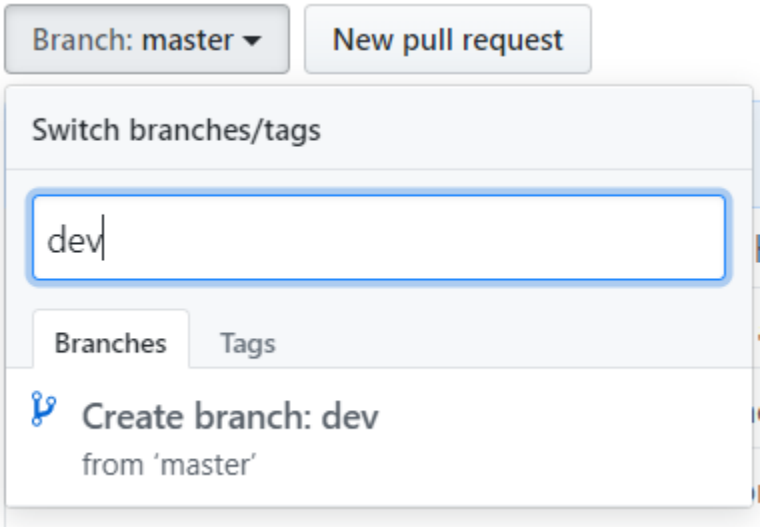
5.4.7 Pull Request

Une pull request ou « demande de tirage » réalise une demande pour que les modifications d'une branche intègre une autre branche.

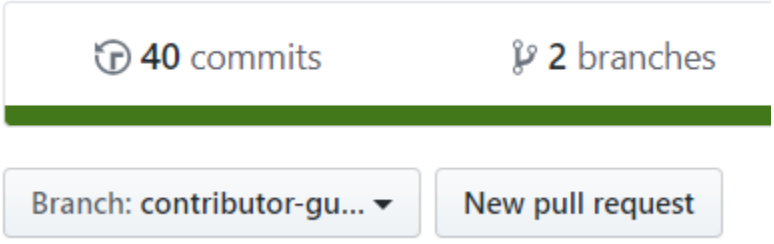
Vous devez créer une pull request pour apporter une contribution de votre branche au sein de votre repository mviewer vers le repository mviewer/mviewer.

Pour réaliser une pull request, dirigez-vous sur votre fork GitHub :

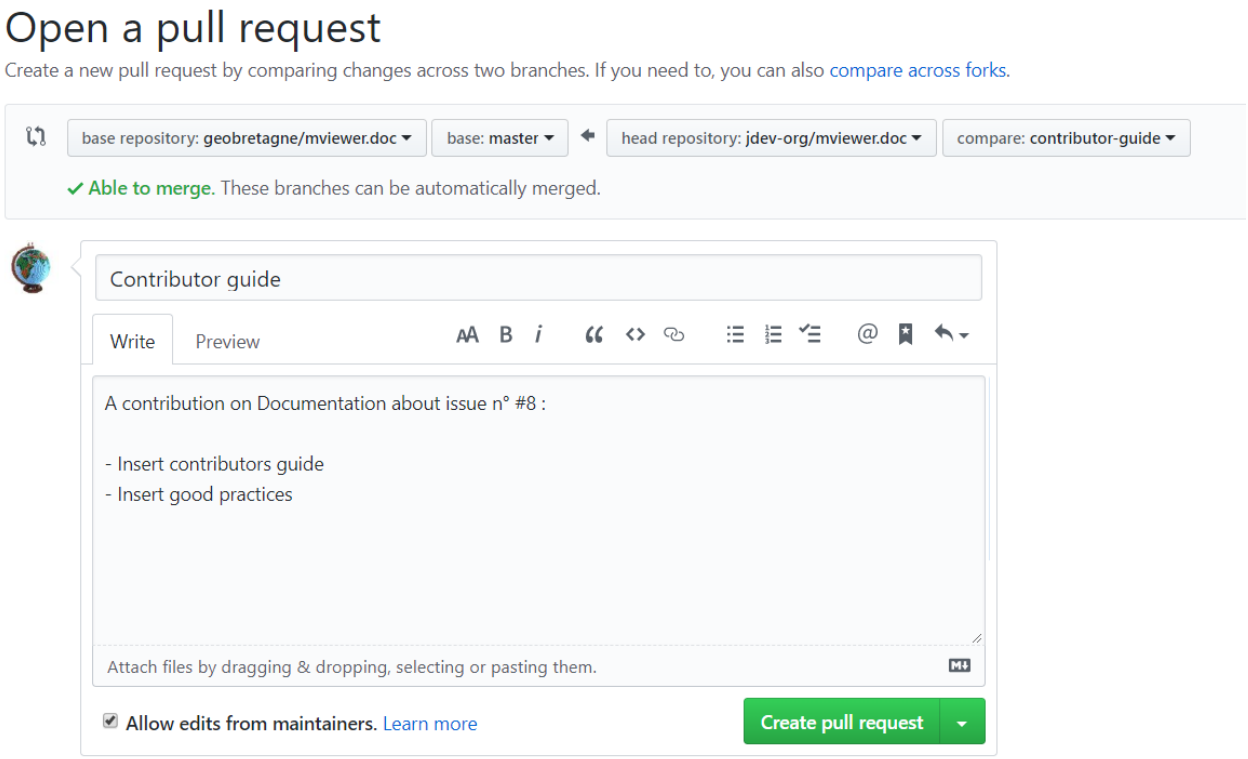
— Sélectionnez votre branche qui contient vos modifications à apporter en contribution



— Cliquez sur « New pull request »



— Ajouter un titre simple mais distinctif et parlant
— Ajouter un explicatif, avec de préférence le lien vers l'issue concernée
— Cliquez sur « Create pull request »



— Vous pourrez accéder à la pull request et discuter via le volet dédié du repository mviewer/mviewer.

Votre pull request sera revue et vous aurez très certainement un retour pour réaliser des ajustements ou bien vous notifier que votre demande est acceptée.

N'hésitez-pas à laisser un message dans la pull request pour relancer la communauté si vous n'avez pas de réponse dans un délai raisonnable.

5.4.8 Cherry-pick

Si vous ne souhaitez reprendre qu'un seul commit d'une autre branche ou d'un autre repository, vous pouvez utilisez le cherry-pick. C'est un report manuel avec Git d'un commit d'une branche vers une autre branche, peu importe le repository.

Pour peu de commits, cette solution peut paraître plus simple que d'utiliser la technique de rebase.

Exemple avec un numéro de commit 235c47f à récupérer sur une branche nommée « dev » :

```
cd /home/user/jean/git/mviewer
git checkout dev
git cherry-pick 235c47f
```

Parcourez la documentation plus pas pour plus de détails.

5.4.9 Contribution

Pour contribuer, nous cous recommandons de suivre la documentation « *Contribuer* ».

5.4.10 Documentation

1. OpenClassrooms
2. Débuter avec Git
3. Mémo Git
4. Cherry-pick

5.5 Faire une release de Mviewer & MviewerStudio

Les versions des projets Mviewer et MviewerStudio sont liées. Il faut donc faire une release des deux applications.

Il n'y a pas de système de build pour ces applications qui se veulent simples de mise en place. Il n'y a donc pas de commande permettant de faire la release simplement.

Vous devez avoir des droits spécifiques sur le repository pour pouvoir faire la release.

5.5.1 Release Mviewer

Commencez par vérifiez que toutes les issues et PR « closed » depuis la dernière release sont bien liées à un milestone.

Créez ensuite un nouveau milestone pour la future version et déplacez toutes les issues souhaitées et non « closed » à l'intérieur. https://github.com/mviewer/mviewer/milestones

Puis fermez le milestone de la version releasée.

Modifiez la version de l'application sur la branch developp dans le fichier (en supprimant -snapshot) : https://github.com/mviewer/mviewer/blob/develop/js/configuration.js#L11

Puis faire une pull request entre la branche develop et la master à un moment stabilisé. Testez le fonctionnement de l'application avec la PR et validez cette pull request.

Pour plus d'informations sur les branches, Mviewer suit ce type de modèle : https://nvie.com/posts/a-successful-git-branching-model/

Une fois le merge effectué sur master, il vous faudra suivre ces indications pour créer la nouvelle release sur la page de release Mviewer.

Après la release, dans la branche develop, retournez modifier le numéro de version pour augmenter d'une version et ajouter -snapshot à la fin.

5.5.2 Release MviewerStudio

La marche à suivre devrait être la même, mais pour l'instant la branche develop n'est pas forcement à jour. A ce jour un travail est encore à faire.

Modifiez le numéro de version sur la branch master directement : https://github.com/mviewer/mviewerstudio/blob/master/js/mviewerstudio.js#L3

Vérifiez que toutes les issues et PR « closed » depuis la dernière release sont bien liées à un milestone.

Créez ensuite un nouveau milestone pour la future version et déplacez toutes les issues souhaitées à l'intérieur : https://github.com/mviewer/mviewerstudio/milestones

Puis fermez le milestone de la version releasée.

Il vous faudra ensuite suivre ces indications pour créer la nouvelle release sur la page de release Mviewer Studio.

Après la release, retournez modifier le numéro de version pour augmenter d'une version et ajouter -snapshot à la fin.

Il n'y a pas de versionning de la documentation actuellement, c'est un élement dont il faudrait discuter.

CHAPITRE 6

Auteurs et licence

Cette documentation a été réalisée par :

- l'équipe « mviewer » *(Région Bretagne)*
- Gwendall Petit *(Lab-STICC – CNRS UMR 6285)*

Sauf indication contraire, cette documentation est sous licence **Creative Commons Attribution - Non Commercial - ShareAlike 4.0** (CC-BY-NC-SA) .

Index

C

clickedCoordinates()
    fonction de base, 110

F

fonction de base
    clickedCoordinates(), 110
    getActiveBaseLayer(), 108
    getLayer(), 108
    getLayersAttribute(), 109
    getMap(), 109
    orderLayerByIndex(), 110
    orderLegendByMap(), 109
    orderTopLayer(), 110
    removeAllLayers(), 109
    reorderLayer(), 109
    setBaseLayer(), 108
    setLegendLayerPos(), 110
    showLayersByAttrOrder(), 110
    showLocation(), 109
    toggleLayer(), 109
    tr(), 109
    zoomToLocation(), 109

G

getActiveBaseLayer()
    fonction de base, 108
getLayer()
    fonction de base, 108
getLayersAttribute()
    fonction de base, 109
getMap()
    fonction de base, 109

O

orderLayerByIndex()
    fonction de base, 110
orderLegendByMap()
    fonction de base, 109
orderTopLayer()
    fonction de base, 110

R

removeAllLayers()
    fonction de base, 109
reorderLayer()
    fonction de base, 109

S

setBaseLayer()
    fonction de base, 108
setLegendLayerPos()
    fonction de base, 110
showLayersByAttrOrder()
    fonction de base, 110
showLocation()
    fonction de base, 109

T

toggleLayer()
    fonction de base, 109
tr()
    fonction de base, 109

Z

zoomToLocation()
    fonction de base, 109